

Informe Laboratorio 1

Sección 4

Renato Óscar Benjamín Contreras Carvajal
e-mail: renato.contreras@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	5
3.3. Actividad 3	12

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 César. 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.

```

0 larycxpajorj h bnpdarmjm nw anmnb
1 kzqxbwozinqi g amoczqlil mv zmlma
2 jypwavyhmpfh f zlnbypkhk lu ylk lz
3 ixovzumxglog e ykmaxojgj kt xkjky
4 hwnuytlwfknd d xjlzwnifi js wji jx
5 gvmtxskvejme c wikyvmeheh ir vihiw
6 fulswrjudild b vhxulgdg hq uhghv
7 etkrvqitchkc a ugiwtkfcef gp tgfgu
8 dsjquphsbgjb z tfhvsjebe fo sfeft
9 criptografia y seguridad en redes
10 bqhosnfqzehz x rdftqhczc dm qdcdr
11 apgnrmepdygy w qcespgbyb cl pcbcq
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 vkbmhzktybt 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 htvi jgxsp s tc gtsth
21 qfwdhcufotwo m gsuifwr or sb f srs g
22 pevcbgtensvn l frthevqnq ra erqrf
23 odubfasdmrum k eqsgdupmp qz dqpqe
24 nctaezrclqtl j dprfctolo py cpopd
25 mbszdyqbksk i coqeb snkn 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

Para comenzar, se genera el programa solicitado en Python 3 utilizando ChatGPT.

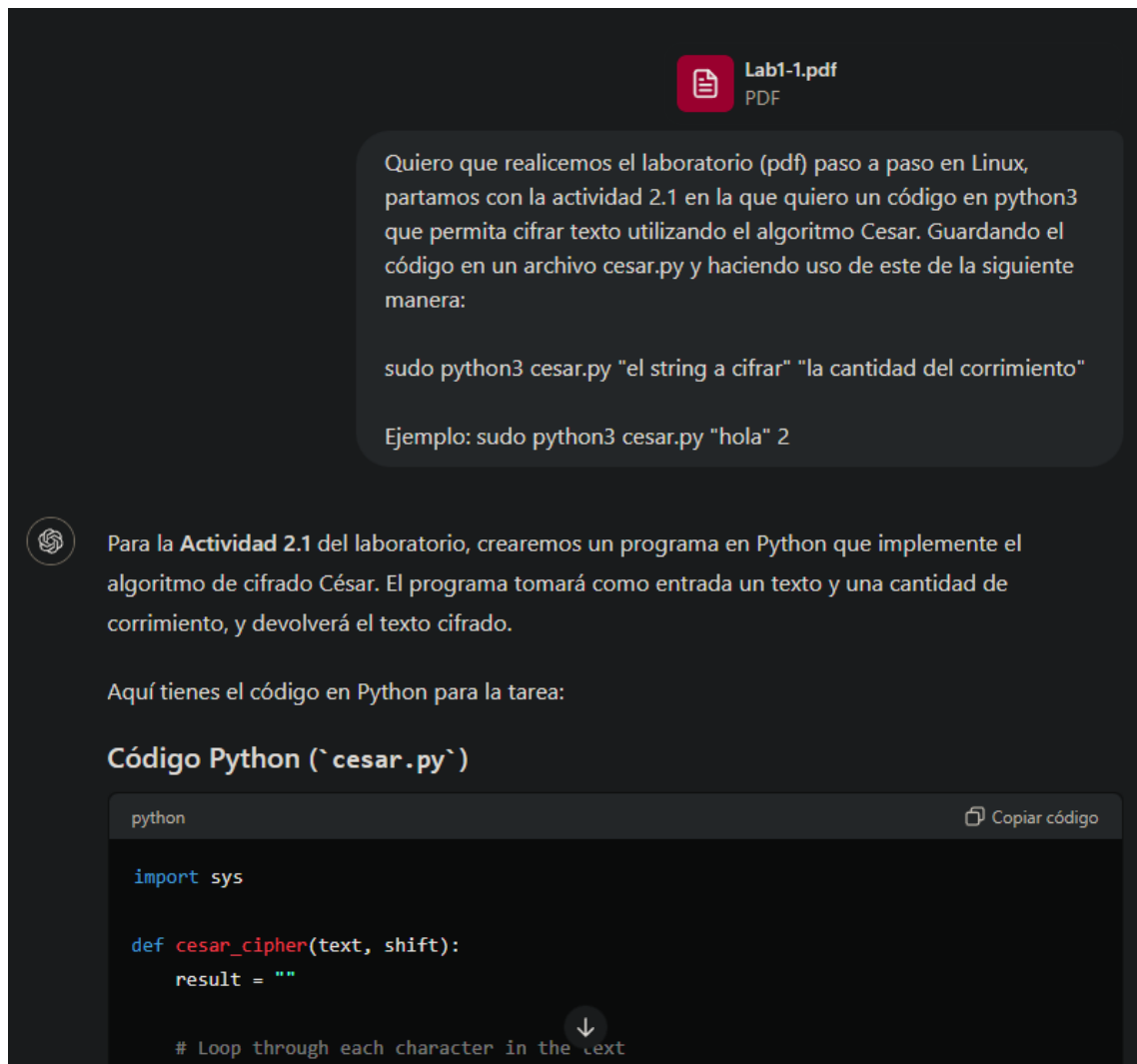
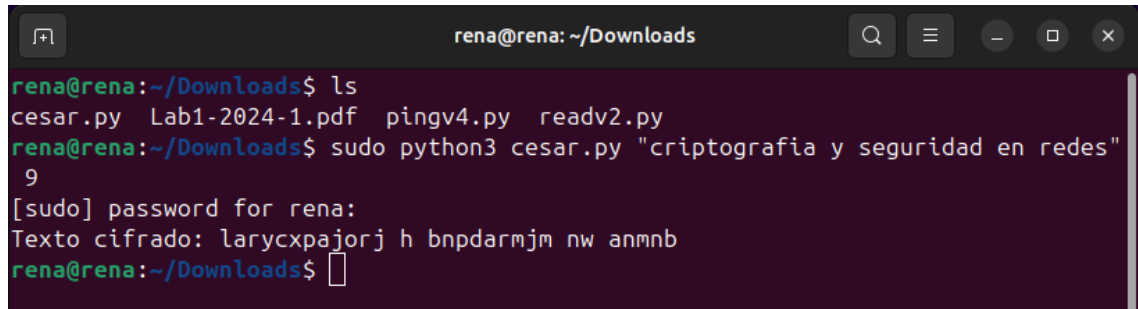


Figura 1: Prompt utilizado para la primera actividad.

