

# Informe Laboratorio 1

## Sección 3

Alumno: Pablo Castro  
e-mail: pablo.castro<sub>d</sub>@mail.udp.cl

4 de Septiembre de 2023

## 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 . . . . .	5
3.3. Actividad 3 . . . . .	10

## 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.

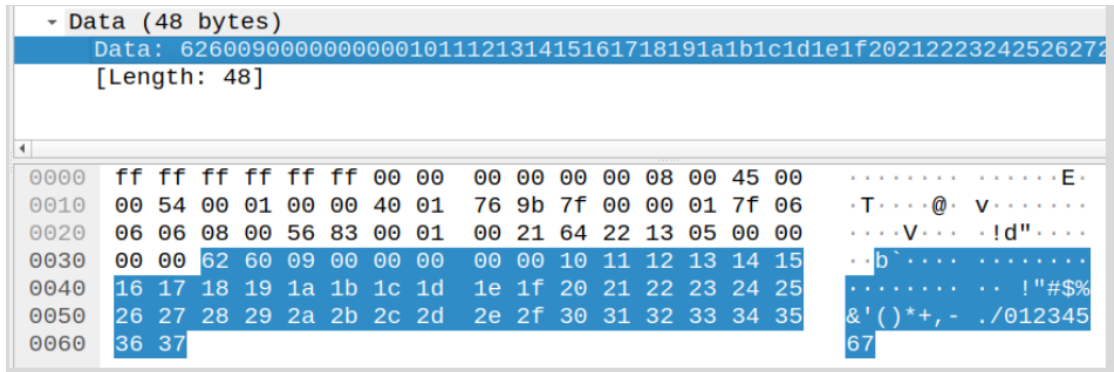
```
└─$ ~/Desktop $ sudo python3 cesar.py "criptografia y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb
```

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

```

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

### 3. Desarrollo de Actividades

#### 3.1. Actividad 1

En esta actividad, se busca implementar un código en python, a través de ChatGPT, transforme un string ingresado por la línea de comando en un cifrado cesar, con un corrimiento también ingresado en la línea de comandos. Sin embargo, se debe tomar en cuenta que el idioma que usaremos es el español, por ende, se tiene que explicitar que se usará el abecedario español, junto a la "ñ". También las mayúsculas se usarán como minúsculas. Para esto se le entrega el siguiente prompt a ChatGPT e inmediatamente da una respuesta:

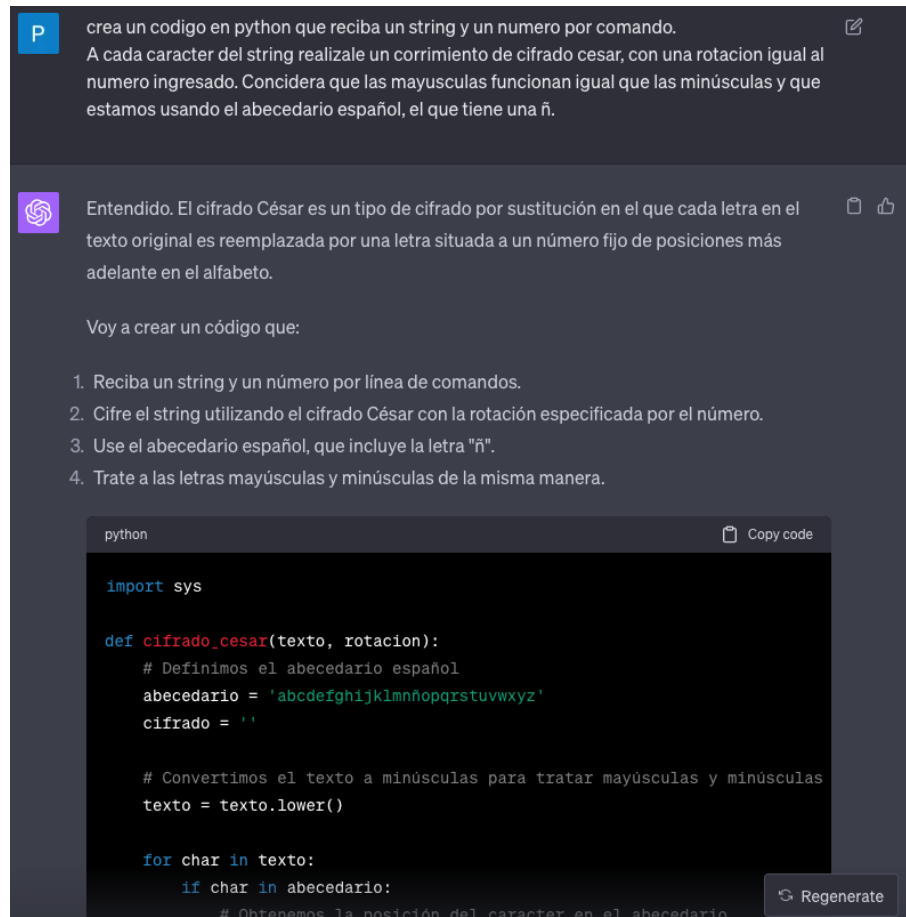


Figura 1: Prompt cifrado cesar, parte 1

