



# תקשורת

## מטלה 1



# WIRESHARK

## INTRO

**1) List 3 different protocols that appear in the protocol column in the unfiltered packet-listing window in step 7 above.**

**2) How long did it take from when the HTTP GET message was sent until the HTTP**

**OK reply was received? (By default, the value of the Time column in the packet-listing window is the amount of time, in seconds, since Wireshark tracing began.)**

**To display the Time field in time-of-day format, select the Wireshark View pull-down menu, then select Time Display Format, then select Time-of-day.)**

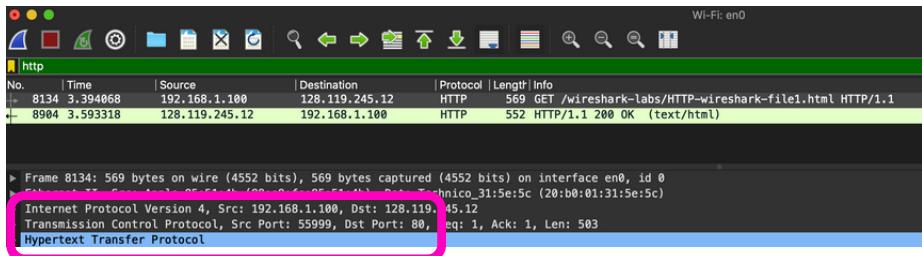
**3). What is the Internet address of the gaia.cs.umass.edu (also known as www-net.cs.umass.edu)? What is the Internet address of your computer?)**

**4). Print the two HTTP messages (GET and OK) referred to in question 2 above. To do so, select Print from the Wireshark File command menu, and select the "Selected Packet Only" and "Print as displayed" radial buttons, and then click OK.**

# Question 1

**List 3 different protocols that appear in the protocol column in the unfiltered packet-listing window in step 7 above.**

# Answer 1



Here we can clearly see 3 different protocols

- Internet Protocol Version 4. "IPV4"
- Transmission Control Protocol "TCP"
- Hypertext Transfer Protocol "HTTP"

## Question 2

How long did it take from when the HTTP GET message was sent until the HTTP OK reply was received? (By default, the value of the Time column in the packet-listing window is the amount of time, in seconds, since Wireshark tracing began. To display the Time field in time-of-day format, select the Wireshark View pull down menu, then select Time Display Format, then select Time-of-day.)

## Answer 2

Time
3.394068
3.593318

Time To execute 3.394068

## Question 3

3). What is the Internet address of the gaia.cs.umass.edu (also known as www-net.cs.umass.edu)?

What is the Internet address of your computer?

## Answer 3

Source	Destination
192.168.1.100	128.119.245.12
128.119.245.12	192.168.1.100

Computer Internet address :

- Source Address      "**192.168.1.100**"

gaia.cs.umass.edu Internet address :

- Destination Address    "**128.119.245.12**"

# Question 4

Print the two HTTP messages (GET and OK) referred to in question 2 above.

To

do so, select Print from the Wireshark File command menu, and select the "Selected Packet Only" and "Print as displayed" radial buttons, and then

click

OK.

# Answer 4

```
/var/folders/2h/_cqkmpms0k91t9rl27l22nph0000gn/T/wireshark_Wi-Fi2QJU0.pcapng 14279 total packets, 2 shown
```

No.	Time	Source	Destination	Protocol	Length	Info
8134	3.394068	192.168.1.100	128.119.245.12	HTTP	569	GET / wireshark-labs/HTTP-wireshark-file1.html HTTP/1.1 Frame 8134: 569 bytes on wire (4552 bits), 569 bytes captured (4552 bits) on interface en0, id 0 Ethernet II, Src: Apple_85:51:4b (88:e9:fe:85:51:4b), Dst: Technico_31:5e:5c (20:b0:01:31:5e: 5c) Internet Protocol Version 4, Src: 192.168.1.100, Dst: 128.119.245.12 Transmission Control Protocol, Src Port: 55999, Dst Port: 80, Seq: 1, Ack: 1, Len: 503 Hypertext Transfer Protocol

# WIRESHARK

## HTTP

1. Is your browser running HTTP version 1.0 or 1.1? What version of HTTP is the server running?
2. What languages (if any) does your browser indicate that it can accept to the server?
3. What is the IP address of your computer? Of the gaia.cs.umass.edu server?
4. What is the status code returned from the server to your browser?
5. When was the HTML file that you are retrieving last modified at the server?
6. How many bytes of content are being returned to your browser?
7. By inspecting the raw data in the packet content window, do you see any headers within the data that are not displayed in the packet-listing window? If so, name one.
8. Inspect the contents of the first HTTP GET request from your browser to the server. Do you see an “IF-MODIFIED-SINCE” line in the HTTP GET?
9. Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?
10. Now inspect the contents of the second HTTP GET request from your browser to the server. Do you see an “IF-MODIFIED-SINCE:” line in the HTTP GET? If so, what information follows the “IF-MODIFIED-SINCE:” header?
11. What is the HTTP status code and phrase returned from the server in response to this second HTTP GET? Did the server explicitly return the contents of the file? Explain.
12. How many HTTP GET request messages did your browser send? Which packet number in the trace contains the GET message for the Bill of Rights?
13. Which packet number in the trace contains the status code and phrase associated with the response to the HTTP GET request?
14. What is the status code and phrase in the response?
15. How many data-containing TCP segments were needed to carry the single HTTP response and the text of the Bill of Rights?
16. How many HTTP GET request messages did your browser send? To which Internet addresses were these GET requests sent?
17. Can you tell whether your browser downloaded the two images serially, or whether they were downloaded from the two web sites in parallel? Explain.
18. What is the server’s response (status code and phrase) in response to the initial HTTP GET message from your browser?
19. When your browser’s sends the HTTP GET message for the second time, what new field is included in the HTTP GET message?

# Question 1

Is your browser running HTTP version 1.0 or 1.1? What version of HTTP is the server running?

# Answer 1

```
552 HTTP/1.1 200 OK (text/html)
```

My browser running HTTP version 1.1

# Question 2

What languages (if any) does your browser indicate that it can accept to the server?

# Answer 2

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,  
Accept-Encoding: gzip, deflate\r\n  
Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7\r\n
```

French & English (US)

## Question 3

What is the IP address of your computer? Of the gaia.cs.umass.edu server?

## Answer 3

Source	Destination
192.168.1.100	128.119.245.12

Computer Internet address :

- Source Address        "**192.168.1.100**"

gaia.cs.umass.edu Internet address :

- Destination Address    "**128.119.245.12**"

## Question 4

What is the status code returned from the server to your browser?

## Answer 4

552 HTTP/1.1 200 OK (text/html)

**200 OK**

## Question 5

When was the HTML file that you are retrieving last modified at the server?

## Answer 5

Last-Modified: Sat, 14 Nov 2020 06:59:02 GMT\r\n

Saturday 14 November 2020 at 06:59:02

## Question 6

How many bytes of content are being returned to your browser?

## Answer 6

Accept-Ranges: bytes\r\n▼ Content-Length: 128\r\n[Content length: 128]  
Keep-Alive: timeout=5, max=1

Content-Length 128 Bytes

## Question 7

By inspecting the raw data in the packet content window, do you see any headers within the data that are not displayed in the packet-listing window? If so, name one. No. The raw data appears to match up exactly with what is shown in the packet-listing window.

## Answer 7

No, i don't see any header

## Question 8

Inspect the contents of the first HTTP GET request from your browser to the server. Do you see an "IF-MODIFIED-SINCE" line in the HTTP GET?

## Answer 8

No, i don't see any "IF-MODIFIED-SINCE"

## Question 9

Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?

## Answer 9

```
\n<html>\n\nCongratulations again! Now you've downloaded the file lab2-2.html. <br>\nThis file's last modification date will not change. <p>\nThus if you download this multiple times on your browser, a complete copy <br>\nwill only be sent once by the server due to the inclusion of the IN-MODIFIED-SINCE<br>\nfield in your browser's HTTP GET request to the server.\n\n</html>\n
```

Yes i can clearly see the contain html :  
"Congratulation. Again! ..."

## Question 10

Now inspect the contents of the second HTTP GET request from your browser to the server. Do you see an "IF-MODIFIED-SINCE:" line in the HTTP GET? If so, what information follows the "IF-MODIFIED-SINCE:" header?

## Answer 10

```
Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7\r\nIf-None-Match: "80-5b40bae1a9518"\r\nIf-Modified-Since: Sat, 14 Nov 2020 06:59:02 GMT\r\n\r\n
```

Sat, 14 Nov 2020

06:59:02

## Question 11

What is the HTTP status code and phrase returned from the server in response to this second HTTP GET? Did the server explicitly return the contents of the file? Explain.

## Answer 11

**304 HTTP/1.1 304 Not Modified**

▼ Hypertext Transfer Protocol  
    ▼ **HTTP/1.1 304 Not Modified\r\n**

The second HTTP GET returned 304 Not Modified because the request don't have to be renew the cash well know this address so it can be signified that nothing was modified since my last request .

## Question 12

How many HTTP GET request messages did your browser send? Which packet number in the trace contains the GET message for the Bill or Rights?

## Answer 12

1645	192.168.1.100	128.119.245.12	HTTP	569	GET /wireshark-labs/HTTP-wireshark-file3.htm
1652	128.119.245.12	192.168.1.100	HTTP	883	HTTP/1.1 200 OK (text/html)

My browser send 1 HTTP GET request message

## Question 13

Which packet number in the trace contains the status code and phrase associated with the response to the HTTP GET request?

## Answer 13

43	6	336645	192.168.1.100	128.119.245.12	HTTP	569	GET /wireshark-labs/HTT
		883	533652	128.119.245.12	HTTP	883	HTTP/1.1 200 OK (text/

No. 43

## Question 14

What is the status code and phrase in the response?

## Answer 14

GET /wireshark-labs/HTTP-wireshark-file3.html HTTP/1.1

GET /wireshark-labs/HTPP-wireshark-file3.html HTTP/1.1

## Question 15

How many data-containing TCP segments were needed to carry the single HTTP response and the text of the Bill of Rights?

## Answer 15

GET /wireshark-labs/HTTP-wireshark-file3.html HTTP/1.1  
HTTP/1.1 200 OK (text/html)

2 TCP segments     (1GET )(1 OK ).

## Question 16

How many HTTP GET request messages did your browser send? To which Internet addresses were these GET requests sent?

## Answer 16

No.	Time	Source	Destination	Protocol	Length	Info
52	2.834963	192.168.1.100	128.119.245.12	HTTP	569	GET /wireshark-labs/HTTP-wireshark-file4.html HTTP/1.1
75	3.038690	128.119.245.12	192.168.1.100	HTTP	1139	HTTP/1.1 200 OK (text/html)
77	3.057843	192.168.1.100	128.119.245.12	HTTP	581	GET /pearson.png HTTP/1.1
106	3.269795	128.119.245.12	192.168.1.100	HTTP	981	HTTP/1.1 200 OK (PNG)
133	3.478382	192.168.1.100	128.119.245.12	HTTP	475	GET /-kurose/cover_5th_ed.jpg HTTP/1.1
278	4.309361	128.119.245.12	192.168.1.100	HTTP	284	HTTP/1.1 200 OK (JPEG/JFIF image)

**1 destinations :**

- 128.119.245.12

## Question 17

Can you tell whether your browser downloaded the two images serially, or whether they were downloaded from the two web sites in parallel?  
Explain.

## Answer 17

### Picture 1

[3 Reassembled TCP Segments (3611 bytes): #103(1348), #104(1348), #106(915)]

### Picture 2

► [77 Reassembled TCP Segments (101318 bytes): #137(1348), #138(1348)]

both images have been downloaded in series

## Question 18

What is the server's response (status code and phrase) in response to the initial HTTP GET message from your browser?

## Answer 18

68 3.848689	192.168.1.100	128.119.245.12	HTTP	601	GET /wireshark-labs/protected_pages/HTTP-wireshark- HTTP/1.1
74 4.441301	128.119.245.12	192.168.1.100	HTTP	783	HTTP/1.1 401 Unauthorized (text/html)
319 32.771244	192.168.1.100	128.119.245.12	HTTP	660	GET /wireshark-labs/protected_pages/HTTP-wireshark- HTTP/1.1
324 33.791718	128.119.245.12	192.168.1.100	HTTP	586	HTTP/1.1 404 Not Found (text/html)

The server's response in the initial HTTP GET is the N°68  
GET/wireshark-labs/protected\_pages/HTTP-wireshark-  
HTTP/1.1

## Question 19

When your browser's sends the HTTP GET message for the second time, what new field is included in the HTTP GET message?

## Answer 19

68 3.848689	192.168.1.100	128.119.245.12	HTTP	601	GET /wireshark-labs/protected_pages/HTTP-wireshark- HTTP/1.1
74 4.441301	128.119.245.12	192.168.1.100	HTTP	783	HTTP/1.1 401 Unauthorized (text/html)
319 32.771244	192.168.1.100	128.119.245.12	HTTP	660	GET /wireshark-labs/protected_pages/HTTP-wireshark- HTTP/1.1
324 33.791718	128.119.245.12	192.168.1.100	HTTP	586	HTTP/1.1 404 Not Found (text/html)

Unauthorized



# תקשרות

## מטלה ג

Nathanael Benichou  
342769130



# WIRESHARK

## DNS

1. Run nslookup to obtain the IP address of a Web server in Asia. What is the IP address of that server?

```
nathben97@nathben97-VirtualBox: ~
Server:      127.0.0.53
Address:     127.0.0.53#53

Non-authoritative answer:
tsinghua.edu.cn nameserver = ns2.cuhk.hk.
tsinghua.edu.cn nameserver = dns.tsinghua.edu.cn.
tsinghua.edu.cn nameserver = dns2.tsinghua.edu.cn.
tsinghua.edu.cn nameserver = dns2.edu.cn.

Authoritative answers can be found from:

nathben97@nathben97-VirtualBox: ~$ nslookup www.apple.co.jp
Server:      127.0.0.53
Address:     127.0.0.53#53

Non-authoritative answer:
www.apple.co.jp canonical name = apple.co.jp.
Name:  apple.co.jp
Address: 17.172.224.38
Name:  apple.co.jp
Address: 17.142.160.9
Name:  apple.co.jp
Address: 17.178.96.9
```

Adresses :

- 17.172.224.38
- 17.142.160.9
- 17.178.96.9

## 2. Run nslookup to determine the authoritative DNS servers for a university in Europe.

```
nathben97@nathben97-VirtualBox: ~
www.apple.co.jp canonical name = apple.co.jp.
Name: apple.co.jp
Address: 17.172.224.38
Name: apple.co.jp
Address: 17.142.160.9
Name: apple.co.jp
Address: 17.178.96.9

nathben97@nathben97-VirtualBox:~$ c
c : commande introuvable
nathben97@nathben97-VirtualBox:~$ nslookup -type=NS univ-amu.fr
nslookup: '-type=NS' is not a legal IDNA2008 name (string contains a disallowed character), use +noidn
nathben97@nathben97-VirtualBox:~$ nslookup -type=NS univ-amu.fr
Server: 127.0.0.53
Address: 127.0.0.53#53

Non-authoritative answer:
univ-amu.fr      nameserver = ns1.univmed.fr.
univ-amu.fr      nameserver = cnudns.cines.fr.

Authoritative answers can be found from:
```

### Servers :

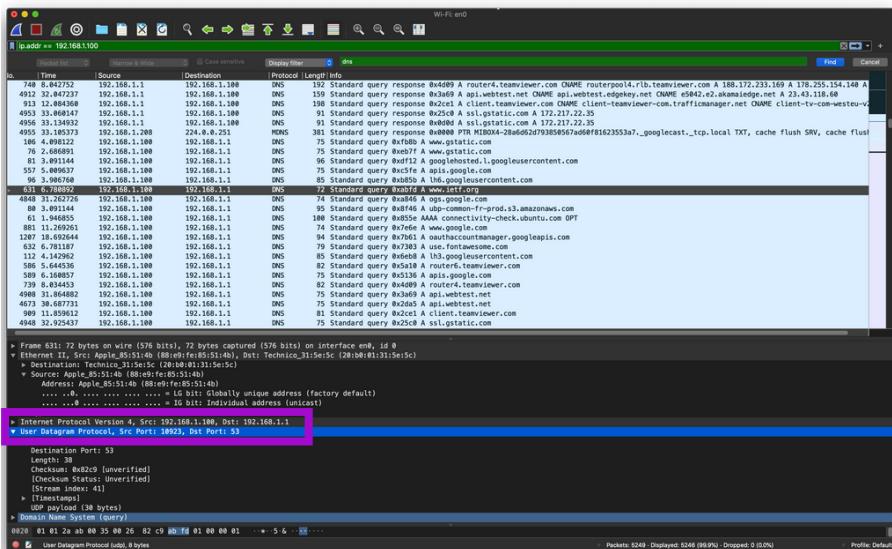
- ns1.univmed.fr
- cnudns.cines.fr

3.Run nslookup so that one of the DNS servers obtained in Question 2 is queried for the mail servers for Yahoo! mail. What is its IP address?