El código obtenido ('cesar.py') permite cifrar texto utilizando el algoritmo César, usando el alfabeto inglés, como se observa en la siguiente imagen. Además, el comando para utilizarlo es: 'sudo python3 cesar.py 'el string a cifrar entre comillas' 'la cantidad del corrimiento'.

A terminal window titled 'rena@rena: ~/Downloads' with standard window controls. The terminal shows the following commands and output:

```
rena@rena:~/Downloads$ ls
cesar.py  Lab1-2024-1.pdf  pingv4.py  readv2.py
rena@rena:~/Downloads$ sudo python3 cesar.py "criptografia y seguridad en redes"
9
[sudo] password for rena:
Texto cifrado: larycxpajorj h bnpdarmjm nw anmnb
rena@rena:~/Downloads$
```

Figura 2: Uso del programa de codificación César.

3.2. Actividad 2

Se genera un programa, en python3 utilizando ChatGPT, que permite enviar los caracteres del string (del paso 1) en varios paquetes ICMP request, para ello se realiza lo siguiente:

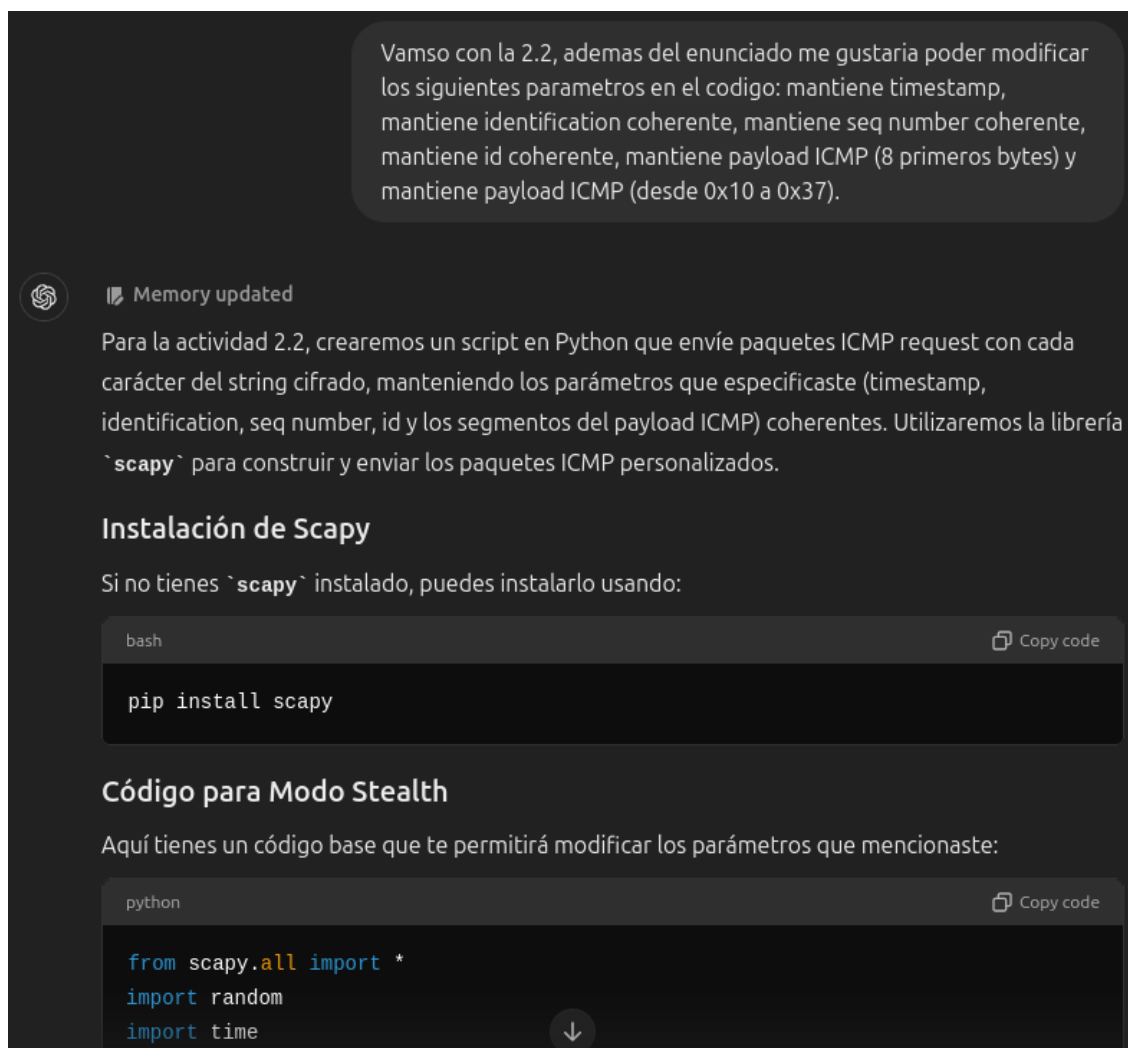


Figura 3: Prompt utilizado para la segunda actividad.


El código obtenido permite enviar los caracteres del string (el del paso 1) en varios paquetes ICMP request. Sin embargo, no cumple con todos los datos requeridos por la pauta, por lo que se realizaron ajustes en cada parámetro para conseguir el resultado deseado.

```
Protocol: ICMP (1)
Header Checksum: 0xa385 [validation disabled]
[Header checksum status: Unverified]
Source Address: 10.0.2.15
Destination Address: 8.8.8.8
▼ Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0xbaa0 [correct]
  [Checksum Status: Good]
  Identifier (BE): 4747 (0x128b)
  Identifier (LE): 35602 (0x8b12)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  [Response frame: 4]
▼ Data (48 bytes)
  Data: 6c0000000000000000101112131415161718191a1b1c1d1e
  [Length: 48]
```

Figura 4: Captura de datos prematura.