```

# Definimos el abecedario español
abecedario = 'abcdefghijklmnñopqrstuvwxyz'
cifrado = ''

# Convertimos el texto a minúsculas para tratar mayúsculas y minúsculas
texto = texto.lower()

for char in texto:
    if char in abecedario:
        # Obtenemos la posición del caracter en el abecedario
        pos_original = abecedario.index(char)

        # Calculamos la nueva posición con la rotación
        nueva_pos = (pos_original + rotacion) % len(abecedario)

        # Añadimos el nuevo caracter al texto cifrado
        cifrado += abecedario[nueva_pos]
    else:
        # Si el caracter no está en el abecedario, lo añadimos sin cambi
        cifrado += char

return cifrado

if __name__ == '__main__':
    # Obtener el texto y la rotación desde los argumentos de línea de comand
    texto = sys.argv[1]
    rotacion = int(sys.argv[2])

    resultado = cifrado_cesar(texto, rotacion)
    print(resultado)

```

Para usar este código, guárdalo en un archivo, por ejemplo `cifrado_cesar.py`, y ejecútalo desde la línea de comandos de la siguiente manera:

Figura 2: Prompt cifrado cesar, parte 2

El funcionamiento del código coincide en ejecutar el archivo desde la línea de comandos adjuntando la palabra y el corrimiento, para que nos retorne la palabra con el cifrado cesar.