```
nathben97@nathben97-VirtualBox:~$ nslookup univ-am.fr mail.yahoo.com  
;; connection timed out; no servers could be reached
```

```
nathben97@nathben97-VirtualBox:~$ █
```

Connection timed out no servers could be reached



## 4. Locate the DNS query and response messages. Are they sent over UDP or TCP?

UDP

## 5. What is the destination port for the DNS query message? What is the source port of DNS response message?

- Source port :10923
- Destination port :53

6 .To what IP address is the DNS query message sent? Use ipconfig to determine the IP address of your local DNS server. Are these two IP addresses the same?

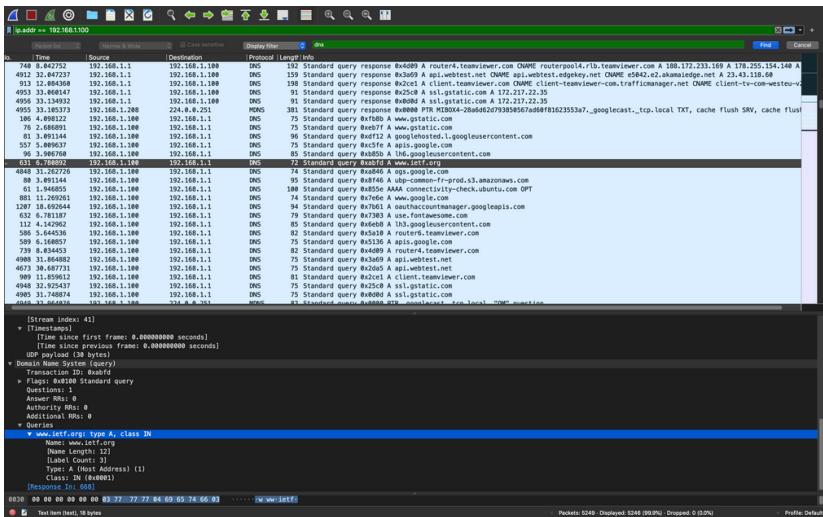
- Source :192.168.1.100
- Destination :192.168.1.1

```
status: inactive
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
    options=400<CHANNEL_IO>
    ether 88:e9:fe:85:51:4b
    inet6 fe80::1ca9:cd24:91e4:6f82%en0 prefixlen 64 secured scopeid 0x6
        inet 192.168.1.100 netmask 0xffffffff broadcast 192.168.1.255
            nd6 options=201<PERFORMNUD,DAD>
            media: autoselect
            status: active
```

The MacOs (ipconfig /all) is "ifconfig" the inet show us that the destination feet with our DNS answer 192.168.100

## 7.Examine the DNS query message.

### What “Type” of DNS query is it? Does the query message contain any “answers”?



Standard Query ;  
Type of DNS Query : A;  
no answer.

## 8. Examine the DNS response message.

### How many “answers” are provided?

### What do each of these answers contain?

No.	Time	Source	Destination	Protocol	Length	Info
658	6.91952	192.168.1.1	192.168.1.100	DNS	149	Standard query response 0xbabfd A www.ietf.org CNAME www.ietf.org.cdn.cloudflare.net A 184.16.45.99 A 184.16.44.99
4892	10.00000	192.168.1.1	192.168.1.100	DNS	143	Standard query response 0xbaf6 A upi-connectivity-prod-13.amazonaws.com CNAME upi-connectivity-prod-13.amazonaws.com A 52.219.47.10
83	1.199326	192.168.1.1	192.168.1.100	DNS	104	Standard query response 0xb55e AAAA connectivity-check.ubuntu.com IPv6
320	1.1993268	192.168.1.1	192.168.1.100	DNS	104	Standard query response 0xb55e AAAA connectivity-check.ubuntu.com IPv6
1222	18.24255	192.168.1.1	192.168.1.100	DNS	118	Standard query response 0xb761 A authaccountmanager.googleapis.com A 23.54.94.42
113	1.194899	192.168.1.1	192.168.1.100	DNS	130	Standard query response 0x8e8b A 13a.googleusercontent.com CNAME googleusercontent.com A 136.58.222.161
624	6.718318	192.168.1.1	192.168.1.100	DNS	206	Standard query response 0x8c38 A router6.teamviewer.com CNAME routerpool16.1b.teamviewer.com A 161.156.67.112 A 178.255.154.139 A
748	12.48475	192.168.1.1	192.168.1.100	DNS	137	Standard query response 0x8e8b A 13a.googleusercontent.com CNAME googleusercontent.com A 136.58.222.161
4933	33.109457	192.168.1.1	192.168.1.100	DNS	159	Standard query response 0x8d89 A router4.teamviewer.com CNAME routerpool4.1b.teamviewer.com A 180.172.233.160 A 178.255.154.148 A
4933	33.109457	192.168.1.1	192.168.1.100	DNS	130	Standard query response 0x8c38 A router6.teamviewer.com CNAME routerpool16.1b.teamviewer.com A 161.156.67.112 A 178.255.154.139 A
4953	33.109457	192.168.1.1	192.168.1.100	DNS	91	Standard query response 0x8c38 A s1.gptatic.com A 172.272.2.20
4953	33.109457	192.168.1.1	192.168.1.100	DNS	262	Standard query response 0x8c38 A s1.gptatic.com CNAME s1.gptatic.com A 172.272.2.20
4953	33.109457	192.168.1.1	192.168.1.100	DNS	381	Standard query response 0x8000 PTR M1024x28468d0793852353a7..._googlecast._tcp.local TXT, cache flush SW, cache flush SW, cache flush SW
180	4.003327	192.168.1.100	192.168.1.1	DNS	161	Standard-memo 0xb7bb A www.ietf.org

Answers :

3 answers contains :

( name, type, class, time to live, data ,length ,adress)

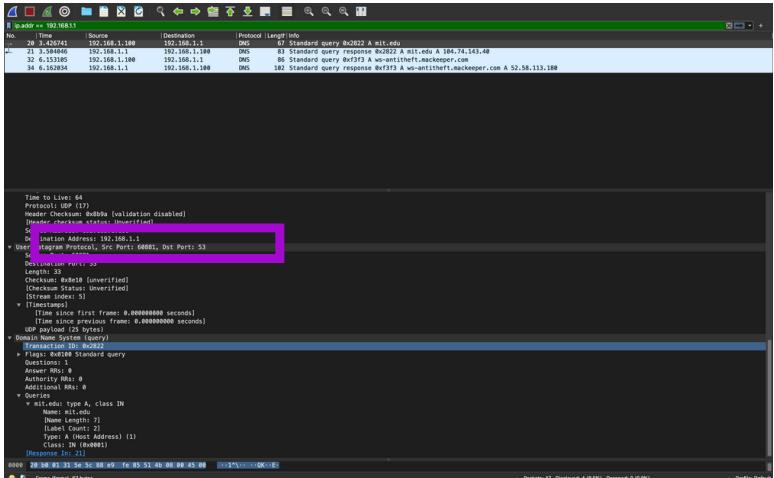
9. Consider the subsequent TCP SYN packet sent by your host. Does the destination IP address of the SYN packet correspond to any of the IP addresses provided in the DNS response message?

Answer : adresse of provided by the DNS server of ietf.org is 104.16.44.99 as seen in the last screenshot

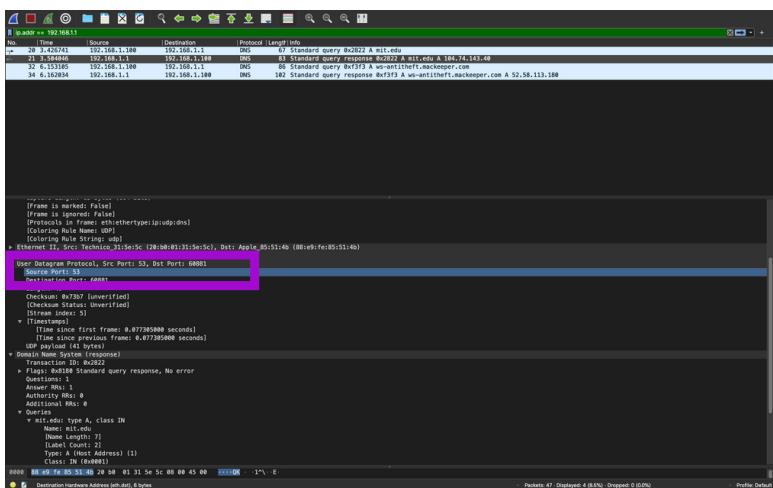
10. This web page contains images. Before retrieving each image, does your host issue new DNS queries?

All the images coming from ietf.org so no need to use a new DNS queries

# 11. What is the destination port for the DNS query message? What is the source port of DNS response message?

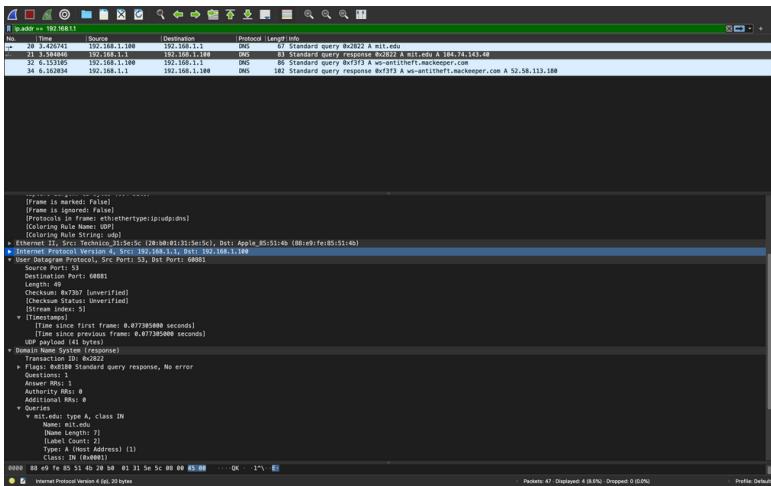


Request port 53



Answer port 53

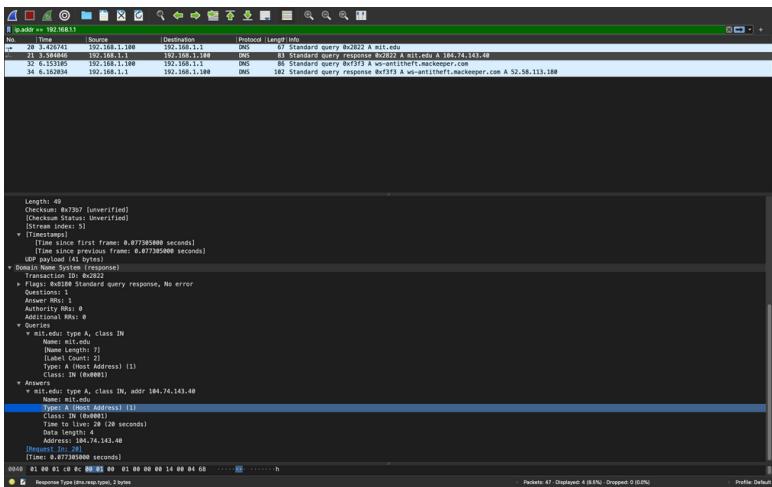
## 12. To what IP address is the DNS query message sent? Is this the IP address of your default local DNS server?



Destination is 192.168.1.100 its my Dns default ip as seen in the ifconfig screenshot

```
status: inactive
: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
options=400<CHANNEL_ID>
ether 88:e9:fe:85:51:4b
inet6 fe80::1ca9:c2d4:91e4:6f82%en0 prefixlen 64 secured scopeid 0x6
inet 192.168.1.100 netmask 0xffffffff broadcast 192.168.1.255
nd6 options=201<PERFORMNUD,DAD>
media: autoselect
status: active
```

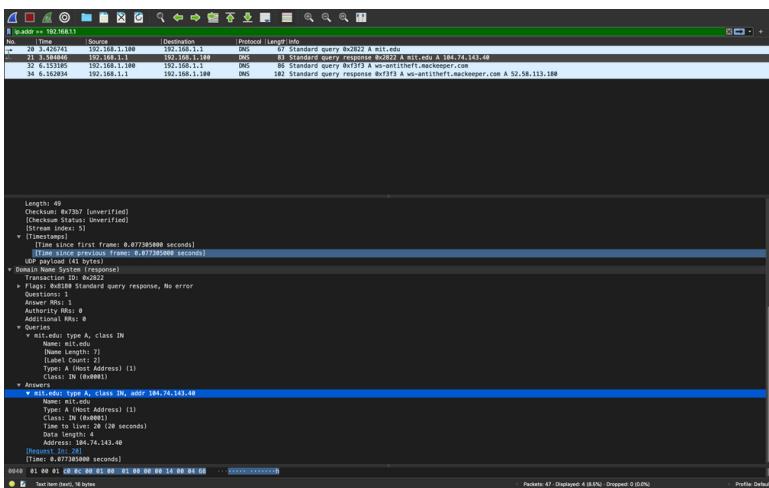
## 13. Examine the DNS query message. What “Type” of DNS query is it? Does the query message contain any “answers”?



No answer /DNS type :A

14. Examine the DNS response message. How many “answers” are provided? What do each of these answers contain?

## 15.ScreenShot

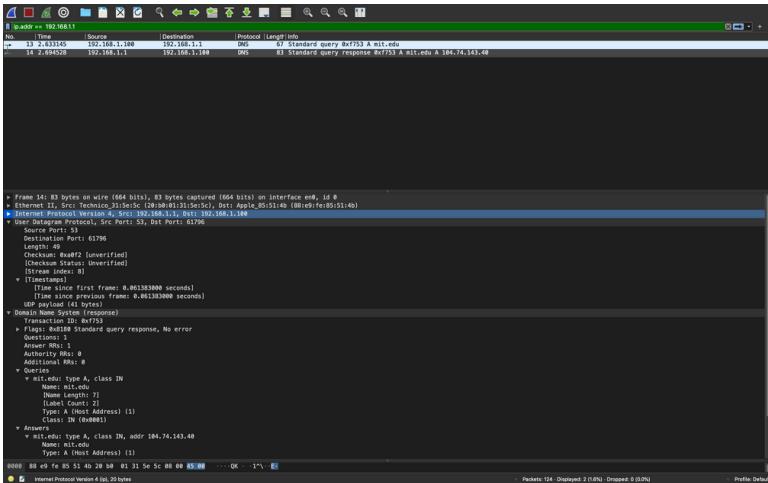


Answers :

1 answers contains :

( name, type, class, time to live, data ,length ,adress)

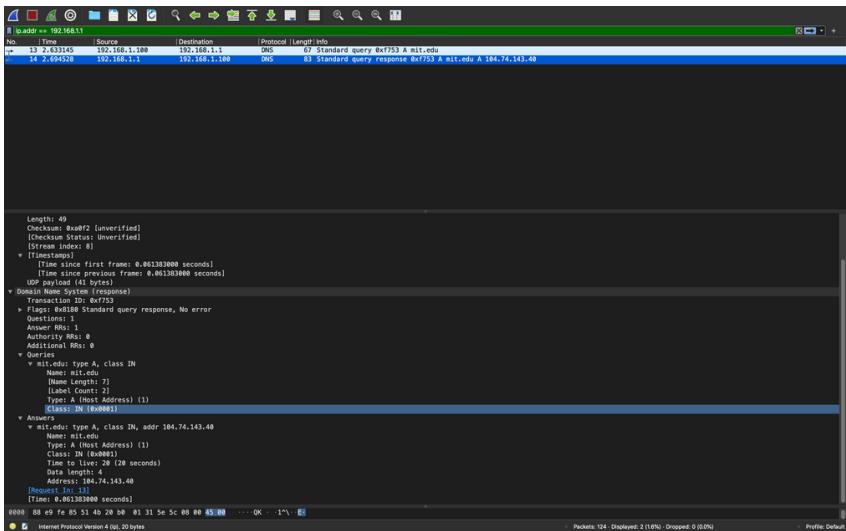
## 16. To what IP address is the DNS query message sent? Is this the IP address of your default local DNS server?



Destination is 192.168.1.100 its my Dns default ip as seen in the ifconfig screenshot

```
status: inactive
: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
  options=400<CHANNEL_IO>
  ether 88:e9:fe:85:51:4b
  inet6 fe80::1ca9:cd24:91e4:6f82%en0 prefixlen 64 secured scopeid 0x6
    inet 192.168.1.100 netmask 0xffffffff broadcast 192.168.1.255
      nd6 options=201<PERFORMNUD,DAD>
      media: autoselect
      status: active
```

## 17. Examine the DNS query message. What “Type” of DNS query is it? Does the query message contain any “answers”?



The screenshot shows a network traffic capture in Wireshark. The top part displays two DNS frames. Frame 13 is a query from 192.168.1.100 to 192.168.1.1 (mit.edu) asking for the A record for 'mit.edu'. Frame 14 is a response from 192.168.1.100 back to 192.168.1.100, which includes the IP address 104.74.143.40 for 'mit.edu'. The bottom part of the screenshot is a detailed view of the DNS response (Frame 14). It shows the packet structure with fields like Length: 49, Checksum: 0x00f2 [unverified], and Flags: (none) Status: Unverified. The DNS section details a standard query response with a Transaction ID of 0x7f53, a Time since previous frame of 0.061383000 seconds, and a Data length of 44 bytes. The 'Answers' section contains one entry for 'mit.edu' with type A and class IN, pointing to the IP address 104.74.143.40. There are also sections for Questions, Authority, and Additional records.

Answer: Type A query, and no answer

```
nathanaelbenichou — bash — 80x22
Q Rechercher
Address: 104.74.143.40
MacBook-Pro-de-Nathanael:~ nathanaelbenichou$ nslookup mit.edu
Server:      192.168.1.1
Address:      192.168.1.1#53

Non-authoritative answer:
Name:  mit.edu
Address: 104.74.143.40

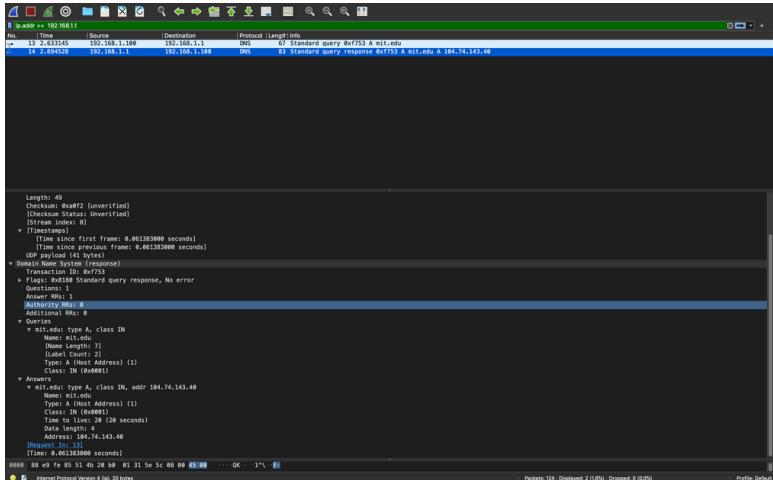
MacBook-Pro-de-Nathanael:~ nathanaelbenichou$ nslookup mit.edu
Server:      192.168.1.1
Address:      192.168.1.1#53