Al realizar una prueba, se observó que no hay forma de corroborar el timestamp, ya que debería estar debajo del 'sequence number', por lo que se investiga el motivo de esta ausencia.

pero debo mantener el mismo timestamp para todos los paquetes, eso es posible? porque?


 Sí, es posible mantener el mismo timestamp para todos los paquetes ICMP, pero hay algunas consideraciones a tener en cuenta:

¿Qué es el Timestamp en ICMP?

El timestamp en ICMP se refiere generalmente al tiempo en el que se envía el paquete, pero es importante notar que:

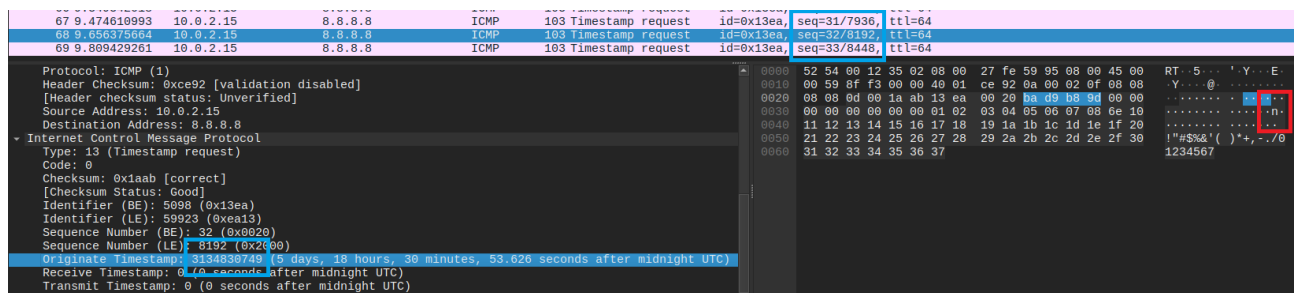
1. **No todos los paquetes ICMP tienen un campo de timestamp:** Solo los mensajes ICMP de tipo *Timestamp Request* (tipo 13) y *Timestamp Reply* (tipo 14) tienen un campo explícito de timestamp.
2. **Para mensajes ICMP tipo "Echo Request" (tipo 8), el timestamp no es parte del estándar ICMP:** Sin embargo, se puede incluir un valor personalizado en el payload de datos del paquete.

Cómo Mantener el Mismo Timestamp para Todos los Paquetes

Si deseas mantener el mismo timestamp en cada paquete ICMP, necesitas definir un campo de timestamp en el payload del paquete y asegurarte de que este valor sea constante para todos los paquetes. Esto es lo que haces actualmente en tu código, pero solo necesitas almacenarlo en el

Figura 5: Prompt para obtener información sobre el timestamp.

El protocolo ICMP posee estructuras específicas para cada tipo de mensaje. Al utilizar los campos de timestamp ('ts_ori', 'ts_rx', 'ts_tx') en un mensaje ICMP tipo 8 (Echo Request), es posible que herramientas como Wireshark no los interpreten correctamente. Cabe mencionar que el laboratorio fue realizado en una máquina virtual con Ubuntu, mientras que el sistema operativo anfitrión era Windows, lo que podría ser la causa del problema. Se realizaron pruebas alternando entre los tipos 8 y 13; este último es donde se observa el timestamp y se utilizará solo en la siguiente captura. El resto de la actividad se realiza con mensajes de tipo 8 para poder visualizar el campo 'data'.



The image shows a Wireshark packet capture of an ICMP Timestamp Request (Type 13). The packet list on the left shows three packets, all of which are ICMP Timestamp Requests. The selected packet (packet 69) is expanded in the packet details pane on the left, showing the following fields:

- Protocol: ICMP (1)
- Header Checksum: 0xce92 [validation disabled]
- [Header checksum status: Unverified]
- Source Address: 10.0.2.15
- Destination Address: 8.8.8.8
- Internet Control Message Protocol
- Type: 13 (Timestamp request)
- Code: 0
- Checksum: 0x1aab [correct]
- [Checksum Status: Good]
- Identifier (BE): 5098 (0x13ea)
- Identifier (LE): 59923 (0xea13)
- Sequence Number (BE): 32 (0x0020)
- Sequence Number (LE): 8192 (0x2000)
- Originate Timestamp: 3134839749 (5 days, 18 hours, 30 minutes, 53.626 seconds after midnight UTC)
- Receive Timestamp: 0 (0 seconds after midnight UTC)
- Transmit Timestamp: 0 (0 seconds after midnight UTC)

The packet bytes pane on the right shows the raw data of the packet, with the timestamp field (0x1aab) highlighted in blue.

Figura 6: Verificación del valor del timestamp.

Analizando la imagen anterior, observamos que nos encontramos en el paquete con secuencia 32 (destacado en celeste arriba), siendo este el penúltimo carácter transmitido, que corresponde a una 'n' (destacado en rojo a la derecha). Se puede ver el timestamp con un valor de 3134830749 (en celeste abajo a la izquierda), que es el mismo valor que en el último paquete, como se muestra en la siguiente imagen:

```

66 9.349342018 10.0.2.15 8.8.8.8 ICMP 103 Timestamp request id=0x13ea, seq=30/7680, ttl=64
67 9.474610993 10.0.2.15 8.8.8.8 ICMP 103 Timestamp request id=0x13ea, seq=31/7936, ttl=64
68 9.656375664 10.0.2.15 8.8.8.8 ICMP 103 Timestamp request id=0x13ea, seq=32/8192, ttl=64
69 9.889429261 10.0.2.15 8.8.8.8 ICMP 103 Timestamp request id=0x13ea, seq=33/8448, ttl=64

Frame 69: 103 bytes on wire (824 bits), 103 bytes captured (824 bits) on interface enp0s3, id 0
Ethernet II, Src: PCSSystemec_fe:59:95 (08:00:27:fe:59:95), Dst: 52:54:00:12:35:02 (52:54:00:12:35:02)
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 8.8.8.8
Internet Control Message Protocol
  Type: 13 (Timestamp request)
  Code: 0
  Checksum: 0x26aa [correct]
  [Checksum Status: Good]
  Identifier (BE): 5098 (0x13ea)
  Identifier (LE): 59923 (0xea13)
  Sequence Number (BE): 32 (0x0020)
  Sequence Number (LE): 8448 (0x2100)
  Originate Timestamp: 3134830749 (5 days, 18 hours, 30 minutes, 53.626 seconds after midnight UTC)
  Receive Timestamp: 0 (0 seconds after midnight UTC)
  Transmit Timestamp: 0 (0 seconds after midnight UTC)

```

Figura 7: Verificación del valor del timestamp.