```

mimac@MacBook-Air-de-Mi 1.Cifrado % python3 cesar.py "Tengo mucho sueño y mucha hambre" 8
bmuñw tckow acmvw g tckoi oitjzm

```

Figura 3: Cifrado Cesar, final

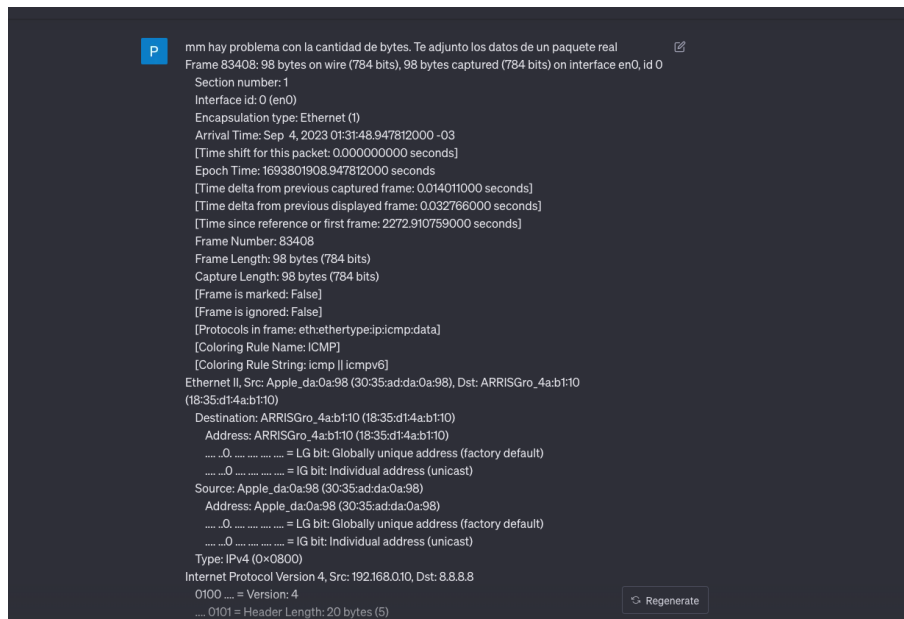
Se demuestra que la actividad de cifrado funciona correctamente.

### 3.2. Actividad 2

En esta actividad, se nos pide realizar tráfico ICMP, similar al ping, pero escondiendo en cada paquete ICMP, un carácter en el último byte del payload. Con el fin de pasar

desapercibido. Para lograr esto, se intentaron una enorme cantidad de prompts, por lo que en este informe se mostrarán las partes más importantes. ChatGPT no ubicaba bien los bits del payload y los campos de los paquetes, provocando así una distorsión en el paquete lo que hacía que llamara mucho la atención. Sin embargo, el prompt que hizo un verdadero cambio, fue cuando se le adjuntaron los campos reales de un paquete ping -c 1 8.8.8.8.

Nota: En esta actividad se intentó inyectar los caracteres del string "hola".



The screenshot shows a chat window with a dark background. On the left, there is a blue square icon with a white letter 'P'. To its right is a prompt: "mm hay problema con la cantidad de bytes. Te adjunto los datos de un paquete real". Below the prompt is a detailed network packet analysis output. The output includes fields such as "Frame 83408: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface en0, id 0", "Section number: 1", "Interface id: 0 (en0)", "Encapsulation type: Ethernet (1)", "Arrival Time: Sep 4, 2023 01:31:48.947812000 -03", "[Time shift for this packet: 0.000000000 seconds]", "Epoch Time: 1693801908.947812000 seconds", "[Time delta from previous captured frame: 