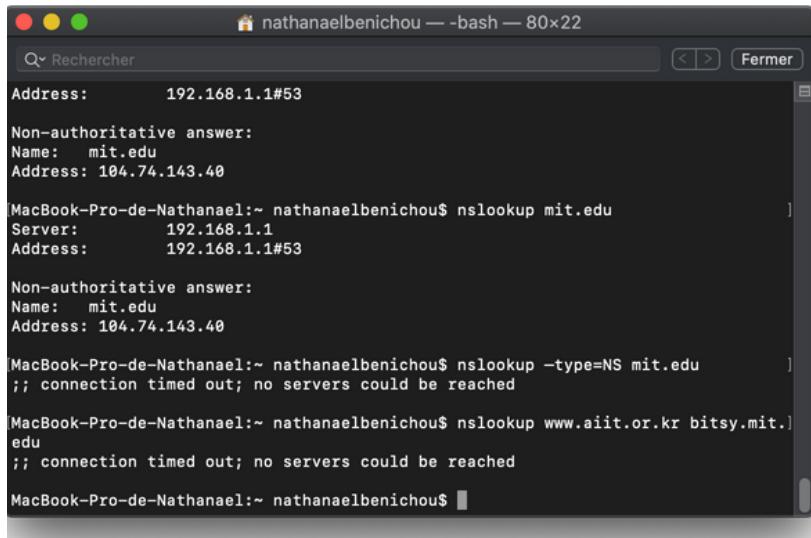
Non-authoritative answer:
Name:  mit.edu
Address: 104.74.143.40

MacBook-Pro-de-Nathanael:~ nathanaelbenichou$ nslookup -type=NS mit.edu
;; connection timed out; no servers could be reached
MacBook-Pro-de-Nathanael:~ nathanaelbenichou$
```

18. Examine the DNS response message.  
What MIT nameservers does the response message provide? Does this response message also provide the IP addresses of the MIT namesers?

19. Provide a screenshot.





nathanaelbenichou — bash — 80x22

Q Rechercher Fermer

```
Address: 192.168.1.1#53
Non-authoritative answer:
Name: mit.edu
Address: 104.74.143.40

MacBook-Pro-de-Nathanael:~ nathanaelbenichou$ nslookup mit.edu
Server: 192.168.1.1
Address: 192.168.1.1#53

Non-authoritative answer:
Name: mit.edu
Address: 104.74.143.40

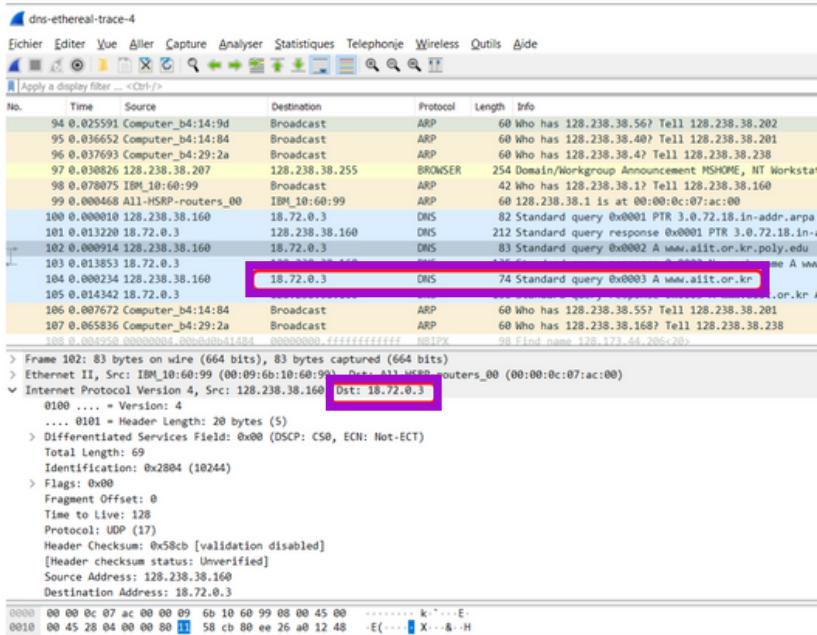
MacBook-Pro-de-Nathanael:~ nathanaelbenichou$ nslookup -type=NS mit.edu
;; connection timed out; no servers could be reached

MacBook-Pro-de-Nathanael:~ nathanaelbenichou$ nslookup www.aiit.or.kr bitsy.mit.edu
;; connection timed out; no servers could be reached

MacBook-Pro-de-Nathanael:~ nathanaelbenichou$
```

Doesn't work so i used the given pdf in  
this case

20. To what IP address is the DNS query message sent? Is this the IP address of your default local DNS server? If not, what does the IP address correspond to?



Answer: DNS send to 18.72.0.3  
the IP address of the wait response sender.

## 21. Examine the DNS query message. What "Type" of DNS query is it? Does the query message contain any "answers"?

No.	Time	Source	Destination	Protocol	Length	Info
94	0.025591	Computer_b4:14:9d	Broadcast	ARP	60	Who has 128.238.38.56? Tell 128.238.38.202
95	0.036652	Computer_b4:14:84	Broadcast	ARP	60	Who has 128.238.38.40? Tell 128.238.38.201
96	0.037693	Computer_b4:29:2a	Broadcast	ARP	60	Who has 128.238.38.47 Tell 128.238.38.238
97	0.038826	128.238.38.207	128.238.38.255	BROWSER	254	Domain/Workgroup Announcement MSHOME, NT Workstation, Domain
98	0.078875	IBM_10:60:99	Broadcast	ARP	42	Who has 128.238.38.17 Tell 128.238.38.160
99	0.000468	All-HSRP-routers_00	IBM_10:60:99	ARP	60	128.238.38.1 is at 00:00:0c:07:ac:00
100	0.000018	128.238.38.160	18.72.0.3	DNS	82	Standard query 0x0001 PTR 3.0.72.18.in-addr.arpa
101	0.013220	18.72.0.3	128.238.38.160	DNS	212	Standard query response 0x0001 PTR 3.0.72.18.in-addr.arpa PTR
102	0.000914	128.238.38.160	18.72.0.3	DNS	83	Standard query 0x0002 A www.ailt.on.kr.poly.edu
103	0.013853	18.72.0.3	128.238.38.160	DNS	135	Standard query response 0x0002 No such name A www.ailt.on.kr.
104	0.000234	128.238.38.160	18.72.0.3	DNS	74	Standard query 0x0003 A www.ailt.on.kr
105	0.014342	18.72.0.3	128.238.38.160	DNS	156	Standard query response 0x0003 A www.ailt.on.kr A 218.36.94.2
106	0.007672	Computer_b4:14:84	Broadcast	ARP	60	Who has 128.238.38.55? Tell 128.238.38.201
107	0.065836	Computer_b4:29:2a	Broadcast	ARP	60	Who has 128.238.38.160? Tell 128.238.38.238
108	0.000454	00000000.00000000-ffff-ffff-ffff-ffffffff	0010PX		98	Find name 128.173.44.20c<20>

```
> Frame 102: 83 bytes on wire (664 bits), 83 bytes captured (664 bits)
> Ethernet II, Src: IBM_10:60:99 (00:09:60:10:60:99), Dst: All-HSRP-routers_00 (00:00:0c:07:ac:00)
> Internet Protocol Version 4, Src: 128.238.38.160, Dst: 18.72.0.3
> User Datagram Protocol, Src Port: 3752, Dst Port: 53
└ Domain Name System (query)
   Transaction ID: 0x0002
   Flags: 0x0100 Standard query
   Questions: 1
     Answer RRs: 0
     Authority RRs: 0
     Additional RRs: 0
   Queries
     > www.aitl.or.kr.poly.edu: type A, class IN
      [Response In: 103]
```

Answer: Type of DNS : A  
Standard query  
no answer

22. Examine the DNS response message. How many "answers" are provided? What does each of these answers contain?

23. Provide a Screenshot:

The screenshot shows a Wireshark capture titled "dns-ethereal-trace-4". It displays a sequence of network frames. Frame 103 is a DNS query from host 18.72.0.3 to port 53, asking for the A record of "www.alit.or.kr.poly.edu". Frame 135 is a DNS response from port 53 back to 18.72.0.3, indicating that no such name exists. The interface is set to "All" and the display filter is "`Apply a display filter ... <Ctrl-f>`".

No.	Time	Source	Destination	Protocol	Length	Info
94	0.025591	Computer_b4:14:9d	Broadcast	ARP	60	Who has 128.238.38.56? Tell 128.238.38.202
95	0.036652	Computer_b4:14:84	Broadcast	ARP	60	Who has 128.238.38.40? Tell 128.238.38.201
96	0.037693	Computer_b4:29:2a	Broadcast	ARP	60	Who has 128.238.38.47? Tell 128.238.38.238
97	0.038262	128.238.38.207	128.238.38.255	BROWSER	254	Domain/Workgroup Announcement MSHOME, NT Workstation, Domain Env
98	0.078075	IBM_10:60:99	Broadcast	ARP	42	Who has 128.238.38.17? Tell 128.238.38.160
99	0.080468	All-HSRP-routers_00	IBM_10:60:99	ARP	60	128.238.38.1 is at 00:00:0c:07:ac:00
100	0.080919	128.238.38.160	18.72.0.3	DNS	82	Standard query 0x0001 PTR 3.0.72.18.in-addr.arpa
101	0.013220	18.72.0.3	128.238.38.160	DNS	212	Standard query Response 0x0001 PTR 3.0.72.18.in-addr.arpa PTR B1
102	0.080914	128.238.38.160	18.72.0.3	DNS	83	Standard query 0x0002 A www.alit.or.kr.poly.edu
103	0.013220	18.72.0.3	128.238.38.160	DNS	135	Standard query response 0x0002 No such name A www.alit.or.kr.poly.edu
104	0.080234	128.238.38.160	18.72.0.3	DNS	74	Standard query 0x0003 A www.alit.or.kr
105	0.014342	18.72.0.3	128.238.38.160	DNS	156	Standard query response 0x0003 A www.alit.or.kr A 218.36.94.200
106	0.007672	Computer_b4:14:84	Broadcast	ARP	60	Who has 128.238.38.55? Tell 128.238.38.201
107	0.065836	Computer_b4:29:2a	Broadcast	ARP	60	Who has 128.238.38.168? Tell 128.238.38.238

> Frame 103: 135 bytes on wire (1080 bits), 135 bytes captured (1080 bits)  
> Ethernet II, Src: Cisco\_83:e4:54 (00:0b:8e:83:e4:54), Dst: IBM\_10:60:99 (00:09:6b:10:60:99)  
> Internet Protocol Version 4, Src: 18.72.0.3, Dst: 128.238.38.160  
> User Datagram Protocol, Src Port: 53, Dst Port: 3752  
▼ Domain Name System (response)  
    Transaction ID: 0x0002  
    > Flags: 0x0583 Standard query response, No such name  
        Questions: 1  
        Answers RRs: 0  
        Authority RRs: 1  
        Additional RRs: 0  
    ▼ Queries  
        > www.alit.or.kr.poly.edu: type A, class IN  
    > Authoritative nameservers  
        [Request In: 102]  
        [Time: 0.013853000 seconds]  
0000 00 09 6b 10 60 99 00 b0 8e 83 e4 54 08 00 45 00 ··k.·· ··T-E···  
0010 00 79 b5 42 40 08 f1 11 1a 58 12 48 00 03 88 ee ·v-BB·· ·X-H···

Answer: Type of DNS : A  
Standard query  
no answer



# תקשרות

Nathanael Benichou  
342769130

—  
JORDAN PEREZ  
336165733



מטלה {

# PART I

## Question 1

הציגו יתרון אחד לשימוש ב-HoDo והסבירו אותו  
(כਮובן, מעבר לעובדה שהוא מאובטח ומצוון)

## Answer 1

מעבר להצפת הנתונים, ל- HoS יש יתרון  
בהגדלת הביצועים, גם במקרה של אובדן חבילות,  
הוא מהיר יותר בשילוחת מידע.

## Question 2

הציגו והסבירו על שני חסרונות לשימוש בשיטת  
HoD לועמת DNS הרגיל

## Answer 2

(א)

HoD מונע מעקב אחר משתמשים.  
בחינת שאילות DNS יכולה להיות חשפנית מאוד וניתן  
להשתמש בה לטובה (גילוי פעילות של תוכנות זדוניות) או  
לروع (לאתור מי מבקר באיזה אתר). שוב, זו המטרה  
המשמעות של HoD להקשות על הניטור.

לפיכך, מפעיל רשות אינטראקטיבית יכולם לסנן את התוכן של  
אתרים לא רצויים מכיוון שאין שמי משתמש ב-HoD עוקף את  
רשת האבטחה הנו"ל, ומסכן את המשתמש.

(ב)

השימוש ב-HoD הופך את הרשת לאיטה ופחות  
יעילה מכיוון שה מידע עבר ברשות מרוחקת

## Question 3

בחרו אחד מהחסרונות משלה (2), הציעו דרך למתן\Lעקו\לפתרון חיסרונו זה והסבירו אותה.

## Answer 3

השימוש ב- GEODNS יכול להפוך את השימוש ב- HOD לאיי פחות, GeoDns משפר את חיפוש שמות הדומיינים על ידי רגולציה כתובת בהתבסס על המיקום הגיאוגרפי של הלוח.

## Answer 4

א) יישום ה- HoD בرمת היישום

יתרונות: ניתן לישם את פרוטוקול ה- DNS בישומים הנתמכים.  
חסרון - אם המשתמש מתעלם מבקשתו, היישום לא יוכל להציג את המשתמש שהתעלם ממנו.

ב) יישום HoD בرمת שרת ה- \* Proxy - ברשות -

יתרון (אנונימיות): ה- proxy הוא נתיב המאפשר להפוך לאNONIMO. שימוש זה נותן אפשרות למשתמש כי IP שלהם, ולכן זהותם, נותר לא ידוע.

חסרון - למשתמשים שכנים המזוהים עם אותה רשות יש גישה למידע.

ג) יישום HoD של שרת ה- proxy המקומי -

יתרון  
גם אם המשתמשים מחוברים לאותה רשות הבקשות יוצפנו ולכן ההתקפה תהיה קשה יותר.

חסרון איקון  
התקנה כללית, באופן מקומי, עלולה ליצור עומס.

ד) התקן תוסף המישם את HoDo:

יתרונות:  
בניגוד ל-Proxy, בהיותו מודול תוסף, אין צורך להתקין אותו בכל המכונות בנפרד.

חסרון:  
לא כל מערכות הפעלה תומכות בתוסף

## מסקנות

נראה שהכי טוב הוא לחסוך זמן ולהימנע מכל סוג של עומס, הפתרון המועיל ביותר נראה לי הפלאגין.

## Question 5

נניח שאנו ברשות שקיים בה איבוד פקודות (packet loss) באחוז לא ידוע לנו רוצים לטעון דף שציריך 25 שאלות כדי לבדוק את כל המשאבים שבו. הציגו יתרון בhor שיש ל-Ho53 לעומת Ho53. (רמז: מנגן הקיים ב-TCP)

## Answer 5

בהתליר Ho53 משתמש בפרוטוקול UDP.

אין לו את אותם המאפיינים.

כאשר משתמשים ב-Ho53 במהלך אובדן מנות, מנגן שולח *acks*, מים לחכילה לפני זה שאבד, ולכן ניתן יהיה לשולח אותו שוב ללא בעיה.

# PART II

```
Jordan@Jordan-VirtualBox: ~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp
```



```
jordan@Jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$ ./measure
Socket successfully created..
Socket successfully binded..
Server listening..
server acccept the client...
```

Currently using cubic:

```
Received file 1 from client in 0.0000050 seconds
Received file 2 from client in 0.0000020 seconds
Received file 3 from client in 0.0000010 seconds
Received file 4 from client in 0.0000020 seconds
Received file 5 from client in 0.0000010 seconds
```

Average time: 0.0000022 seconds

Now using reno:

```
Received file 1 from client in 0.0000020 seconds
Received file 2 from client in 0.0000010 seconds
Received file 3 from client in 0.0000010 seconds
Received file 4 from client in 0.0000010 seconds
Received file 5 from client in 0.0000030 seconds
```

Average time: 0.0000016 seconds

```
Jordan@Jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$
```

```
Jordan@Jordan-VirtualBox: ~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp
```



```
jordan@Jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$ ./sender
Socket successfully created..
Current: cubic
connected to the server..
New: reno
jordan@Jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$
```

# No Packet Loss

```
jordan@jordan-VirtualBox:~$ sudo tc qdisc add dev lo root netem loss 10%
jordan@jordan-VirtualBox:~$ █
```

```
Jordan@jordan-VirtualBox: ~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp
```

```
jordan@jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$ ./measure
Socket successfully created..
Socket successfully binded..
Server listening..
server acccept the client...
```

```
Currently using cubic:
Received file 1 from client in 0.0000040 seconds
Received file 2 from client in 0.00000170 seconds
Received file 3 from client in 0.0000010 seconds
Received file 4 from client in 0.0000020 seconds
Received file 5 from client in 0.0000020 seconds
```

Average time: 0.0000052 seconds

```
Now using reno:
Received file 1 from client in 0.0000020 seconds
Received file 2 from client in 0.0000020 seconds
Received file 3 from client in 0.0000020 seconds
Received file 4 from client in 0.0000010 seconds
Received file 5 from client in 0.0000020 seconds
```

Average time: 0.0000018 seconds

```
jordan@jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$ █
```

```
Jordan@Jordan-VirtualBox: ~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp
```

```
jordan@jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last verston/Tcp$ ./sender
Socket successfully created..
Current: cubic
connected to the server..
New: reno
jordan@jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$ █
```

# 10% Packet Loss