Por otra parte, se solicitó que mantuviera una identificación, un ID y un número de secuencia coherente; para esto, se utilizó ChatGPT de la siguiente forma:

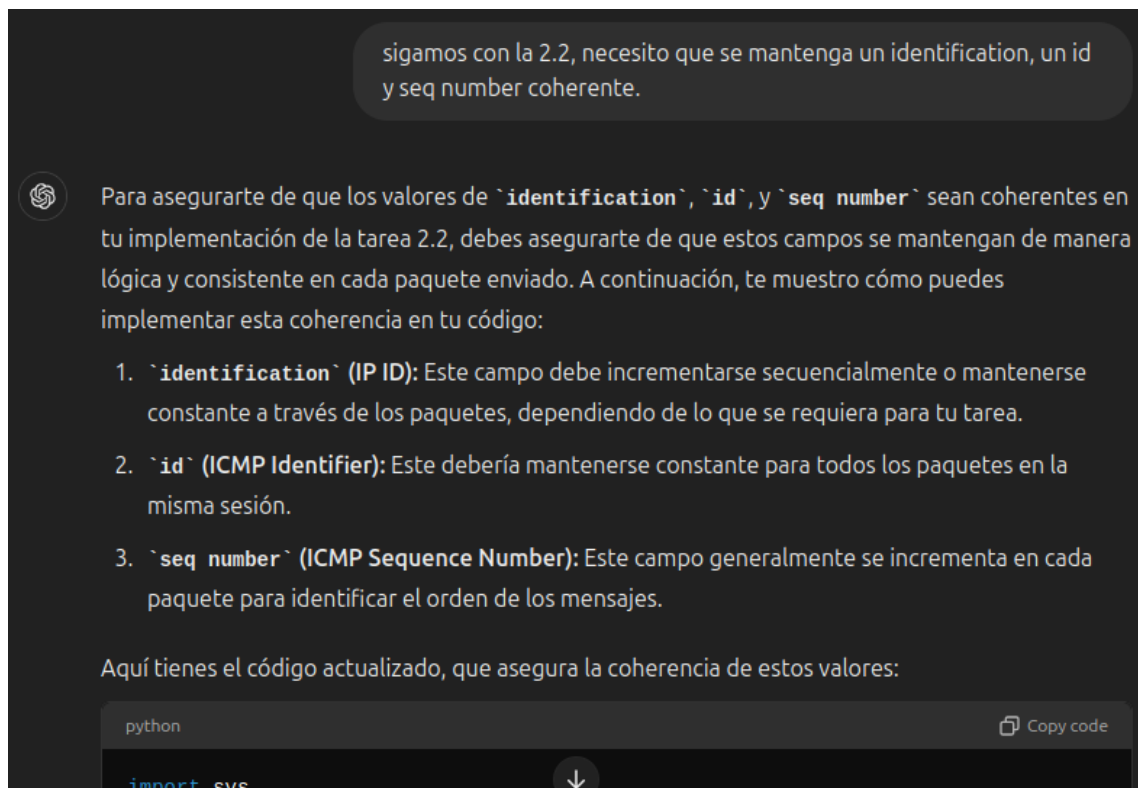


Figura 8: Prompt para mejorar el código de la segunda actividad.

Como último retoque del código, se le pidió a ChatGPT que mantuviera un orden en el

payload, con los primeros 8 bytes estaticos, el siguiente dedicado a la transmisión del caracter cifrado y los siguientes desde 0x10 a 0x37.

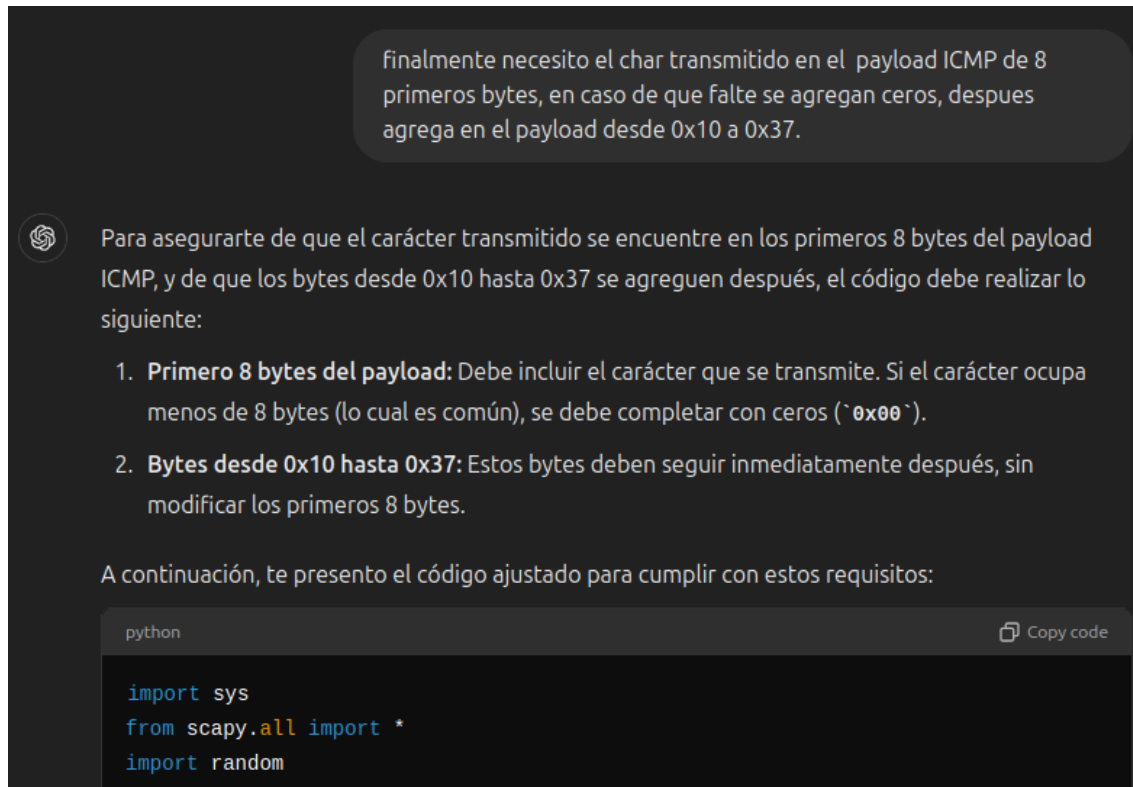


Figura 9: Prompt para mejorar el código de la segunda actividad.