0.014011000 seconds]", "[Time delta from previous displayed frame: 0.032766000 seconds]", "[Time since reference or first frame: 2272.910739000 seconds]", "Frame Number: 83408", "Frame Length: 98 bytes (784 bits)", "Capture Length: 98 bytes (784 bits)", "[Frame is marked: False]", "[Frame is ignored: False]", "[Protocols in frame: eth:ethertype:ip:icmp:data]", "[Coloring Rule Name: ICMP]", "[Coloring Rule String: icmp || icmpv6]", "Ethernet II, Src: Apple\_da:0a:98:30:35:ad:da:0a:98, Dst: ARRISGro\_4a:b1:10:18:35:d1:4a:b1:10", "Destination: ARRISGro\_4a:b1:10:18:35:d1:4a:b1:10", "Address: ARRISGro\_4a:b1:10:18:35:d1:4a:b1:10", "... 0 ... = LG bit: Globally unique address (factory default)", "... 0 ... = IG bit: Individual address (unicast)", "Source: Apple\_da:0a:98:30:35:ad:da:0a:98", "Address: Apple\_da:0a:98:30:35:ad:da:0a:98", "... 0 ... = LG bit: Globally unique address (factory default)", "... 0 ... = IG bit: Individual address (unicast)", "Type: IPv4 (0x0800)", "Internet Protocol Version 4, Src: 192.168.0.10, Dst: 8.8.8.8", "0100 ... = Version: 4", "... 0101 = Header Length: 20 bytes (5)". At the bottom right of the chat window, there is a button labeled "Regenerate".

Figura 4: Prompt Stealth ICMP, parte 12

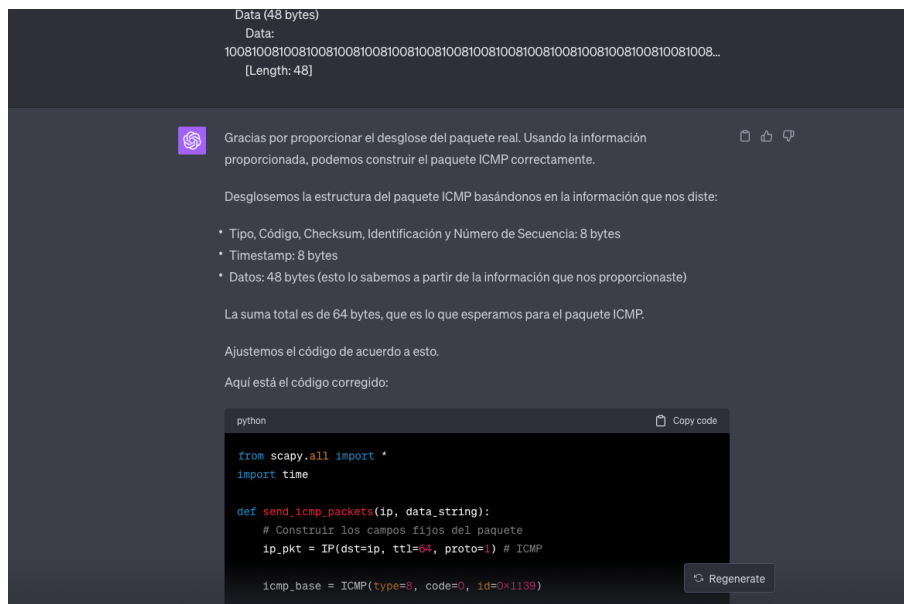


Figura 5: Prompt Stealth ICMP, parte 15

A lo que ChatGPT responde:

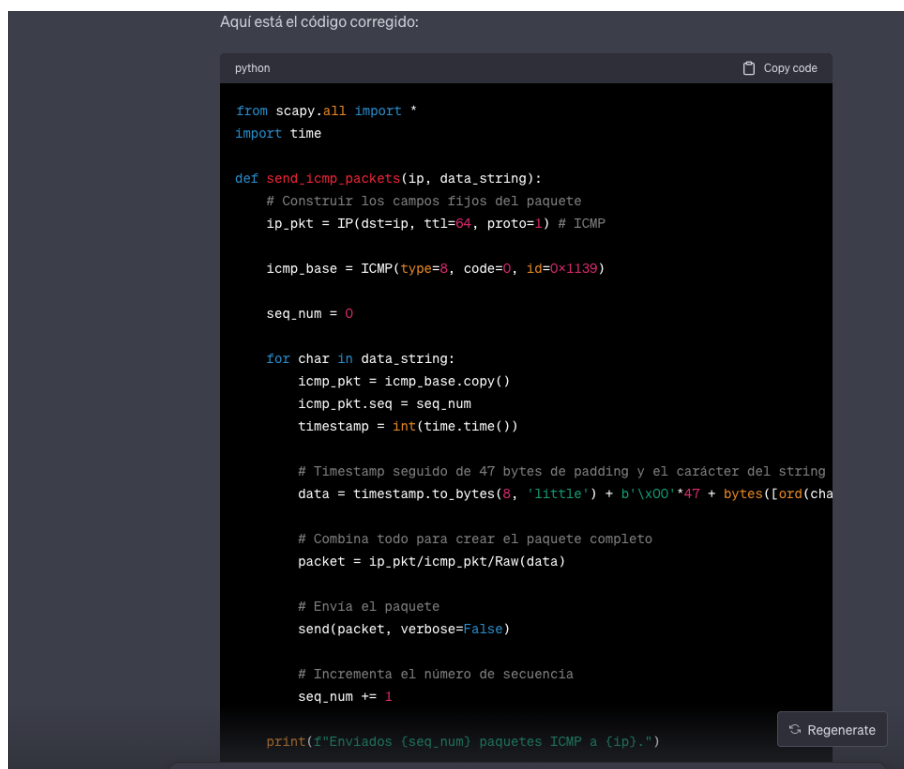


Figura 6: Prompt cifrado cesar, parte 16

Paquete que sirvió de referencia:

78843	1955.699767	192.168.0.10	8.8.8.8	ICMP	98 Echo (ping) request	id=0xc3f8, seq=1/256, ttl=64 (reply in 78844)
78856	1956.703947	192.168.0.10	8.8.8.8	ICMP	98 Echo (ping) request	id=0xc3f8, seq=2/512, ttl=64 (reply in 78857)