```
jordan@jordan-VirtualBox:~$ sudo tc qdisc add dev lo root netem loss 15%
jordan@jordan-VirtualBox:~$ █
```

```
jordan@jordan-VirtualBox: ~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$ █
```

```
jordan@jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$ ./measure
```

```
Socket successfully created..
```

```
Socket successfully binded..
```

```
Server listening..
```

```
server acccept the client...
```

```
Currently using cubic:
```

```
Received file 1 from client in 0.0000190 seconds
```

```
Received file 2 from client in 0.0000020 seconds
```

```
Received file 3 from client in 0.0000020 seconds
```

```
Received file 4 from client in 0.0000020 seconds
```

```
Received file 5 from client in 0.0000020 seconds
```

Average time: 0.0000054 seconds

```
Now using reno:
```

```
Received file 1 from client in 0.0000220 seconds
```

```
Received file 2 from client in 0.0000020 seconds
```

```
Received file 3 from client in 0.0000020 seconds
```

```
Received file 4 from client in 0.0000010 seconds
```

```
Received file 5 from client in 0.0000030 seconds
```

Average time: 0.0000060 seconds

```
jordan@jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$ █
```

```
jordan@jordan-VirtualBox: ~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$ ./sender
```

```
Socket successfully created..
```

```
Current: cubic
```

```
connected to the server..
```

```
New: reno
```

```
jordan@jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$ █
```

# 15% Packet Loss

jordan@jordan-VirtualBox: ~

```
jordan@jordan-VirtualBox:-$ sudo tc qdisc add dev lo root netem loss 20%
jordan@jordan-VirtualBox:-$ 
```

Jordan@jordan-VirtualBox: ~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp

```
jordan@jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$ ./measure
Socket successfully created..
Socket successfully binded..
Server listening..
server acccept the client...
```

Currently using cubic:  
Received file 1 from client in 0.0000060 seconds  
Received file 2 from client in 0.0000560 seconds  
Received file 3 from client in 0.0000030 seconds  
Received file 4 from client in 0.0000020 seconds  
Received file 5 from client in 0.0000010 seconds

Average time: 0.0000136 seconds

Now using reno:  
Received file 1 from client in 0.0000010 seconds  
Received file 2 from client in 0.0000010 seconds  
Received file 3 from client in 0.0000020 seconds  
Received file 4 from client in 0.0000020 seconds  
Received file 5 from client in 0.0000040 seconds

Average time: 0.0000020 seconds

```
jordan@jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$ 
```

Jordan@jordan-VirtualBox: ~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp

```
jordan@jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$ ./sender
Socket successfully created..
Current: cubic
connected to the server..
New: reno
jordan@jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$ 
```

# 20% Packet Loss

jordan@Jordan-VirtualBox: ~

```
jordan@jordan-VirtualBox:~$ sudo tc qdisc add dev lo root netem loss 25%
jordan@jordan-VirtualBox:~$ █
```

Jordan@Jordan-VirtualBox: ~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp

```
jordan@jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$ ./measure
Socket successfully created..
Socket successfully binded..
Server listening..
server acccept the client...
```

```
Currently using cubic:
Received file 1 from client in 0.0000070 seconds
Received file 2 from client in 0.0000020 seconds
Received file 3 from client in 0.0000010 seconds
Received file 4 from client in 0.0000010 seconds
Received file 5 from client in 0.0000020 seconds
```

Average time: 0.0000026 seconds

```
Now using reno:
Received file 1 from client in 0.0000010 seconds
Received file 2 from client in 0.0000030 seconds
Received file 3 from client in 0.0000570 seconds
Received file 4 from client in 0.0000050 seconds
Received file 5 from client in 0.0000250 seconds
```

Average time: 0.0000182 seconds

```
jordan@jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$ █
```

Jordan@Jordan-VirtualBox: ~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp

```
jordan@jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$ ./sender
Socket successfully created..
Current: cubic
connected to the server..
New: reno
jordan@jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$
```

# 25% Packet Loss

Jordan@Jordan-VirtualBox: ~

```
jordan@jordan-VirtualBox:~$ sudo tc qdisc add dev lo root netem loss 30%
jordan@jordan-VirtualBox:~$ 
```

Jordan@Jordan-VirtualBox: ~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp

```
jordan@jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$ ./measure
```

```
Socket successfully created..
```

```
Socket successfully binded..
```

```
Server listening..
```

```
server acccept the client...
```

```
Currently using cubic:
```

```
Received file 1 from client in 0.0000490 seconds
```

```
Received file 2 from client in 0.0000030 seconds
```

```
Received file 3 from client in 0.0000010 seconds
```

```
Received file 4 from client in 0.0000010 seconds
```

```
Received file 5 from client in 0.0000020 seconds
```

Average time: 0.0000112 seconds

```
Now using reno:
```

```
Received file 1 from client in 0.0000020 seconds
```

```
Received file 2 from client in 0.0000010 seconds
```

```
Received file 3 from client in 0.0000010 seconds
```

```
Received file 4 from client in 0.0000010 seconds
```

```
Received file 5 from client in 0.0000040 seconds
```

Average time: 0.0000018 seconds

Jordan@Jordan-VirtualBox: ~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp\$

Jordan@Jordan-VirtualBox: ~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp

```
jordan@jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp$ ./sender
```

```
Socket successfully created..
```

```
Current: cubic
```

```
connected to the server..
```

```
New: rno
```

Jordan@Jordan-VirtualBox: ~/Desktop/Tikchoret/Tikchoret 3/last version/Tcp\$

# 30% Packet Loss



# תקשרות

## מטרה ק'

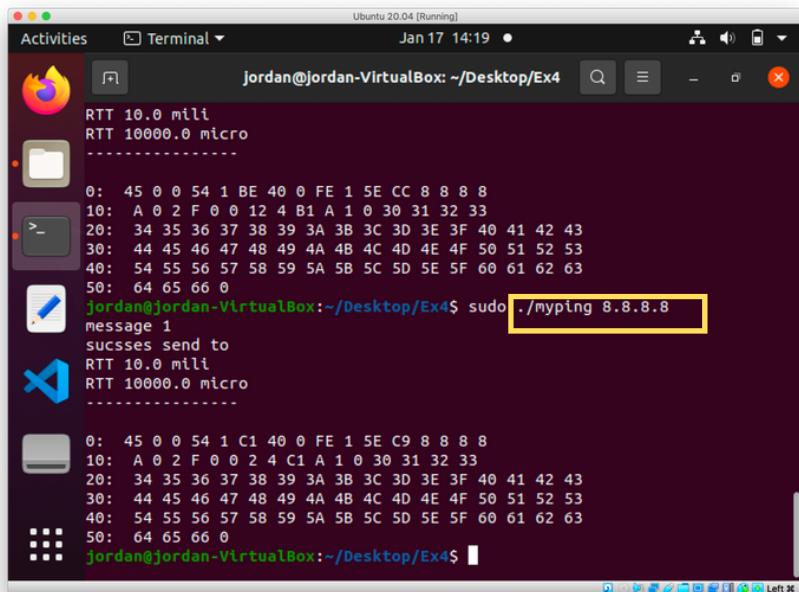
Nathanael Benichou-Jordan Perez  
342769130 - 336165733



# PARTIE 1

אך חלון - *myping*

Send Ping with myping.c

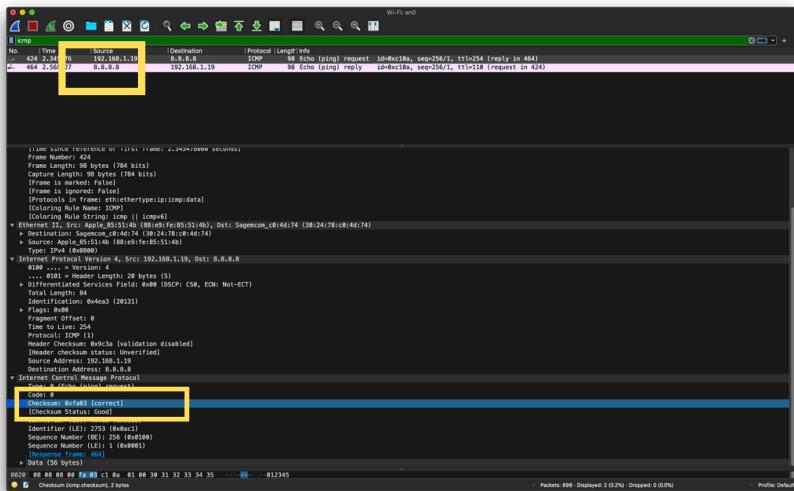


The screenshot shows a terminal window on an Ubuntu 20.04 desktop environment. The terminal title is "Terminal" and the prompt is "jordan@jordan-VirtualBox: ~/Desktop/Ex4\$". The user has run the command "sudo ./myping 8.8.8.8" which is highlighted with a yellow rectangle. The terminal displays the raw hex dump of the ping message sent to 8.8.8.8, followed by the response from Google's public DNS server.

```
RTT 10.0 mili
RTT 10000.0 micro
-----
0: 45 0 0 54 1 BE 40 0 FE 1 5E CC 8 8 8 8
10: A 0 2 F 0 0 12 4 B1 A 1 0 30 31 32 33
20: 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43
30: 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53
40: 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63
50: 64 65 66 0
jordan@jordan-VirtualBox:~/Desktop/Ex4$ sudo ./myping 8.8.8.8
message 1
succses send to
RTT 10.0 mili
RTT 10000.0 micro
-----
0: 45 0 0 54 1 C1 40 0 FE 1 5E C9 8 8 8 8
10: A 0 2 F 0 0 2 4 C1 A 1 0 30 31 32 33
20: 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43
30: 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53
40: 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63
50: 64 65 66 0
jordan@jordan-VirtualBox:~/Desktop/Ex4$
```

*Send ping to 8.8.8.8*

# Capture pacquets with WireShark



*Paquets ICMP well sniffed , CheckSum  
Correct ,Destination is 8.8.8.8*

# Send Ping(Another try)

A screenshot of a Linux desktop environment. In the center is a terminal window titled "Terminal" with the command "Jordan@jordan-VirtualBox: ~/Desktop/Ex4\$ sudo ./myping 8.8.8.8". The terminal shows several lines of output from the ping command, including message counts, RTT values, and hex dump of the transmitted packets. Below the terminal is a file browser window showing a folder named "Ex4".

```
20: 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43
30: 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53
40: 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63
50: 64 65 66 0
jordan@jordan-VirtualBox:~/Desktop/Ex4$ sudo ./myping 8.8.8.8
Message 1
succses send to
RTT 10.0 mili
RTT 10000.0 micro
-----
0: 45 0 0 54 1 C1 40 0 FE 1 5E C9 8 8 8 8
10: A 0 2 F 0 0 2 4 C1 A 1 0 30 31 32 33
20: 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43
30: 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53
40: 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63
50: 64 65 66 0
jordan@jordan-VirtualBox:~/Desktop/Ex4$ sudo ./myping 8.8.8.7
Message 1
succses send to
RTT 45.0 mili
RTT 45000.0 micro
^C
jordan@jordan-VirtualBox:~/Desktop/Ex4$
```

*Send ping to 8.8.8.7*

A screenshot of the Wireshark network traffic analyzer. It shows a single ICMP echo request (ping) packet. The packet details pane shows the source as 192.168.1.19 and the destination as 8.8.8.7. The bytes pane shows the raw hex and ASCII data of the ICMP packet.

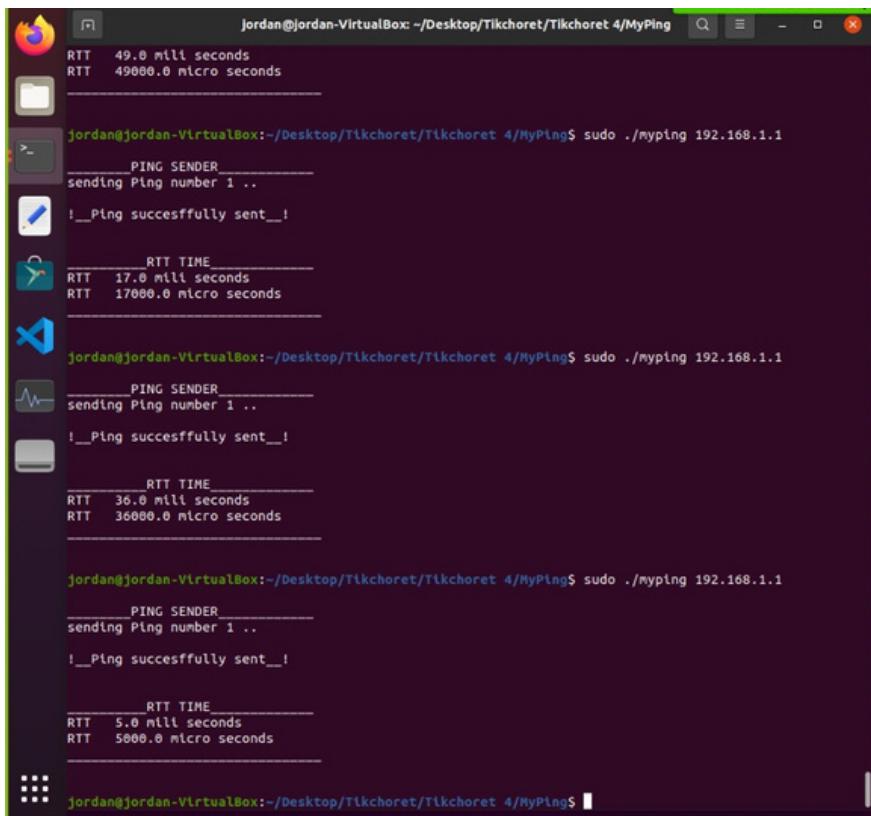
```
Capture: 1 packets on wire (96 bytes)
[Frame is marked: False]
[Frame is ignored: False]
[Protocol name: Ethernet]
[Physical Layer: IEEE 802.3 (Ethernet)]
[Coloring Rule String: "ethertype(ether) and (ether[12]>=89)"]
[Coloring Rule ID: 1]
[Coloring Rule String: "icmp || icmpv6"]
[Coloring Rule ID: 2]
[Selected: 1]
[Total length: 96 bytes]
[Next: 2]
[Previous: 0]
[Type: IPv4 (0x0800)]
Internet Protocol Version 4, Src: 192.168.1.19, Dst: 8.8.8.7
... IP = Header Length: 20 bytes (5)
... Identification: 0x0000 (0)
... TTL: 254 (0x00)
... TOS: 0x00 (0)
... Flags: 0x00 (0)
... 0... .... Reserved bit: Not set
... .0... .... More fragments: Not set
... .1... .... More fragments: Not set
Fragment offset: 0
Time to Live: 254
Protocol: ICMP (1)
Header checksum: 0x0000 (validation disabled)
[Header checksum status: Unverified]
Sequence Number (8B): 256 (0x0100)
Destination Address: 8.8.8.7
Internet Control Message Protocol
Code: 0
Checksum: 0x0000 (0)
[Checksum Status: Good]
Identification (16B): 0x0000 (0)
Identification (16B): 256 (0x0100)
Sequence Number (8B): 256 (0x0100)
Sequence Number (8B): 256 (0x0100)
[No response seen]
Data (56 bytes)
00 24 00 00 40 00 fe 01 0C d3 c9 a8 01 13 00 00
Destination Address (8B): 0x0000
```

*Well Recieved*

# PARTIE 2

## Sniffing – בְּמַלְחָקָה

*Send 7 Ping to the Same Adress*



```
Jordan@Jordan-VirtualBox: ~/Desktop/Tikchoret/Tikchoret 4/MyPing$ sudo ./myping 192.168.1.1
____ PING SENDER _____
sending Ping number 1 ..
!_Ping successfully sent_!

      RTT TIME
RTT  49.0 mill seconds
RTT 49000.0 micro seconds

____ PING SENDER _____
sending Ping number 1 ..
!_Ping successfully sent_!

      RTT TIME
RTT 17.0 mill seconds
RTT 17000.0 micro seconds

____ PING SENDER _____
sending Ping number 1 ..
!_Ping successfully sent_!

      RTT TIME
RTT 36.0 mill seconds
RTT 36000.0 micro seconds

____ PING SENDER _____
sending Ping number 1 ..
!_Ping successfully sent_!

      RTT TIME
RTT 5.0 mill seconds
RTT 5000.0 micro seconds

____ PING SENDER _____
sending Ping number 1 ..
!_Ping successfully sent_!

      RTT TIME
RTT 5.0 mill seconds
RTT 5000.0 micro seconds

____ PING SENDER _____
sending Ping number 1 ..
!_Ping successfully sent_!

      RTT TIME
RTT 5.0 mill seconds
RTT 5000.0 micro seconds

____ PING SENDER _____
sending Ping number 1 ..
!_Ping successfully sent_!

      RTT TIME
RTT 5.0 mill seconds
RTT 5000.0 micro seconds

jordan@Jordan-VirtualBox: ~/Desktop/Tikchoret/Tikchoret 4/MyPing$ sudo ./myping 192.168.1.1
```

# *Sniff The Ping Sent with Sniffer.c*

```
jordan@jordan-VirtualBox:~/Desktop/Tikchoret/Tikchoret 4/Or Sniffer
```

---

```
Available Devices are :
1. enp0s3 - (null)
2. enp0s8 - (null)
3. lo - (null)
4. any - Pseudo-device that captures on all interfaces
5. bluetooth-monitor - Bluetooth Linux Monitor
6. nflog - Linux netfilter log (NFLOG) interface
7. nfqueue - Linux netfilter queue (NFQUEUE) interface

Which device would you want to sniff ? 1
Opening device enp0s3 for sniffing ... Done
Number of Packet ICMP : 6
#Source IP      : 10.0.2.15
#Destination IP : 192.168.1.1
#Code : 0
#Type : 8

Number of Packet ICMP : 1
#Source IP      : 192.168.1.1
#Destination IP : 10.0.2.15
#Code : 0
#Type : 8

Number of Packet ICMP : 2
#Source IP      : 10.0.2.15
#Destination IP : 192.168.1.1
#Code : 0
#Type : 8

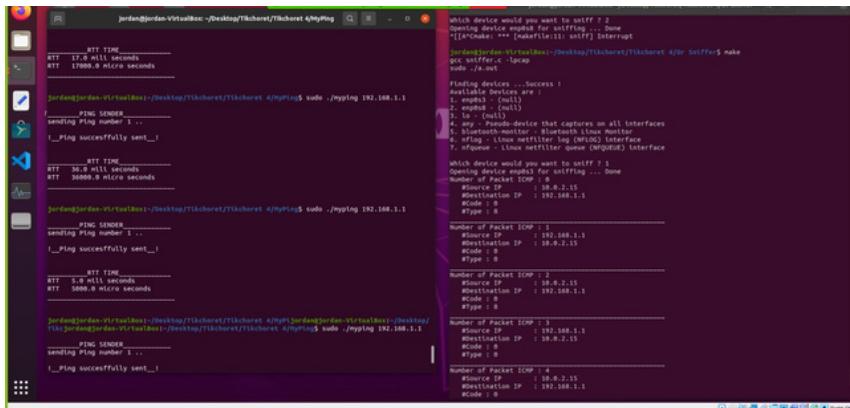
Number of Packet ICMP : 3
#Source IP      : 192.168.1.1
#Destination IP : 10.0.2.15
#Code : 0
#Type : 0

Number of Packet ICMP : 4
#Source IP      : 10.0.2.15
#Destination IP : 192.168.1.1
#Code : 0
#Type : 8

Number of Packet ICMP : 5
#Source IP      : 192.168.1.1
#Destination IP : 10.0.2.15
#Code : 0
#Type : 0
```



# Compare My Sniffer And WireShark



```
Jordan@Jordan-VirtualBox:~/Desktop/Tikshoret/Tikshoret 4$ ./MyPing
RTT TIME_
RTT 17.0 mill seconds
RTT 17000.0 micro seconds

Jordan@Jordan-VirtualBox:~/Desktop/Tikshoret/Tikshoret 4$ ./MyPing 192.168.1.1
PING SENDER
sending Ping number 1 ...
l..._Ping successfully sent_!