El código y sus mejoras se encuentran en el archivo 'pingv4.py' en el repositorio, el cual, gracias a ChatGPT, está comentado indicando las partes clave de su funcionamiento.

Finalmente, se realiza un ping previo y posterior (hacia google.com) al generado por Python. Los comandos utilizados son:

```
> ping google.com
> sudo python3 pingv4.py "larycxpajorj h bnpdarmjm nw anmnb"
> ping google.com
```

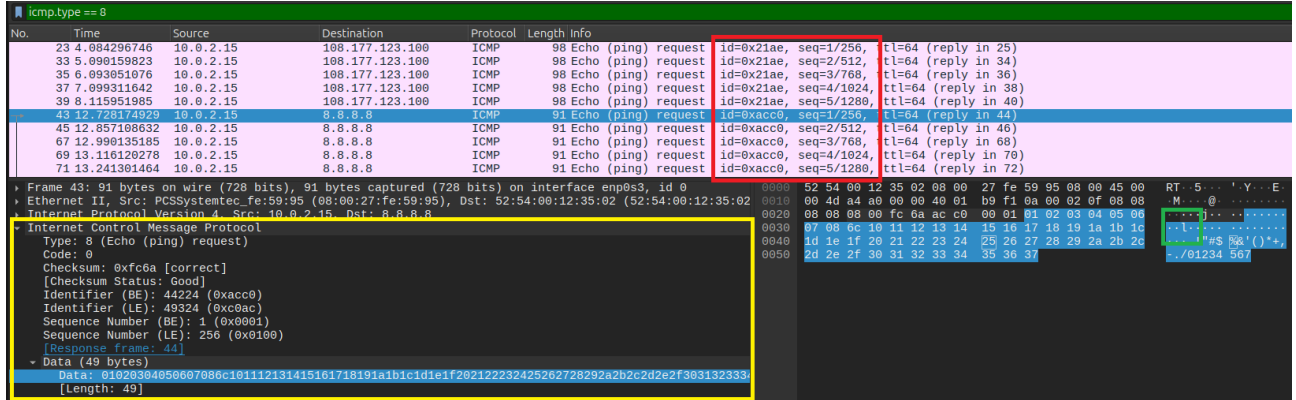


Figura 10: 5 pings de Google seguidos de 5 pings de Python.

Al analizar la imagen anterior, se observa que en el bloque rojo el ID permanece constante y el número secuencial se implementa correctamente. En el recuadro amarillo, se muestra el tipo de paquete (8), su identificador y número de secuencia de manera más detallada, junto con la data del paquete. Esta data comienza con 8 bytes estáticos, seguidos por el carácter cifrado (en este caso, '6c'), y continúa hasta el valor '0x10', finalizando en '0x37'. Finalmente, en el recuadro verde se presenta una representación gráfica de la data, donde se puede ver el primer carácter enviado, que es 'l'.

Es importante mencionar que los pings de Python pueden enviarse a la IP local (127.0.0.1) o a una IP como los DNS de Google; esta última es la utilizada y sirve para que el mensaje sea respondido.

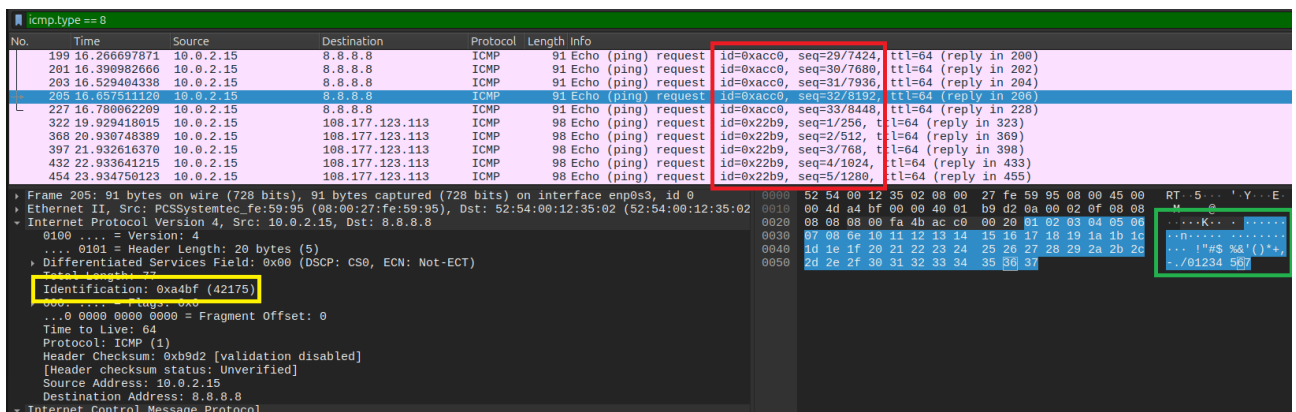


Figura 11: 5 pings desde Python seguidos de 5 pings hacia Google.

En el recuadro rojo se puede observar el correcto funcionamiento tanto del ID como del número secuencial, que en este caso es 32. En el recuadro verde (resaltado en celeste a la derecha) se identifica la letra 'n', que corresponde al penúltimo carácter enviado. A la izquierda, se presenta un ejemplo del campo 'identification', el cual, como se verá en la siguiente imagen, va variando.

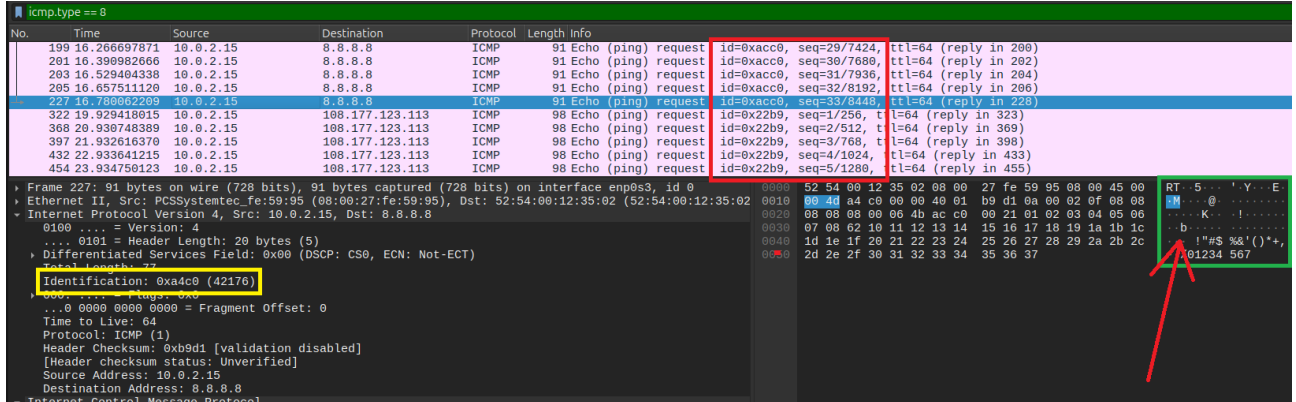


Figura 12: 5 pings desde Python seguidos de 5 pings hacia Google.

Finalmente, en el recuadro rojo se muestra el último paquete enviado desde Python, donde la flecha roja indica el último carácter transmitido, que es una 'b'. A la izquierda, en el recuadro amarillo, se observa un identificador distinto. El comportamiento de las diferentes variables fue replicado al realizar pings hacia Google, donde se comprobó que el campo 'identification' variaba, mientras que el ID permanecía estático y el número secuencial aumentaba de uno en uno, comenzando desde 1.

3.3. Actividad 3

En esta sección, se solicitó a ChatGPT que generara un programa en Python 3 capaz de recuperar el mensaje transmitido en el paso 2.2. Para simplificar el proceso, se utilizó una captura limpia que solo contiene los pings o paquetes ICMP relacionados con el mensaje transmitido de tipo 13, esto demuestra que aunque no se vea el data en este tipo si es ocupable su informacion. Dado que el desplazamiento del cifrado no se conoce de antemano, el programa debe generar todas las combinaciones posibles y mostrar el texto descifrado, destacando en verde la opción más probable de ser el mensaje original. CORREGIR

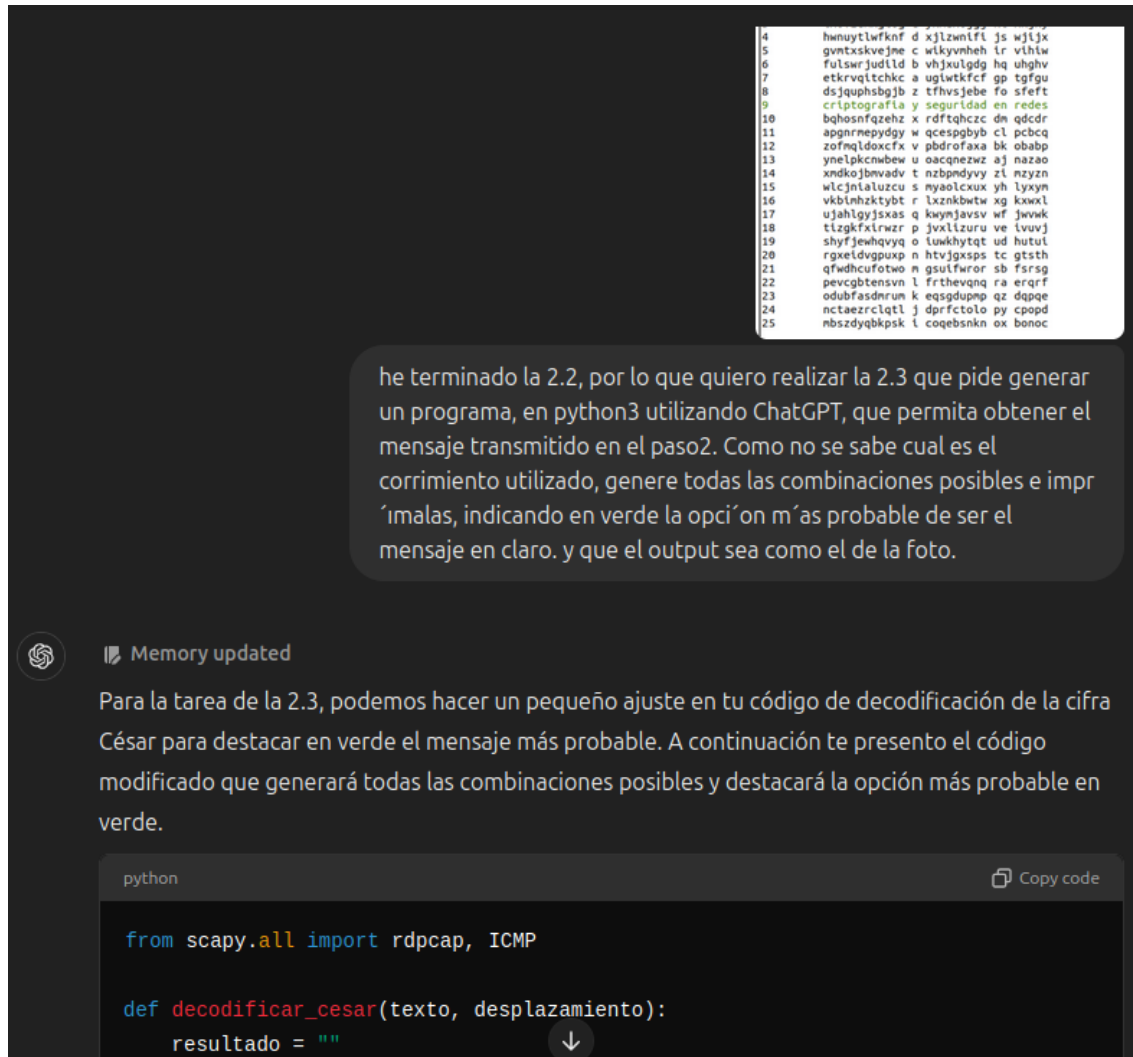


Figura 13: Prompt del comienzo de la actividad 3.