RTT TIME_
RTT 30.0 mill seconds
RTT 36000.0 micro seconds

Jordan@Jordan-VirtualBox:~/Desktop/Tikshoret/Tikshoret 4$ ./MyPing 192.168.1.1
PING SENDER
sending Ping number 1 ...
l..._Ping successfully sent_!

RTT TIME_
RTT 5.0 mill seconds
RTT 5800.0 micro seconds

Jordan@Jordan-VirtualBox:~/Desktop/Tikshoret/Tikshoret 4$ ./MySniff
sending Ping number 1 ...
l..._Ping successfully sent_!

Jordan@Jordan-VirtualBox:~/Desktop/Tikshoret/Tikshoret 4$ ./MySniff
which device would you want to sniff? 2
Opening device named for sniffing... done
[["A-Chair"] *** [A-sniff] sniff) Interrupt
Jordan@Jordan-VirtualBox:~/Desktop/Tikshoret/Tikshoret 4$ ./MySniff 192.168.1.1
sudo /bin/sh
Finding devices ... Success !
Available Device are :
1. enp0s3 (null)
2. enp0s8 (null)
3. enp0s9 (null)
4. any - Pseudo-device that captures on all interfaces
5. nflog - Linux netfilter log (NFLOG) Interface
6. nfqueue - Linux netfilter queue (NFQUEUE) Interface
7. nfqueue - Linux netfilter queue (NFQUEUE) Interface

which device would you want to sniff ? 2
Opening device named for sniffing... done
Number of Packet ICMP : 0
#Source IP : 192.168.1.15
#Destination IP : 192.168.1.5
#Code : 0
#Type : 8

Number of Packet ICMP : 1
#Source IP : 192.168.1.5
#Destination IP : 192.168.1.15
#Code : 0
#Type : 8

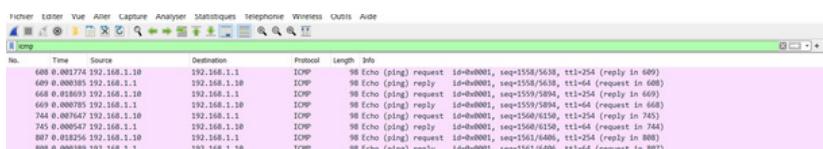
Number of Packet ICMP : 2
#Source IP : 192.168.1.15
#Destination IP : 192.168.1.5
#Code : 0
#Type : 8

Number of Packet ICMP : 3
#Source IP : 192.168.1.5
#Destination IP : 192.168.1.15
#Code : 0
#Type : 8

Number of Packet ICMP : 4
#Source IP : 192.168.1.5
#Destination IP : 192.168.1.15
#Code : 0
#Type : 8
```



*Send 4 Ping to the Same Adresse  
Sniffe With sniffer.c & WireShark*



No.	Time	Source	Destination	Protocol	Length	Info
608	0.000174	192.168.1.10	192.168.1.1	ICMP	98	Echo (ping) request 1d=0x0001, seq=1558/5639, ttl=254 (reply in 689)
609	0.000195	192.168.1.1	192.168.1.10	ICMP	98	Echo (ping) reply 1d=0x0001, seq=1558/5639, ttl=254 (request in 689)
610	0.000215	192.168.1.1	192.168.1.10	ICMP	98	Echo (ping) request 1d=0x0001, seq=1559/5894, ttl=254 (reply in 668)
605	0.000785	192.168.1.10	192.168.1.10	ICMP	98	Echo (ping) reply 1d=0x0001, seq=1559/5894, ttl=64 (request in 668)
744	0.007647	192.168.1.10	192.168.1.1	ICMP	98	Echo (ping) request 1d=0x0001, seq=1560/6156, ttl=254 (reply in 745)
745	0.008547	192.168.1.1	192.168.1.10	ICMP	98	Echo (ping) reply 1d=0x0001, seq=1560/6156, ttl=64 (request in 745)
807	0.000250	192.168.1.10	192.168.1.1	ICMP	98	Echo (ping) request 1d=0x0001, seq=1561/6400, ttl=254 (reply in 807)
808	0.000387	192.168.1.1	192.168.1.10	ICMP	98	Echo (ping) reply 1d=0x0001, seq=1561/6400, ttl=64 (request in 807)



# תקשורת

## FINAL PROJECT

Packet Sniffing and Spoofing Lab

Nathanael Benichou-Jordan Perez  
342769130 - 336165733



# PACKET SNIFFING AND SPOOFING LAB

*Packet sniffing and spoofing are the two important concepts in network security; they are two major threats in network communication. Being able to understand these two threats is essential for understanding security measures in networking.*

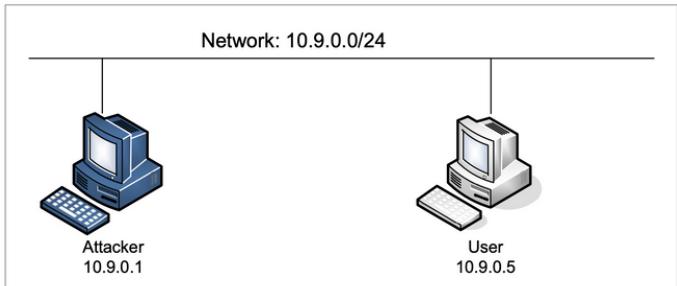
*There are many packet sniffing and spoofing tools, such as Wireshark, Tcpdump, Netwox, etc. Some of these tools are widely used by security experts, as well as by attackers. Being able to use these tools is important for students, but what is more important for students in a network security course is to understand how these tools work, i.e., how packet sniffing and spoofing are implemented in software.*

*The objective of this lab is to master the technologies underlying most of the sniffing and spoofing tools.*

*Students will play with some simple sniffer and spoofing programs, read their source code, modify them, **perform a man in the middle attack**, and eventually gain an in-depth understanding on the technical aspects of these programs. At the end of this lab, students should be able to write their own sniffing and spoofing programs.*

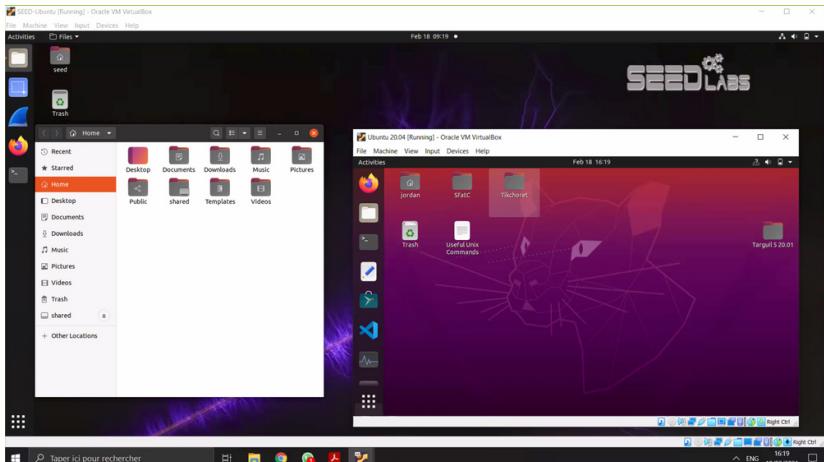


# ENVIRONMENT SETUP USING CONTAINER



## Lab Environment Setup: Three Virtual Machines emulating Linux Ubuntu 20.04 (SEED Version)

- 1.VM (Seed attacker) - IP 10.0.2.7
- 2.VM SEED Ubuntu (Victim 1) - IP 10.0.2.8
- 3.VM SEED Ubuntu (Victim2) - IP 10.0.2.9



*First of all we configured a shared folder  
for more convenience*

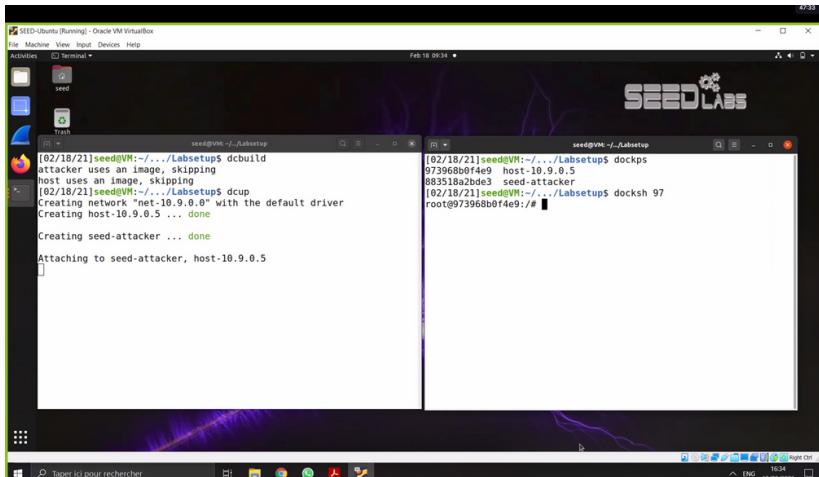
# FIRST CONTACT WITH DOCKERS

Instead of using multiple VMs we could also use Dockers to emulate containers like this:

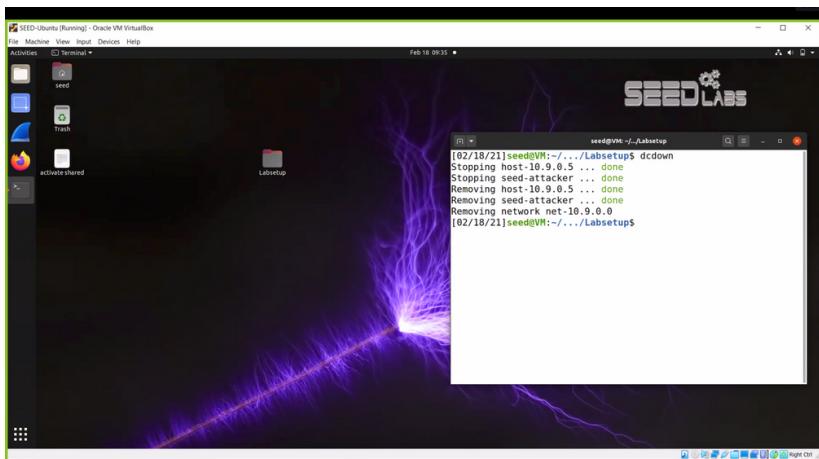
*Commands used :*

**\$ dockps**

**\$ docksh <id>**



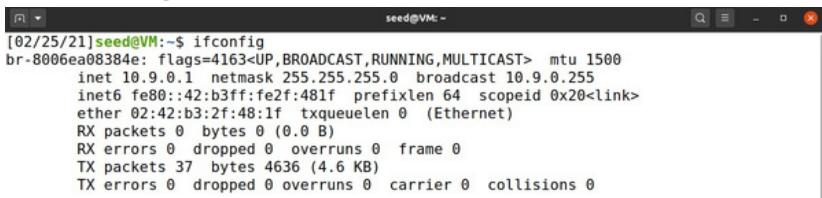
**\$ dcdown**



# 3 USING SCAPY TO SNIFF AND SPOOF PACKETS

## 3.1 TASK 1.1. SNIFFING PACKETS

\$ ifconfig



```
[02/25/21] seed@VM: ~ ifconfig
br-8006ea08384e: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.1 netmask 255.255.255.0 broadcast 10.9.0.255
        inet6 fe80::42:b3ff:fe2f:481f prefixlen 64 scopeid 0x20<link>
            ether 02:42:b3:2f:48:1f txqueuelen 0 (Ethernet)
            RX packets 0 bytes 0 (0.0 B)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 37 bytes 4636 (4.6 KB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

So the name of the Network is :  
**br-8006ea08384e**

**Python code to sniff icmp packets**

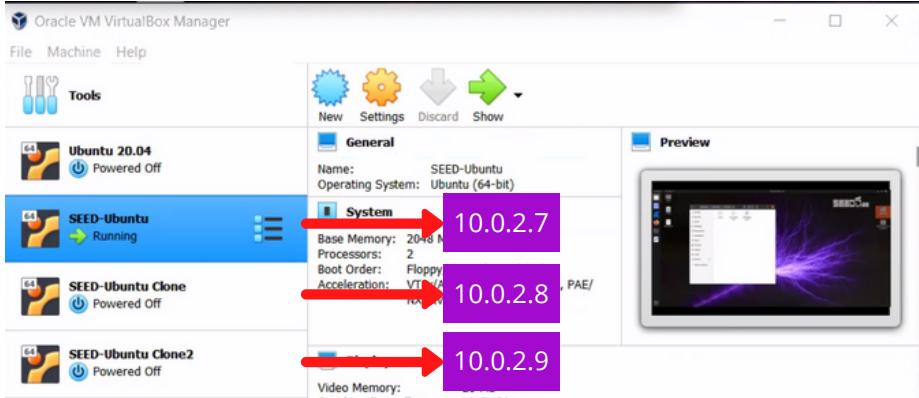


```
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
    pkt.show()
pkt = sniff(iface=["br-8006ea08384e", "enp0s3"], filter="icmp", prn=print_pkt)
```

Python 3 Tab Width: 8 Ln 7, Col 36 INS



# Using 3 VM (Seed) with different MAC Adresses Attached to NAT network



## \$ifconfig on the three VM

```
root@SEED-Ubuntu:~# ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST
        mtu 1500
        inet 10.0.2.7 netmask 255.255.255.0 brd 10.0.2.255
                ether 02:42:fa:40:7f:7f txqueuelen 1000
                RX packets 168 bytes 44172 (44.1 KB)
                RX errors 0 dropped 0 overruns 0 frame 0
                TX packets 135 bytes 18990 (18.9 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 brd 127.0.0.1
                ether 00:00:00:00:00:00 txqueuelen 1000
                RX packets 165 bytes 14271 (14.2 KB)
                RX errors 0 dropped 0 overruns 0 frame 0
                TX packets 165 bytes 14271 (14.2 KB)
                TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
root@SEED-Ubuntu:~# 

root@SEED-Ubuntu Clone:~# ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST
        mtu 1500
        inet 10.0.2.8 netmask 255.255.255.0 brd 10.0.2.255
                ether 02:42:fa:40:7f:80 txqueuelen 1000
                RX packets 90 bytes 33294 (33.2 KB)
                RX errors 0 dropped 0 overruns 0 frame 0
                TX packets 142 bytes 20131 (20.1 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 brd 127.0.0.1
                ether 00:00:00:00:00:00 txqueuelen 1000
                RX packets 169 bytes 14982 (14.9 KB)
                RX errors 0 dropped 0 overruns 0 frame 0
                TX packets 169 bytes 14982 (14.9 KB)
                TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
root@SEED-Ubuntu Clone:~# 

root@SEED-Ubuntu Clone2:~# ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST
        mtu 1500
        inet 10.0.2.9 netmask 255.255.255.0 brd 10.0.2.255
                ether 02:42:fa:40:7f:81 txqueuelen 1000
                RX packets 90 bytes 33294 (33.2 KB)
                RX errors 0 dropped 0 overruns 0 frame 0
                TX packets 142 bytes 20131 (20.1 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 brd 127.0.0.1
                ether 00:00:00:00:00:00 txqueuelen 1000
                RX packets 169 bytes 14982 (14.9 KB)
                RX errors 0 dropped 0 overruns 0 frame 0
                TX packets 169 bytes 14982 (14.9 KB)
                TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
root@SEED-Ubuntu Clone2:~# 
```

## Task 1.1.B

- Capturing only the ICMP packet: See 1.1.A
- Sniffing TCP from 10.0.2.8 with filter on port 23  
(Using Telnet cmds to check it's working)

### Code :

```
File: TCP_sniffer_port23.py
Open   TCP_sniffer_port23.py -/Desktop/LabSetup/Python Codes Save  -  X
1# Capture any TCP packet that comes from a particular IP and with
2# a destination port number 23.
3from scapy.all import *
4
5def print_pkt(pkt):
6    pkt.show()
7
8print("**** Start sniffing ***")
9pkt = sniff(iface=["br-8006ea08384e", "enp0s3"], filter="tcp and
10#src 10.0.2.8 and port 23", prn=print_pkt)
10print("**** Stop sniffing***")
```

Python Tab Width: 8 Ln 1, Col 2 INS

### Result :

```
Terminal
seedj
id      = 21282
flags   = DF
frag    = 0
ttl     = 64
proto   = tcp
chksum  = 0x55d8
src     = 10.0.2.8
dst     = 52.87.81.83
options  \
###[ TCP ]##[
    sport  = 60728
    dport  = telnet
    seq    = 1115902021
    ack    = 0
    dataofs = 10
    reserved = 0
    flags   = S
    window  = 64240
    checksum = 0xf861
    urgptr  = 0
    options  = [('MSS', 1460), ('SACKOK', b''), ('Timestamp',
p', (4228400866, 0)), ('NOP', None), ('WScale', 7)]
```

```
10 packets transmitted, 0 received, +9 errors, 100% packet loss, time
9225ms
---[02/28/21]seed@VM:~$ ping 192.168.1.10
PING 192.168.1.10 (192.168.1.10) 56(84) bytes of data.
64 bytes from 192.168.1.10: icmp_seq=1 ttl=127 time=0.842 ms
64 bytes from 192.168.1.10: icmp_seq=2 ttl=127 time=0.823 ms
64 bytes from 192.168.1.10: icmp_seq=3 ttl=127 time=0.793 ms
64 bytes from 192.168.1.10: icmp_seq=4 ttl=127 time=0.693 ms
64 bytes from 192.168.1.10: icmp_seq=5 ttl=127 time=0.628 ms
...
--- 192.168.1.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5085ms
rtt min/avg/max/mdev = 0.693/0.842/1.074/0.114 ms
[02/28/21]seed@VM:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=56 time=119 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=56 time=114 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=56 time=113 ms
c64 bytes from 8.8.8.8: icmp_seq=4 ttl=56 time=118 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=56 time=113 ms
...
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4009ms
rtt min/avg/max/mdev = 112.775/115.403/118.854/2.625 ms
[02/28/21]seed@VM:~$ telnet test.netbeez.net 23
Trying 52.87.81.83...
telnet: Unable to connect to remote host: Connection refused
[02/28/21]seed@VM:~$
```

- Capturing packets that comes from or that goes to a particular subnet.