El primer código no funcionó como se esperaba, como se observa en la siguiente imagen:

```

rena@rena:~/Downloads$ sudo python3 readv2.py cesar.pcapng
Desplazamiento 1: k1234567z1234567q1234567x1234567b1234567w1234567o1234567z1234567i
1234567n1234567q1234567i1234567 1234567g1234567 1234567a1234567m1234567o1234567c
1234567z1234567q1234567l1234567i1234567l1234567 1234567m1234567v1234567 1234567z1
234567m1234567l1234567m1234567a1234567
Desplazamiento 2: j1234567y1234567p1234567w1234567a1234567v1234567n1234567y1234567h
1234567m1234567p1234567h1234567 1234567f1234567 1234567z1234567l1234567n1234567b
1234567y1234567p1234567k1234567h1234567k1234567 1234567l1234567u1234567 1234567y1
234567l1234567k1234567l1234567z1234567
Desplazamiento 3: i1234567x1234567o1234567v1234567z1234567u1234567m1234567x1234567g
1234567l1234567o1234567g1234567 1234567e1234567 1234567y1234567k1234567m1234567a
1234567x1234567o1234567j1234567g1234567j1234567 1234567k1234567t1234567 1234567x1
234567k1234567j1234567k1234567y1234567
Desplazamiento 4: h1234567w1234567n1234567u1234567y1234567t1234567l1234567w1234567f
1234567k1234567n1234567f1234567 1234567d1234567 1234567x1234567j1234567l1234567z
1234567w1234567n1234567i1234567f1234567i1234567 1234567j1234567s1234567 1234567w1

```

Figura 14: Output del primer código de la actividad 3.

Tras solicitar una forma más compleja y con probabilidades, se observó que era demasiado extenso y no relacionado con el curso, por lo que se usó el siguiente prompt, con el que se obtuvo el código casi final:

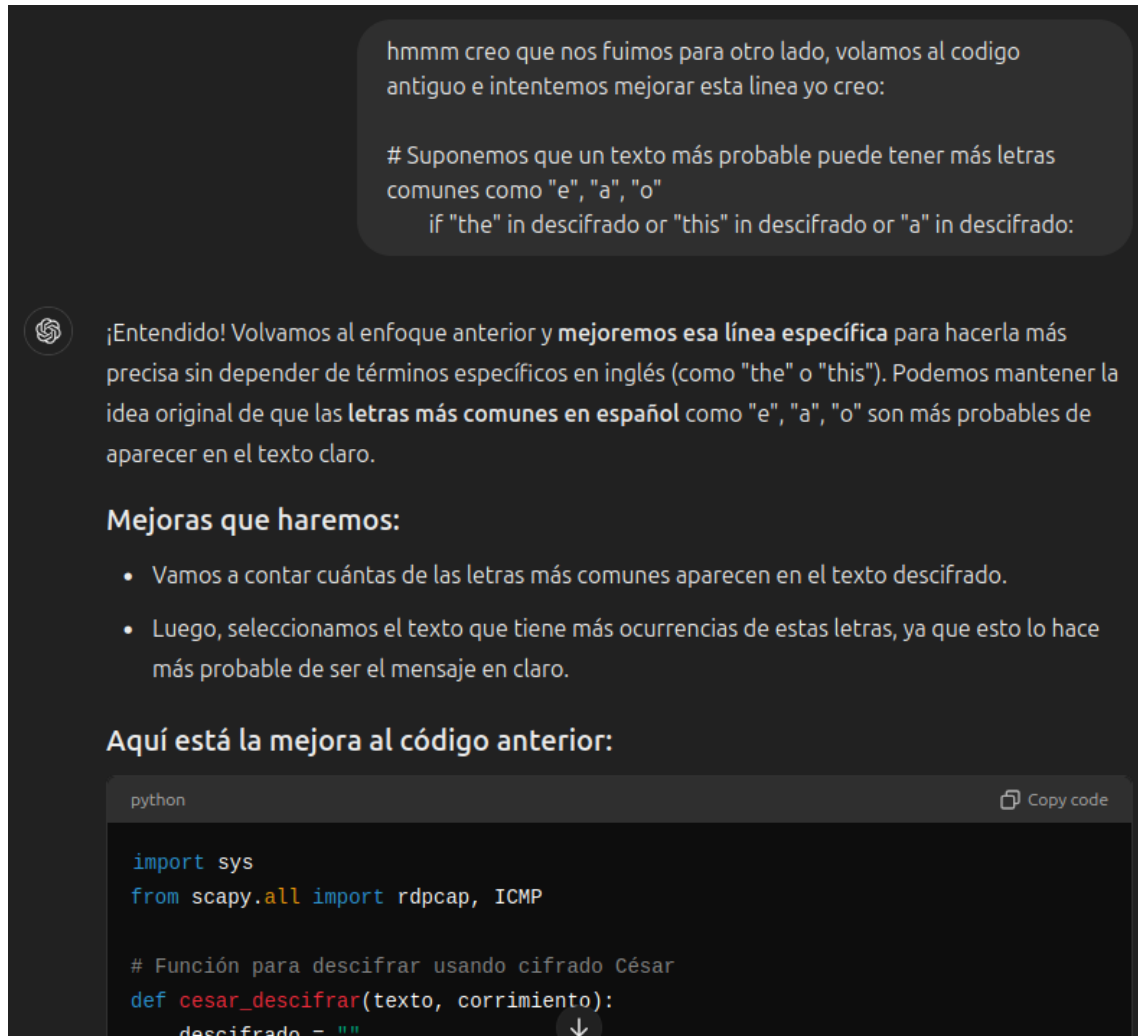


Figura 15: Prompt que termina la actividad 3.