You should not pick the subnet that your VM is attached to.

```

1 # Capture packets comes from or to go to a particular subnet. You
2 # can pick any subnet, such as 128.230.0.0/16; you should not pick
3 # the subnet that your VM is attached to.
4
5 from scapy.all import *
6
7 def print_pkt(pkt):
8     pkt.show()
9
10 print("**** Start sniffing ***")
11 pkt = sniff(iface=["br-8006ea08384e", "enp0s3"], filter="tcp and"
12     not ip host 10.0.2.7", prn=print_pkt)
13
14 # host means choosing randomly one of the subnets
15 print("**** Stop sniffing*** ")
16

```

## pcap filter host command:

Allowable primitives are:

**dst host host**

True if the IPv4/v6 destination field of the packet is *host*, which may be either an address or a name.

**src host host**

True if the IPv4/v6 source field of the packet is *host*.

**host host**

True if either the IPv4/v6 source or destination of the packet is *host*.

Any of the above host expressions can be prepended with the keywords, **ip**, **arp**, **arp**, or **ip6** as in:

**ip host host**

which is equivalent to:

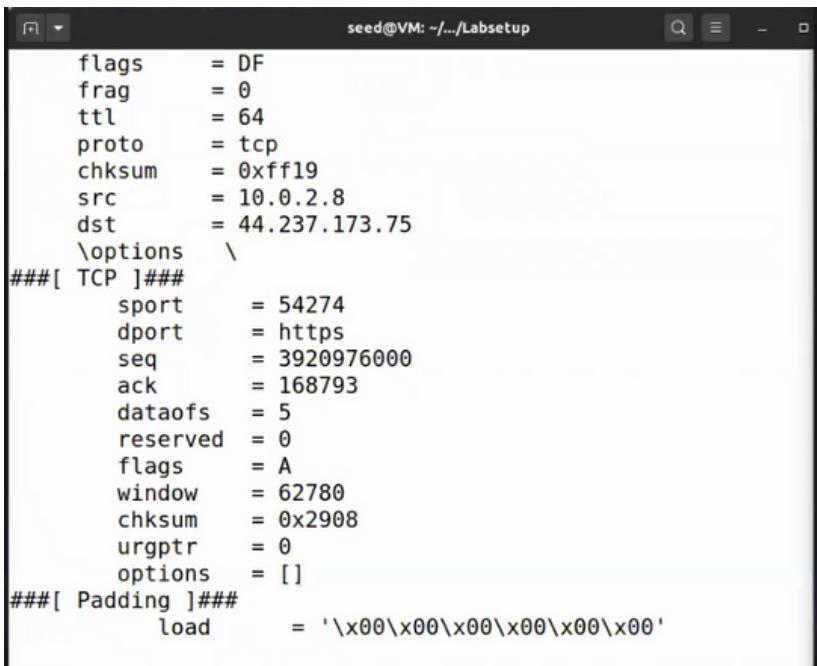
**ether proto \ip and host host**

If *host* is a name with multiple IPv4 addresses, each address will be checked for a match.

## **sudo python3 TCP\_Sniffer\_subnet.py**

```
[02/28/21]seed@VM:~/.../Labsetup$ sudo python3 TCP_sniffer_subnet.py
*** Start sniffing ***
```

**On VM 10.0.2.8 we open Firefox window and we can see that the packets were well sniffed**



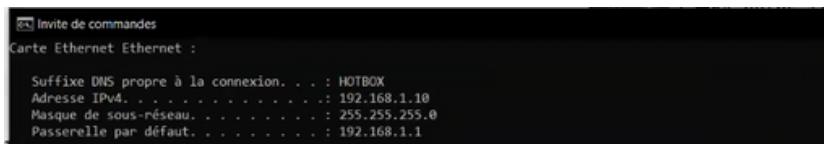
A screenshot of a terminal window titled "seed@VM: ~/.../Labsetup". The window displays a detailed dump of a captured TCP packet. The packet's header fields are listed with their values:

Field	Value
flags	= DF
frag	= 0
ttl	= 64
proto	= tcp
chksum	= 0xff19
src	= 10.0.2.8
dst	= 44.237.173.75
\options	\
###[ TCP ]###	
sport	= 54274
dport	= https
seq	= 3920976000
ack	= 168793
dataofs	= 5
reserved	= 0
flags	= A
window	= 62780
chksum	= 0x2908
urgptr	= 0
options	= []
###[ Padding ]###	
load	= '\x00\x00\x00\x00\x00\x00'

**PS:** When we open a firefox window from VM 10.0.2.7 nothing happens as expected

# 3.2 ICMP SPOOFER

## Running Ipconfig on windows (192.168.1.10)



```
invite de commandes
Carte Ethernet Ethernet :
  Suffrage DNS propre à la connexion... : HOTBOX
  Adresse IPv4... : 192.168.1.10
  Masque de sous-réseau... : 255.255.255.0
  Passerelle par défaut... : 192.168.1.1
```

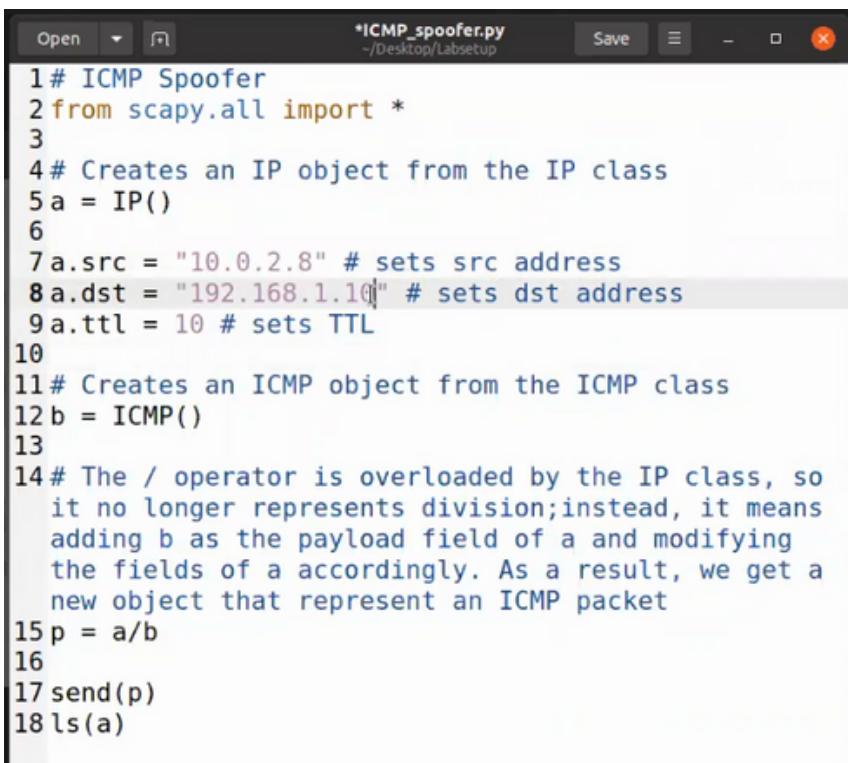
### Task 1.2

For this part we will need 3 VM.

We will run the following code on VM 10.0.2.7 which will be our **man in the middle**

The purpose is to spoof packets sent from one VM to another.

The ping will be issued by VM 10.0.2.8 to 10.0.2.9 using the command **ping -c 1 10.0.2.9** which only issues a single ping request:



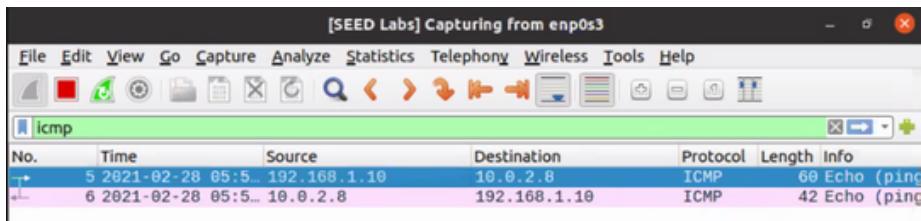
```
Open ↗ Save ⌂ ×
*ICMP_spoof.py
~/Desktop/Labsetup

1# ICMP Spoof
2 from scapy.all import *
3
4# Creates an IP object from the IP class
5a = IP()
6
7a.src = "10.0.2.8" # sets src address
8a.dst = "192.168.1.10" # sets dst address
9a.ttl = 10 # sets TTL
10
11# Creates an ICMP object from the ICMP class
12b = ICMP()
13
14# The / operator is overloaded by the IP class, so
# it no longer represents division; instead, it means
# adding b as the payload field of a and modifying
# the fields of a accordingly. As a result, we get a
# new object that represent an ICMP packet
15p = a/b
16
17 send(p)
18 ls(a)
```

# 3.2 TASK 1.2 SPOOFING ICMP PACKETS

Using Wireshark on VM 10.0.2.9, we can see the ICMP request is successfully spoofed:

Although the request was sent by VM 10.0.2.7 it seems that it has been sent by 192.168.1.10 (which is not true)



## TASK 1.3: TRACEROUTE

The objective of this task is to use Scapy to estimate the distance, in terms of number of routers, between your VM and a selected destination

```
#! /usr/bin/python
# ICMP Spoofer
from scapy.all import *
# Creates an IP object from the IP class
src = "192.168.1.10" # sets src address
dst = "10.0.2.8" # sets dst address
ttl = 1000 # sets TTL
# Creates an ICMP object from the ICMP class
ICMP = ICMP()
# The / operator is overloaded by the IP class, so it no longer represents division; instead, it means adding b as the payload field of a and modifying the fields of a accordingly. As a result, we get a new object that represent an ICMP packet
a = a/b
send(a)
# send(p)
# srl(a)
```

The terminal output shows the script attempting to send an ICMP packet with TTL=1000, but it fails with an error message indicating that the maximum TTL value is 255.

We can see that with TTL=1000 we got an error message indicating that max TTL is 255

# TASK 1.4: SNIFFING AND-THEN SPOOFING

For this part we will also need 3 VM.

We will run the following code on VM 10.0.2.7 which will be our man in the middle.

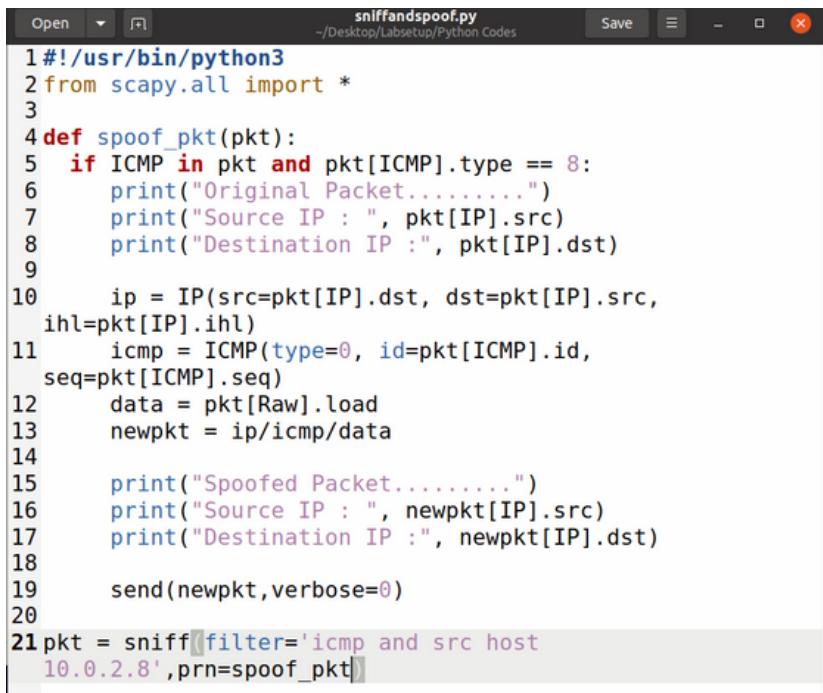
The purpose is to sniff and spoof packets sent from one VM to another.

The ping will be issued by VM 10.0.2.8 to 10.0.2.9 using the command

ping -c 1 10.0.2.9 which only issues a single ping request:

The following code is run on VM 10.0.2.7.

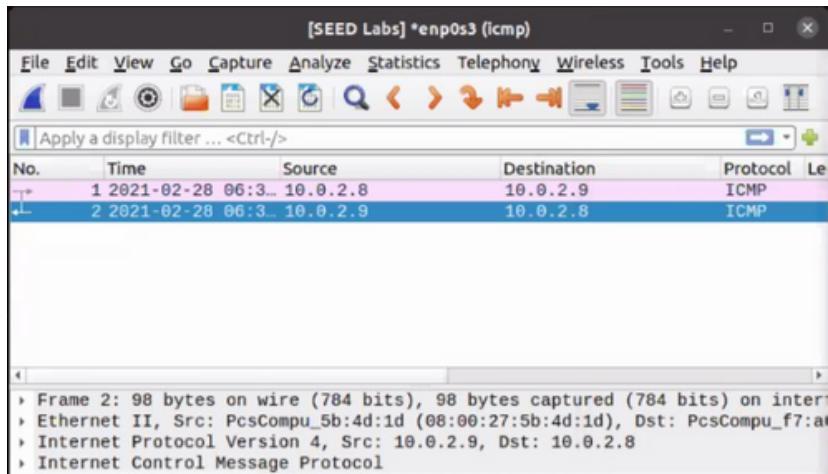
It sniffs the packet then spoofs it by swapping the source IP address with the dest IP address



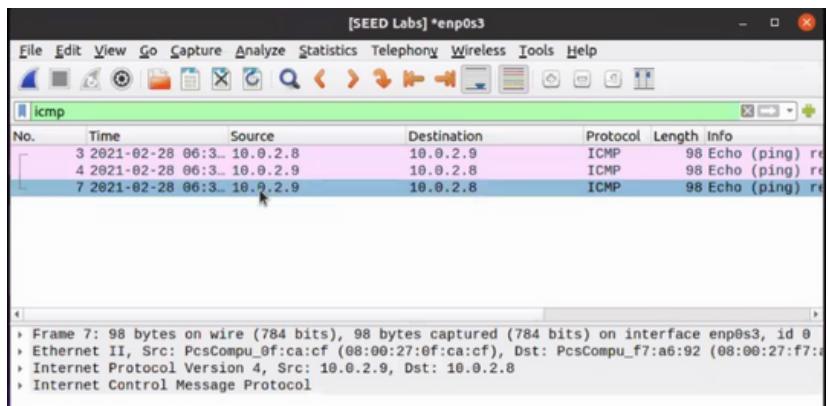
```
Open  ⌂  ~/Desktop/LabSetup/Python Codes  Save  ⌂  ×
sniffandspoof.py
1#!/usr/bin/python3
2from scapy.all import *
3
4def spoof_pkt(pkt):
5    if ICMP in pkt and pkt[ICMP].type == 8:
6        print("Original Packet.....")
7        print("Source IP : ", pkt[IP].src)
8        print("Destination IP : ", pkt[IP].dst)
9
10       ip = IP(src=pkt[IP].dst, dst=pkt[IP].src,
11          ihl=pkt[IP].ihl)
12       icmp = ICMP(type=0, id=pkt[ICMP].id,
13          seq=pkt[ICMP].seq)
14       data = pkt[Raw].load
15       newpkt = ip/icmp/data
16
17       print("Spoofed Packet.....")
18       print("Source IP : ", newpkt[IP].src)
19       print("Destination IP : ", newpkt[IP].dst)
20
21 pkt = sniff(filter='icmp and src host
  10.0.2.8', prn=spoof_pkt)
```

We are going to see the difference when we run a single ping request with and without spoofing ICMP:  
Using Wireshark on VM **10.0.2.9**