El código fue ajustado para corregir la salida, lo que resultó en lo siguiente:


```
rena@rena:~/Downloads$ sudo python3 readv2.py cesar.pcapng
Datos ICMP extraídos: larycxpajorj h bnpdarmjm nw anmnb
Corrimiento 0: larycxpajorj h bnpdarmjm nw anmnb
Corrimiento 1: kzqxbwozinqi g amoczqlil mv zmlma
Corrimiento 2: jypwavnyhmpf f zlnbypkhk lu yklz
Corrimiento 3: ixovzumxglog e ykmaxojgj kt xkjky
Corrimiento 4: hwnuytlwfknf d xjlzwnifi js wjijx
Corrimiento 5: gvmtxskvejme c wikyvmheh ir vihiw
Corrimiento 6: fulswrjudild b vhjxulgdg hq uhghv
Corrimiento 7: etkrvqitchkc a ugiwtkfcf gp tgfgu
Corrimiento 8: dsjquphsbgjb z tfhvsjebe fo sfeft
Corrimiento 9: criptografia y seguridad en redes
Corrimiento 10: bqhosnfqzehz x rdftqhczc dm qdcdr
Corrimiento 11: apgnrmepdygy w qcespgbyb cl pcbcq
Corrimiento 12: zofmqldoxcfx v pbdrofaxa bk obabp
Corrimiento 13: ynelpkcnwbew u oacqnezww aj nazao
Corrimiento 14: xmdkojbmadv t nzbpmdivy zi mzyzn
Corrimiento 15: wlcjnia luzcu s myaolcxux yh lyxym
Corrimiento 16: vkbimhzktybt r lxznkbwtw xg kxwxl
Corrimiento 17: ujahlgysxas q kwymjavsv wf jvwvk
Corrimiento 18: tizgkfxirwzr p jvxlizuru ve ivuvj
Corrimiento 19: shyfjewhqvq o iuwkhytqt ud hutui
Corrimiento 20: rgxeidvgpuxp n htvjgxspc tc gtsth
Corrimiento 21: qfwdhcuftwo m gsuifwrwr sb fsrsg
Corrimiento 22: pevcbtensvn l frthevqnq ra erqrf
Corrimiento 23: odubfasdmrum k eqsgdupmp qz dqpqe
Corrimiento 24: nctaezrclqtl j dprfctolo py cpopd
Corrimiento 25: mbszdyqbkpsk i coqebnskn ox bonoc
rena@rena:~/Downloads$
```

Figura 16: Output final de la actividad 3.

El código comienza extrayendo los paquetes ICMP del archivo .pcapng y obteniendo los primeros 8 bytes del payload de cada paquete, que contienen los caracteres cifrados. Luego, se calculan todos los posibles corrimientos del cifrado César (del 0 al 25). Para cada corrimiento, se genera una versión descifrada del mensaje. Finalmente, con el objetivo de identificar el corrimiento más probable, se basa en la frecuencia de las letras más comunes en español, como e, a, o, s, entre otras, asumiendo que el texto descifrado debe contener estas letras con mayor frecuencia si está correctamente descifrado. El corrimiento que produce un mensaje con más de estas letras comunes es seleccionado como el más probable.

Conclusiones y comentarios

En este proyecto, uno de los principales desafíos fue asegurar que el tráfico de red generado no levantara sospechas en los sistemas de seguridad, lo que implicaba una cuidadosa gestión de los distintos parámetros de cada paquete. Aunque ChatGPT proporcionó un punto de partida útil, fue imprescindible ajustar el código generado para cumplir con los requisitos específicos. El uso de ChatGPT demostró ser una herramienta valiosa, aunque también expuso sus limitaciones. Si bien la IA facilitó ciertas tareas y permitió un ahorro de tiempo, fue necesario dedicar esfuerzos adicionales para ajustar y perfeccionar el código, garantizando así su funcionalidad. La experiencia adquirida en este laboratorio me enseñó que, aunque las herramientas de IA pueden acelerar el desarrollo de software, es fundamental contar con un conocimiento profundo y una supervisión humana constante para optimizar y asegurar la calidad del producto final.

Finalmente, cuatro problemas o situaciones con los que he tenido que lidiar al trabajar con ChatGPT son:

1. **Contabilización de preguntas:** Tuve que realizar múltiples cuentas para determinar la cantidad total de preguntas formuladas, ya que ChatGPT tiene un cupo diario de preguntas.
2. **Desarrollo de la sección 2.2:** En varias ocasiones, para que la IA realizara correctamente lo que se le pedía, era necesario proporcionar instrucciones muy específicas. Por ejemplo, al dar todos los requerimientos de la pauta de golpe, el sistema no respondía de manera adecuada.
3. **Especificidad en la sección 2.3:** Cuando no especificaba claramente cómo deseaba buscar la palabra más probable, ChatGPT adoptaba un enfoque muy creativo y hacía predicciones complejas basadas en probabilidades. Aunque esto no es negativo, a menudo resultaba en respuestas excesivamente complejas, requiriendo aclaraciones adicionales para simplificar la respuesta.
4. **Confusión entre términos:** Hubo casos en los que tuve que repetir que, por ejemplo, 'identification' e 'id' eran distintos, y que uno era una variable y el otro no. Al iterar varias veces en un código, ChatGPT a veces volvía a esos 'patrones' antiguos.