## Without spoofing



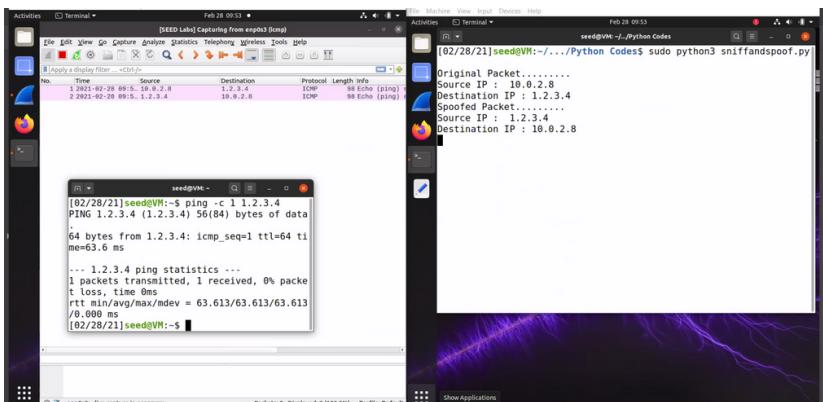
## With spoofing



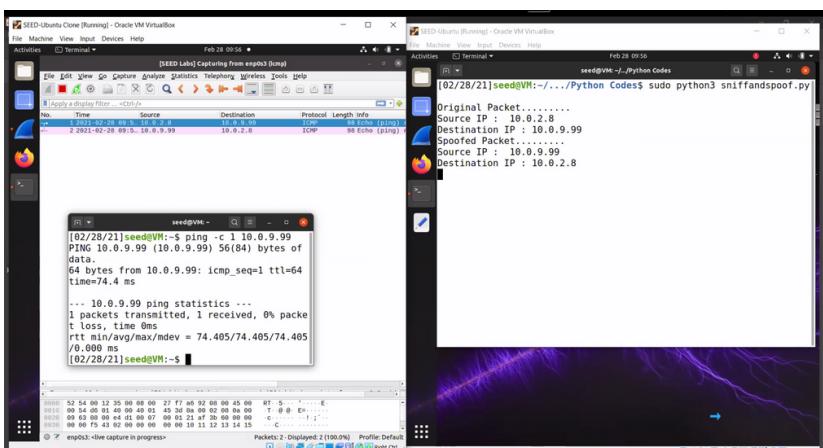
We can see that the sniff and spoof was successful: there is a third ICMP Echo Request from **10.0.2.9** to **10.0.2.8** as expected

```
ping 1.2.3.4      # a non-existing host on the Internet  
ping 10.9.0.99    # a non-existing host on the LAN  
ping 8.8.8.8      # an existing host on the Internet
```

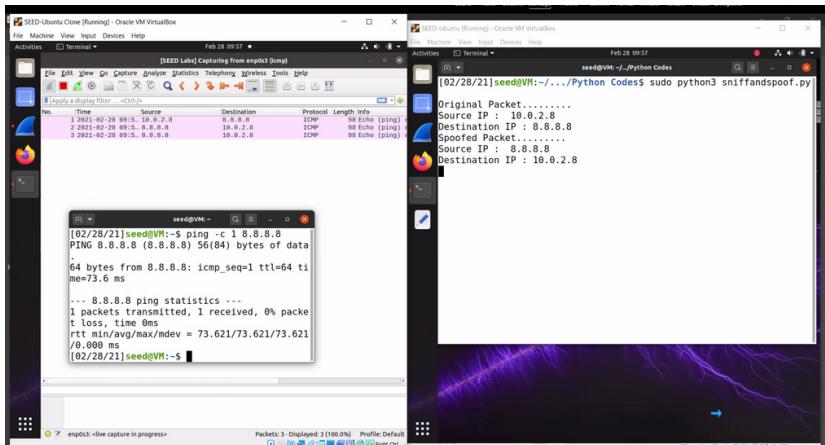
## ping 1.2.3.4 (*non-existing host on the Internet*)



## ping 10.0.9.99 (*non-existing host on the LAN*)



## ping 8.8.8.8 (an-existing host on the Internet)



**Conclusion:** Even though ping requests were issued towards non-existing hosts, the victim still got answers (packets) from them.

It means that the spoof was successful: the attacker successfully altered the communications between the two parties and the victim sees a response from a host that truly doesn't exist.

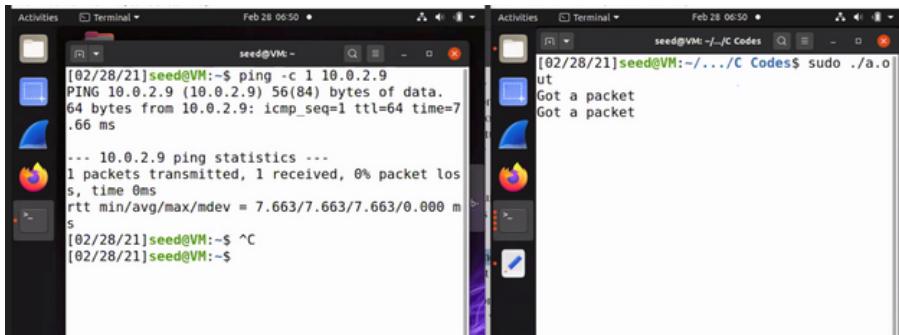
And when the ping is issued towards an existing host on the Internet (here Google) the communication is also altered successfully (IP source and IP destination swapped)

# 4 LAB TASK SET 2:

## WRITING PROGRAMS TO SNIFF AND SPOOF PACKETS

*For this set up of tasks we will compile the C code inside the host VM*

### 4.1 TASK 2.1 WRITING PACKET SNIFFING PROGRAM



```
[02/28/21]seed@VM:~$ ping -c 1 10.0.2.9
PING 10.0.2.9 (10.0.2.9) 56(84) bytes of data.
64 bytes from 10.0.2.9: icmp_seq=1 ttl=64 time=7.66 ms
...
--- 10.0.2.9 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 7.663/7.663/7.663/0.000 ms
[02/28/21]seed@VM:~$ ^C
[02/28/21]seed@VM:~$
```

```
[02/28/21]seed@VM:~/.../C Codes$ sudo ./a.out
Got a packet
Got a packet
```

*Using the code of the pdf (sniff 4.1)*

*In this task, we need to write a sniffer program in C to print out the source and destination IP addresses of each captured packet*

## **TASK 2.1A: UNDERSTANDING HOW A SNIFFER WORKS**

**Question 1. Please use your own words to describe the sequence of the library calls that are essential for sniffer programs. This is meant to be a summary, not detailed explanation like the one in the tutorial or book.**

The following calls are essential for this sniffer program:

- **pcap\_lookupdev** : Find the default device on which to capture
- **pcap\_lookupnet** : Is used to determine the IPv4 network number and mask associated with the network device.
- **pcap\_open\_live** : Open a device to start sniffing
- **pcap\_datalink** : Get the link-layer header type to know what type
- **pcap\_compile** : Compile a filter expression
- **pcap\_setfilter** : Sets the compiled filter
- **pcap\_loop** : Process packets from the capture
- **pcap\_freecode** : Frees up allocated memory generated
- **pcap\_close** : Closes the sniffing session

**Question 2. Why do you need the root privilege to run a sniffer program? Where does the program fail if it is executed without the root privilege?**

We need root privilege to access to the Network Interface Card which needs admin authorization

**For the following questions we are going to use the given main. We only changed eth3 to enp0s3:**

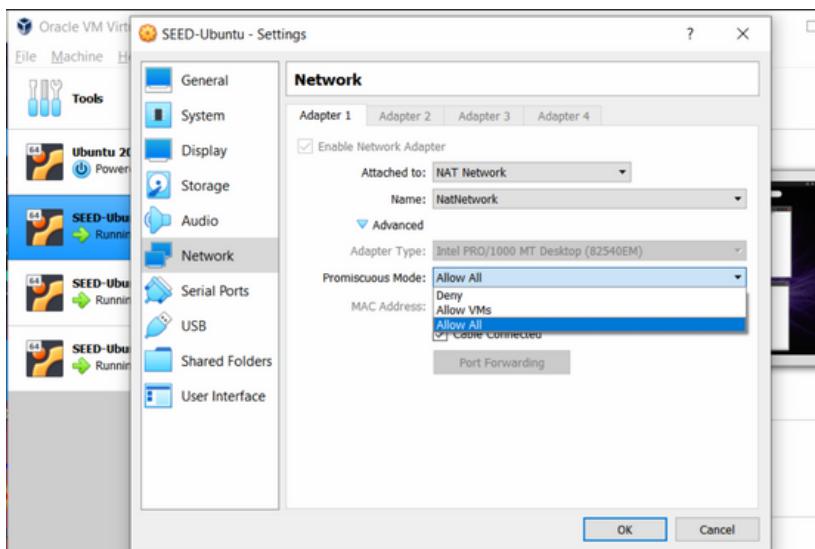
```
int main()
{
    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];
    struct bpf_program fp;
    char filter_exp[] = "ip proto icmp";
    bpf_u_int32 net;
    // Step 1: Open live pcap session on NIC with name eth3
    // Students needs to change "eth3" to the name
    // found on their own machines (using ifconfig).
    handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
    // Step 2: Compile filter exp into BPF pseudo-code
    pcap_compile(handle, &fp, filter_exp, 0, net);
    pcap_setfilter(handle, &fp);
    // Step 3: Capture packets
    pcap_loop(handle, -1, got_packet, NULL);
    pcap_close(handle);
    return 0;
    //Close the handle
}
```

**Question 3. Please turn on and turn off the promiscuous mode in your sniffer program. Can you demonstrate the difference when this mode is on and off? Please describe how you can demonstrate this.**

For this part we will also need 3 VMs.

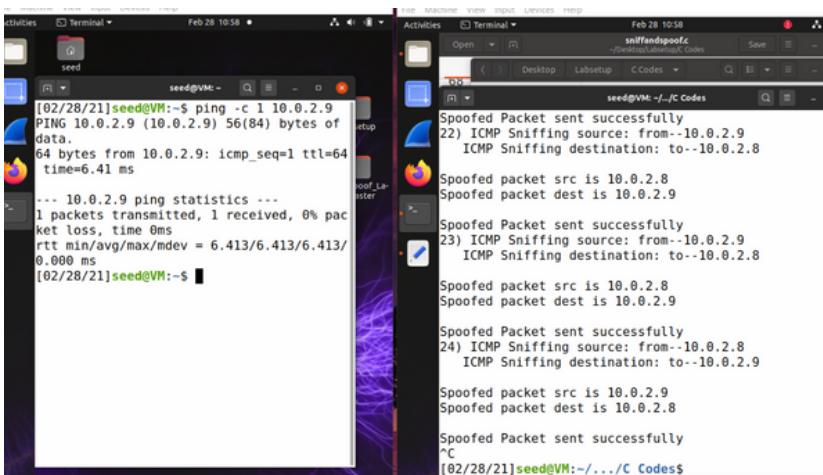
We will run the sniffandspoof.c code on VM 10.0.2.7 which will be our **man in the middle** but we will test the **3 different options** we have for the promiscuous mode in VirtualBox (changes only for the VM being the man in the middle)

Ping requests are sent from VM 10.0.2.8 to VM 10.0.2.9):



**The available modes are: Deny, Allow VMs and Allow All**

# Running sniffandspoof on VM 10.0.2.7 with Promiscous mode Allow All



The image shows two side-by-side terminal windows on a Linux desktop environment. Both windows have the title 'Activities Terminal' and show the command line interface for a user named 'seed'.

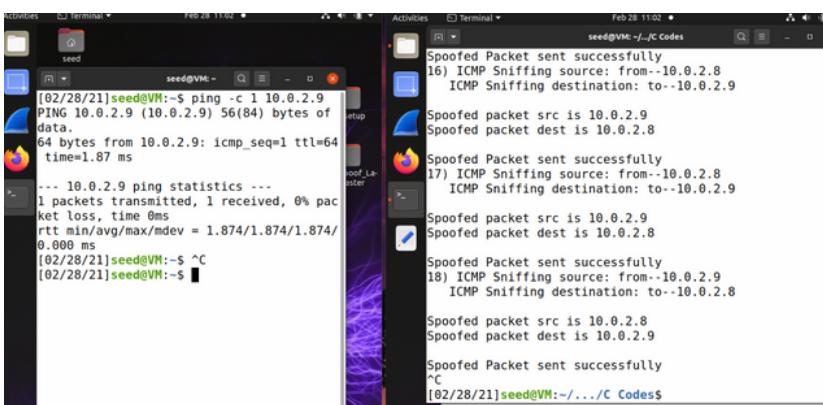
The left terminal window displays the output of a ping command:

```
[02/28/21]seed@VM:~$ ping -c 1 10.0.2.9
PING 10.0.2.9 (10.0.2.9) 56(84) bytes of data.
64 bytes from 10.0.2.9: icmp_seq=1 ttl=64 time=6.41 ms
...
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 6.413/6.413/6.413/0.000 ms
[02/28/21]seed@VM:~$
```

The right terminal window displays the output of the sniffandspoof tool, which is spoofing ICMP packets:

```
Sniffed Packet sent successfully
22) ICMP Sniffing source: from--10.0.2.9
    ICMP Sniffing destination: to--10.0.2.8
Spoofed packet src is 10.0.2.8
Spoofed packet dest is 10.0.2.9
Spoofed Packet sent successfully
23) ICMP Sniffing source: from--10.0.2.9
    ICMP Sniffing destination: to--10.0.2.8
Spoofed packet src is 10.0.2.8
Spoofed packet dest is 10.0.2.9
Spoofed Packet sent successfully
24) ICMP Sniffing source: from--10.0.2.8
    ICMP Sniffing destination: to--10.0.2.9
Spoofed packet src is 10.0.2.9
Spoofed packet dest is 10.0.2.8
Spoofed Packet sent successfully
25) ICMP Sniffing source: from--10.0.2.8
    ICMP Sniffing destination: to--10.0.2.9
Spoofed packet src is 10.0.2.9
Spoofed packet dest is 10.0.2.8
[02/28/21]seed@VM:~/.../C_Codes$
```

# Running sniffandspoof on VM 10.0.2.7 with Promiscous mode Allow VMs



The image shows two side-by-side terminal windows on a Linux desktop environment. Both windows have the title 'Activities Terminal' and show the command line interface for a user named 'seed'.

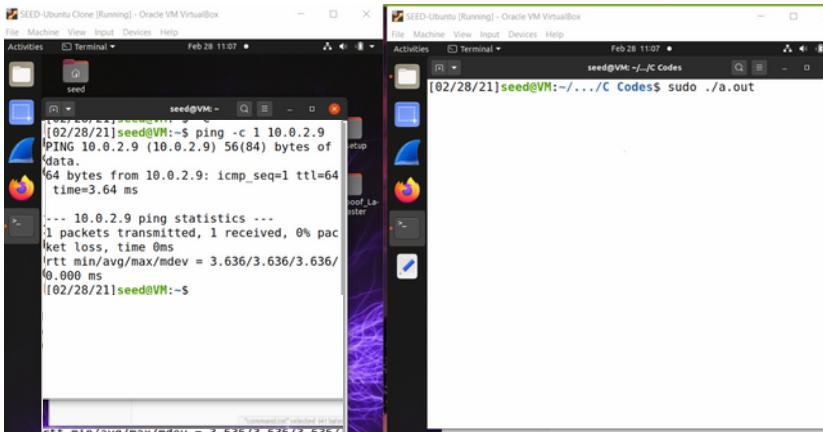
The left terminal window displays the output of a ping command:

```
[02/28/21]seed@VM:~$ ping -c 1 10.0.2.9
PING 10.0.2.9 (10.0.2.9) 56(84) bytes of data.
64 bytes from 10.0.2.9: icmp_seq=1 ttl=64 time=1.87 ms
...
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.874/1.874/1.874/0.000 ms
[02/28/21]seed@VM:~$ ^C
[02/28/21]seed@VM:~$
```

The right terminal window displays the output of the sniffandspoof tool, which is spoofing ICMP packets:

```
Sniffed Packet sent successfully
16) ICMP Sniffing source: from--10.0.2.8
    ICMP Sniffing destination: to--10.0.2.9
Spoofed packet src is 10.0.2.9
Spoofed packet dest is 10.0.2.8
Spoofed Packet sent successfully
17) ICMP Sniffing source: from--10.0.2.8
    ICMP Sniffing destination: to--10.0.2.9
Spoofed packet src is 10.0.2.9
Spoofed packet dest is 10.0.2.8
Spoofed Packet sent successfully
18) ICMP Sniffing source: from--10.0.2.9
    ICMP Sniffing destination: to--10.0.2.8
Spoofed packet src is 10.0.2.8
Spoofed packet dest is 10.0.2.9
Spoofed Packet sent successfully
19) ICMP Sniffing source: from--10.0.2.8
    ICMP Sniffing destination: to--10.0.2.9
Spoofed packet src is 10.0.2.8
Spoofed packet dest is 10.0.2.9
[02/28/21]seed@VM:~/.../C_Codes$
```

# Running sniffandspoof on VM 10.0.2.7 with Promiscous mode Deny



We can see that in Deny mode absolutely no packets are sniffed nor spoofed.

The reason is because Promiscuous mode allows a network sniffer to pass all the traffic from a network controller and not just the traffic that the network controller was intended to receive.

PS: It's possible to manually switch between promiscuous modes directly in the code:

```
/* promisc mode on */
handle = pcap_open_live(dev, SNAP_LEN, 1, 1000, errbuf);

/* promisc mode off */
handle = pcap_open_live(dev, SNAP_LEN, 0, 1000, errbuf);
```

# TASK 2.1B: WRITING FILTERS

***Writing filter expressions for the sniffer program to capture each of the followings:***

- Capture the ICMP packets between two specific hosts**

We can filter only the ICMP packets between two given hosts by modifying the `filter_exp[]` string this way:

```
int main()
{
    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];
    struct bpf_program fp;
    // char filter_exp[] = "ip proto icmp";

    // ICMP packets between this host and 8.8.8.8
    char filter_exp[] = "icmp and (src host 10.0.2.8 and dst host 8.8.8.8) or (src host 8.8.8.8 and dst host 10.0.2.8);

    bpf_u_int32 net;
    // Step 1: Open live pcap session on NIC with name eth3
    // Students needs to change "eth3" to the name
    // found on their own machines (using ifconfig).
    handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
    // Step 2: Compile filter_exp into BPF pseudo-code
    pcap_compile(handle, &fp, filter_exp, 0, net);
    pcap_setfilter(handle, &fp);
    // Step 3: Capture packets
    pcap_loop(handle, -1, got_packet, NULL);
    pcap_close(handle);
    return 0;
    //Close the handle
}
```

**Spoofed successfully with the filter**

The screenshot shows two terminal windows side-by-side. The left window displays the command `ping -c 1 8.8.8.8` and its output, which includes statistics like packet loss and round-trip time. The right window shows the output of a BPF filter program, specifically the `sniffandspoof.c` code, which is sniffing ICMP traffic between the source and destination hosts specified in the filter expression.

```
[02/28/21]seed@VM:~$ ping -c 1 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=56
time=115 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 115.072/115.072/115.072/0.000 ms
[02/28/21]seed@VM:~$ ^C
[02/28/21]seed@VM:~$ █
```

```
[02/28/21]seed@VM:~/.../Codes$ gcc sniffandspoof.c -lpcap
[02/28/21]seed@VM:~/.../Codes$ sudo ./a.out
1) ICMP Sniffing source: from-10.0.2.8
   ICMP Sniffing destination: to--8.8.8.8
Spoofed packet src is 8.8.8.8
Spoofed packet dest is 10.0.2.8
Spoofed Packet sent successfully
2) ICMP Sniffing source: from--8.8.8.8
   ICMP Sniffing destination: to-10.0.2.8
Spoofed packet src is 10.0.2.8
Spoofed packet dest is 8.8.8.8
Spoofed Packet sent successfully
3) ICMP Sniffing source: from--8.8.8.8
   ICMP Sniffing destination: to--8.8.8.8
Spoofed packet src is 8.8.8.8
Spoofed packet dest is 10.0.2.8
Spoofed Packet sent successfully
4) ICMP Sniffing source: from..R R R R
   ICMP Sniffing destination: to--8.8.8.8
```

- Capture the TCP packets with a destination port number in the range from 10 to 100.

We can capture only the TCP packets with a destination port number in the given range from 10 to 100 by modifying the `filter_exp[]` string this way:

```
int main()
{
    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];
    struct bpf_program fp;

    // TCP packets with dest port 10-100
    char filter_exp[] = "tcp dst portrange 10-100";

    bpf_u_int32 net;
    // Step 1: Open live pcap session on NIC with name eth3
    // Students needs to change "eth3" to the name
    // found on their own machines (using ifconfig).
    handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
    // Step 2: Compile filter_exp into BPF psuedo-code
    pcap_compile(handle, &fp, filter_exp, 0, net);
    pcap_setfilter(handle, &fp);
    // Step 3: Capture packets
    pcap_loop(handle, -1, got_packet, NULL);
    pcap_close(handle);
    return 0;
    //Close the handle
}
```

No.	Time	Source	Destination	Protocol	Length	Info
5	2021-03-01 14:10.0.2.7		34.107.221.82	TCP	74	56558 -- 80 [SYN]
13	2021-03-01 14:10.0.2.7		34.107.221.82	TCP	69	88 - 56558 [SYN]
15	2021-03-01 14:10.0.2.7		34.107.221.82	TCP	54	65558 -- 80 [ACK]
16	2021-03-01 14:10.0.2.7		34.107.221.82	HTTP	350	GET /success.txt
42	2021-03-01 14:10.0.2.7		34.107.221.82	TCP	69	88 - 56558 [ACK]
43	2021-03-01 14:10.0.2.7		34.107.221.82	HTTP	274	HTTP/1.1 200 OK
44	2021-03-01 14:10.0.2.7		34.107.221.82	TCP	54	56558 -- 80 [ACK]
55	2021-03-01 14:10.0.2.7		34.107.221.82	TCP	74	56564 -- 80 [SYN]
60	2021-03-01 14:10.0.2.7		34.107.221.82	TCP	69	88 - 56564 [ACK]
65	2021-03-01 14:10.0.2.7		34.107.221.82	TCP	54	56564 -- 80 [ACK]
66	2021-03-01 14:10.0.2.7		34.107.221.82	HTTP	355	GET /success.txt
70	2021-03-01 14:34.107.221.82		10.0.2.7	HTTP	274	HTTP/1.1 200 OK
71	2021-03-01 14:10.0.2.7		34.107.221.82	TCP	54	56564 -- 80 [ACK]
83	2021-03-01 14:10.0.2.7		93.184.220.29	TCP	74	46454 -- 80 [SYN]
96	2021-03-01 14:10.0.2.7		93.184.220.29	TCP	69	88 - 33446 [ACK]
97	2021-03-01 14:10.0.2.7		93.184.220.29	TCP	54	33446 -- 80 [ACK]
98	2021-03-01 14:10.0.2.7		93.184.220.29	OCSP	433	Request
117	2021-03-01 14:10.0.2.7		93.184.220.29	OCSP	853	Response
118	2021-03-01 14:10.0.2.7		93.184.220.29	TCP	54	88 - 33446 [ACK]
362	2021-03-01 14:10.0.2.7		93.184.220.29	OCSP	433	Request
367	2021-03-01 14:10.0.2.7		93.184.220.29	TCP	69	88 - 33446 [ACK]
313	2021-03-01 14:10.0.2.7		216.58.210.195	TCP	74	46452 -- 80 [SYN]
314	2021-03-01 14:10.0.2.7		216.58.210.195	TCP	74	46454 -- 80 [SYN]
315	2021-03-01 14:10.0.2.7		216.58.210.195	OCSP	853	Request
319	2021-03-01 14:10.0.2.7		93.184.220.29	TCP	54	33449 -- 80 [ACK]
321	2021-03-01 14:10.0.2.7		216.58.210.195	TCP	69	88 - 46454 [ACK]
322	2021-03-01 14:10.0.2.7		216.58.210.195	OCSP	439	Request
324	2021-03-01 14:10.0.2.7		216.58.210.195	TCP	69	88 - 46452 [SYN]

[SEED Labs] \*enp0s3

```
From: 10.0.2.7
To: 34.107.221.82
Protocol: TCP

From: 10.0.2.7
To: 34.107.221.82
Protocol: TCP

From: 10.0.2.7
To: 216.58.210.195
Protocol: TCP

From: 10.0.2.7
To: 34.107.221.82
Protocol: TCP

From: 10.0.2.7
To: 216.58.210.195
Protocol: TCP

From: 10.0.2.7
To: 93.184.220.29
Protocol: TCP

From: 10.0.2.7
To: 34.107.221.82
Protocol: TCP

From: 10.0.2.7
To: 93.184.220.29
Protocol: TCP
```

# TASK 2.1C: SNIFFING PASSWORDS

```
int main()
{
    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];
    struct bpf_program fp;

    // TCP packets port 23
    char filter_exp[] = "tcp port 23";

    bpf_u_int32 net;

    // Step 1: Open live pcap session on NIC with name enp0s3
    handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);

    // Step 2: Compile filter_exp into BPF pseudo-code
    pcap_compile(handle, &fp, filter_exp, 0, net);
    pcap_setfilter(handle, &fp);

    // Step 3: Capture packets
    pcap_loop(handle, -1, got_packet, NULL);

    pcap_close(handle); //Close the handle
    return 0;
}
```

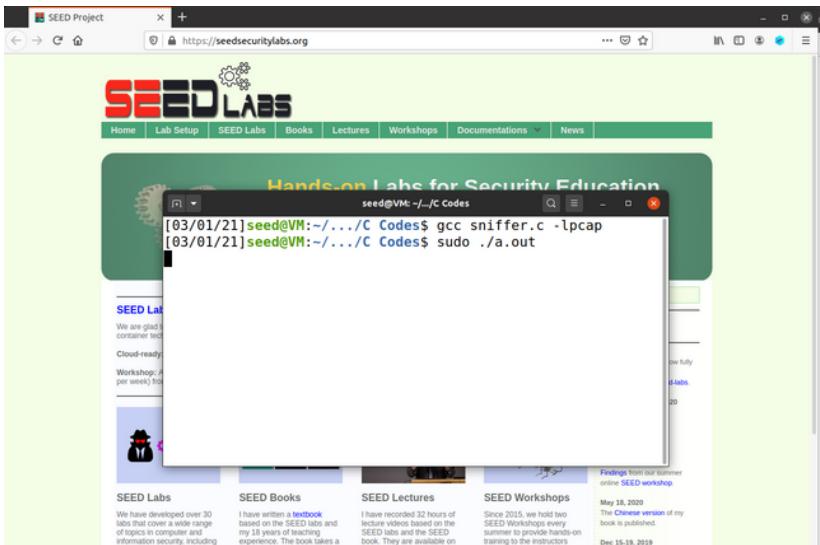
Packets are well sniffed when using telnet  
(port 23) as expected

The screenshot shows two terminal windows side-by-side. The left window shows a failed telnet connection attempt:

```
[03/01/21]seed@VM:~$ telnet test.netbeez.net 23
Trying 52.87.81.83...
telnet: Unable to connect to remote host: Connection refused
[03/01/21]seed@VM:~$
```

The right window shows the captured packets from the sniffing process:

```
[03/01/21]seed@VM:~/.../C Codes$ gcc sniffer.c -lpcap
[03/01/21]seed@VM:~/.../C Codes$ sudo ./a.out
From: 10.0.2.7
To: 52.87.81.83
Protocol: TCP
From: 10.0.2.7
To: 52.87.81.83
Protocol: TCP
From: 52.87.81.83
To: 10.0.2.7
Protocol: TCP
```



**Nothing is captured when opening Firefox  
(since port 23 isn't used) as expected  
(Indeed, HTTP use port 80)**

## 4.2 TASK 2.2: SPOOFING

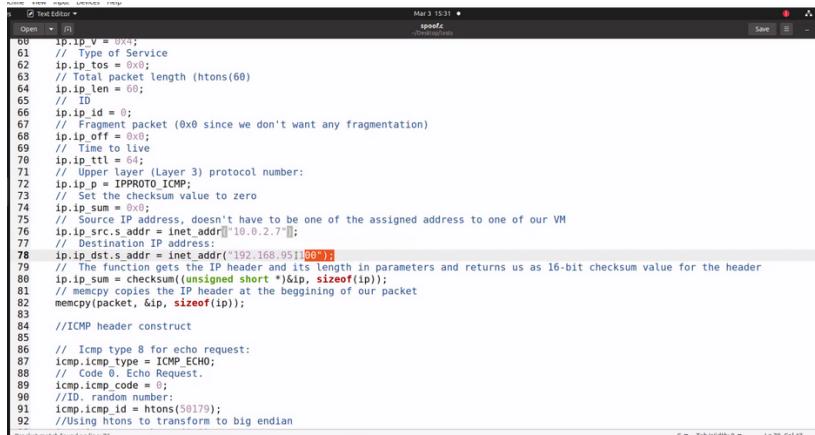
When a normal user sends out a packet, operating systems usually do not allow the user to set all the fields in the protocol headers (such as TCP, UDP, and IP headers). OSes will set most of the fields, while only allowing users to set a few fields, such as the destination IP address, the destination port number, etc.

However, if users have the root privilege, they can set any arbitrary field in the packet headers. This is called packet spoofing, and it can be done through raw sockets.

Raw sockets give programmers the absolute control over the packet construction, allowing programmers to construct any arbitrary packet, including setting the header fields and the payload. Using raw sockets is quite straightforward; it involves four steps: (1) create a raw socket, (2) set socket option, (3) construct the packet, and (4) send out the packet through the raw socket.

# TASK 2.2.A: WRITE A SPOOFING PROGRAM.

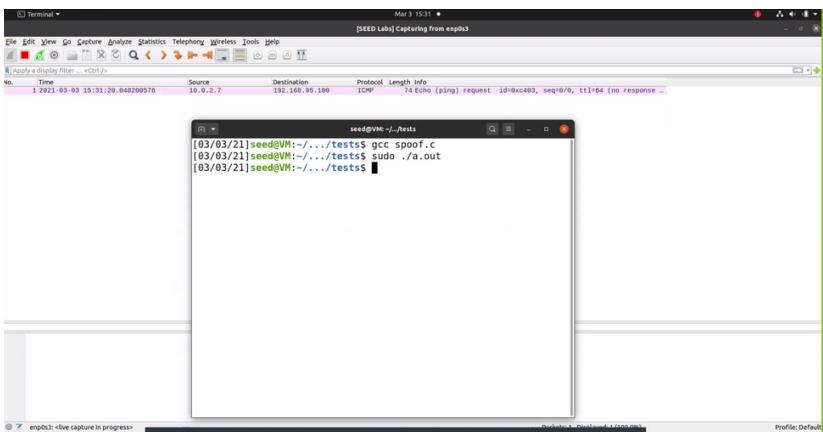
## Spoof Code



```
00 1B.ip.v = 0x4; // Version
01 // Type of Service
02 ip.ip.tos = 0x0;
03 // Total packet length (htons(60))
04 ip.ip.len = 0x0;
05 // ID
06 ip.ip.id = 0;
07 // Fragment packet (0x0 since we don't want any fragmentation)
08 ip.ip.off = 0x0;
09 // Time to live
10 ip.ip.ttl = 0x1;
11 // Upper layer (Layer 3) protocol number:
12 ip.ip.p = IPPROTO_ICMP;
13 // Set the checksum value to zero
14 ip.ip.sum = 0x0;
15 // Source IP address, doesn't have to be one of the assigned address to one of our VM
16 ip.ip.saddr = htonl("10.0.2.7");
17 // Destination IP address:
18 ip.ip.dst.s.addr = inet_addr("192.168.95.100");
19 // The function gets the IP header and its length in parameters and returns us as 16-bit checksum value for the header
20 ip.ip.sum = checksum((unsigned short *)ip, sizeof(ip));
21 // memcpy copies the IP header at the beginning of our packet
22 memcpy(packet, &ip, sizeof(ip));
23
24 // ICMP header construct
25
26 // ICMP type 8 for echo request:
27 icmp.icmp.type = ICMP_ECHO;
28 // Code 0. Echo Request.
29
30 // ID: random number:
31 icmp.icmp.id = htons(50179);
32 // Using htons to transform to big endian
```

# TASK 2.2.B: SPOOF AN ICMP ECHO REQUEST.

Packet successfully spoofed:



- **Question 4.** Can you set the IP packet length field to an arbitrary value, regardless of how big the actual packet is?

The length field in this case should be the length of the IP packet, if not the sendto method (or function) will show an error 'Invalid Argument'. So yes, it is important that it is the length of the IP packet.

- **Question 5.** Using the raw socket programming, do you have to calculate the checksum for the IP header?

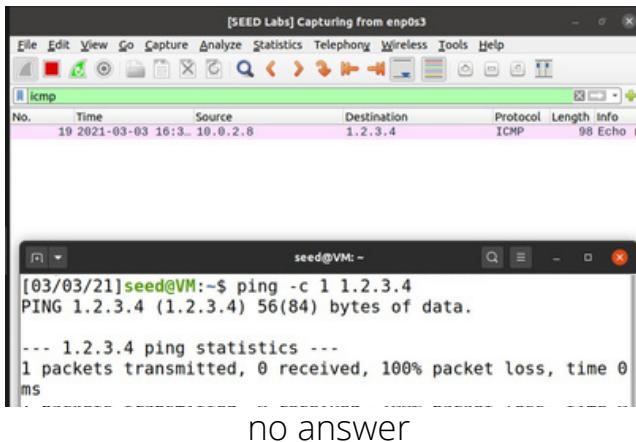
No you don't need to calculate the checksum, it will be filled by the system.

- **Question 6.** Why do you need the root privilege to run the programs that use raw sockets? Where does the program fail if executed without the root privilege?

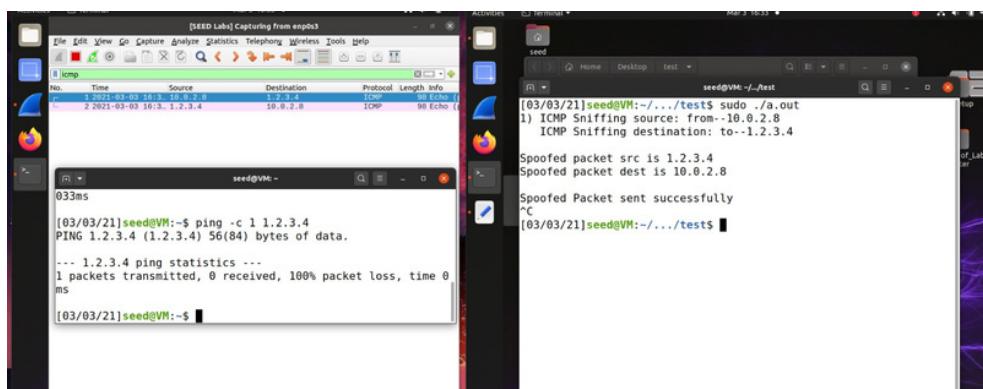
It's restricted to root because it would break some rules for networking that are in place, without root you can't bind a port lower than 1024. With raw sockets you can simulate a server on any port and spoof custom packets which can interfere with inbound traffic

## 4.3 TASK 2.3: SNIFF AND THEN SPOOF

Sending a ping request to a non-existing host  
(1.2.3.4) before running sniffandspoof.c



Sending a ping request to a non-existing host  
(1.2.3.4) after running sniffandspoof.c



We can see that Wireshark displays a successful ping request even though no packet were really sent (100% packet loss), and in the attacker terminal we can also see that the source IP and the destination IP have been successfully swapped, meaning that the spoofing was successful.