78867	1957.708869	192.168.0.10	8.8.8.8	ICMP	98 Echo (ping) request	id=0xc3f8, seq=3/768, ttl=64 (reply in 78868)
78871	1957.758274	192.168.0.10	8.8.8.8	ICMP	54 Echo (ping) request	id=0x0000, seq=0/0, ttl=64 (reply in 78873)
78872	1957.769588	192.168.0.10	8.8.8.8	ICMP	54 Echo (ping) request	id=0x0000, seq=0/0, ttl=64 (reply in 78874)
78875	1957.790872	192.168.0.10	8.8.8.8	ICMP	54 Echo (ping) request	id=0x0000, seq=0/0, ttl=64 (reply in 78876)
78877	1957.818392	192.168.0.10	8.8.8.8	ICMP	54 Echo (ping) request	id=0x0000, seq=0/0, ttl=64 (reply in 78878)
83482	2272.844753	192.168.0.10	8.8.8.8	ICMP	98 Echo (ping) request	id=0x8f39, seq=0/0, ttl=64 (reply in 83483)
83485	2272.877993	192.168.0.10	8.8.8.8	ICMP	98 Echo (ping) request	id=0x1039, seq=0/0, ttl=64 (reply in 83487)
83488	2272.910759	192.168.0.10	8.8.8.8	ICMP	98 Echo (ping) request	id=0x1139, seq=0/0, ttl=64 (reply in 83489)
83418	2272.938329	192.168.0.10	8.8.8.8	ICMP	98 Echo (ping) request	id=0x1239, seq=0/0, ttl=64 (reply in 83411)
85875	2458.473494	192.168.0.10	8.8.8.8	ICMP	42 Echo (ping) request	id=0x2d39, seq=0/0, ttl=64 (reply in 85876)
85885	2459.502588	192.168.0.10	8.8.8.8	ICMP	43 Echo (ping) request	id=0x3139, seq=0/0, ttl=64 (reply in 85886)
85893	2460.564066	192.168.0.10	8.8.8.8	ICMP	44 Echo (ping) request	id=0x3239, seq=0/0, ttl=64 (reply in 85894)
85899	2461.593360	192.168.0.10	8.8.8.8	ICMP	45 Echo (ping) request	id=0x3339, seq=0/0, ttl=64 (reply in 85901)
143998	3568.015107	192.168.0.10	8.8.8.8	ICMP	89 Echo (ping) request	id=0x1234, seq=0/0, ttl=64 (reply in 144000)

Figura 7: Paquete de referencia para generar el tráfico

Paquetes generados por el comando ping desde la terminal:

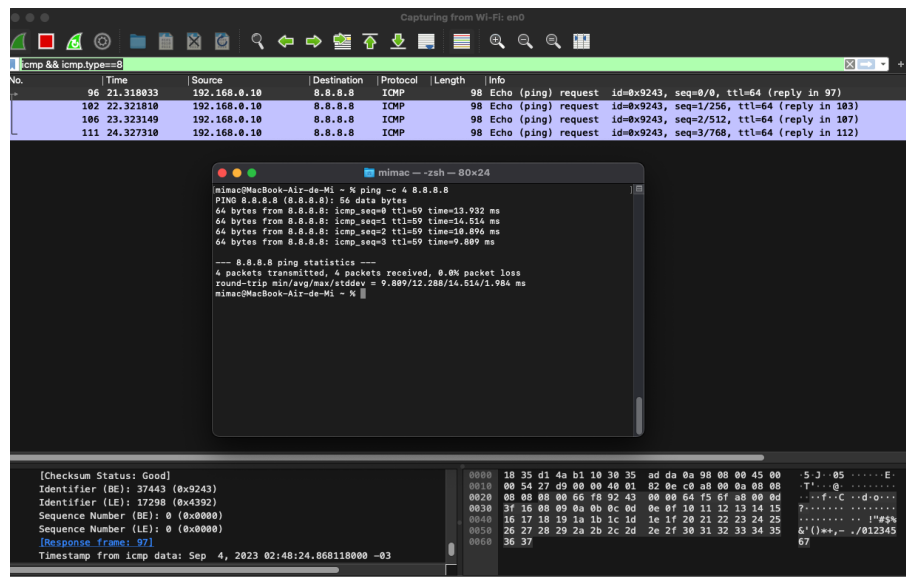


Figura 8: Paquetes generados por ping desde terminal

Paquetes generados por el código y por el ping desde la terminal:

No.	Time	Source	Destination	Protocol	Length	Info
96	21.318033	192.168.0.10	8.8.8.8	ICMP	98	Echo (ping) request id=0x9243, seq=0/0, ttl=64 (reply in 97)
182	22.321818	192.168.0.10	8.8.8.8	ICMP	98	Echo (ping) request id=0x9243, seq=1/256, ttl=64 (reply in 183)
186	23.323149	192.168.0.10	8.8.8.8	ICMP	98	Echo (ping) request id=0x9243, seq=2/512, ttl=64 (reply in 187)
111	24.327310	192.168.0.10	8.8.8.8	ICMP	98	Echo (ping) request id=0x9243, seq=3/768, ttl=64 (reply in 112)
627	70.574603	192.168.0.10	8.8.8.8	ICMP	98	Echo (ping) request id=0x1139, seq=0/0, ttl=64 (reply in 629)
628	70.584017	192.168.0.10	8.8.8.8	ICMP	98	Echo (ping) request id=0x1139, seq=1/256, ttl=64 (reply in 631)
630	70.594798	192.168.0.10	8.8.8.8	ICMP	98	Echo (ping) request id=0x1139, seq=2/512, ttl=64 (reply in 632)
633	70.604588	192.168.0.10	8.8.8.8	ICMP	98	Echo (ping) request id=0x1139, seq=3/768, ttl=64 (reply in 634)

Figura 9: Paquetes generados, con ChatGPT

Como se puede ver, los paquetes modificados pasan desapercibidos a primera vista, ya que al mantener un seq incremental y un mismo identifi er, parecerá que son de un mismo



proceso. Tal como sucede con el comando ping -c 5 8.8.8.8. También tienen un ttl correcto, un timestamp y un lenght de payload, de todos los campos correctos.

Comparación primer paquete ICMP generado por consola v/s generado por chatGPT:

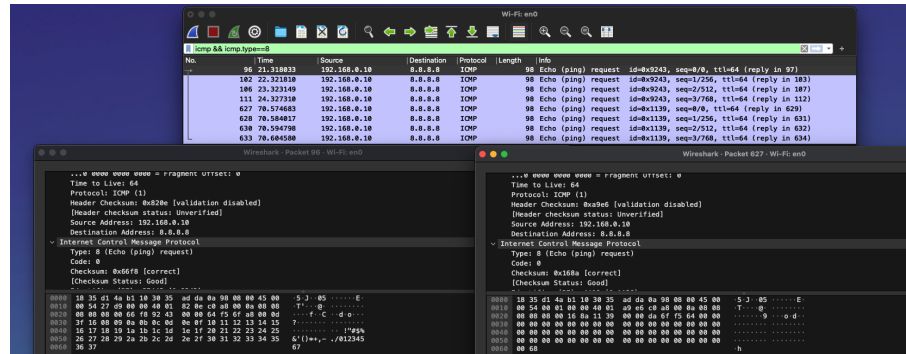


Figura 10: Comparación de paquetes, parte 1

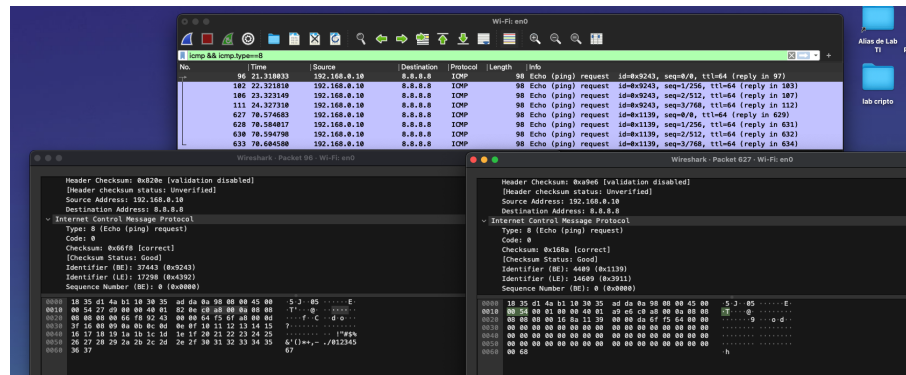


Figura 11: Comparación de paquetes, parte 2

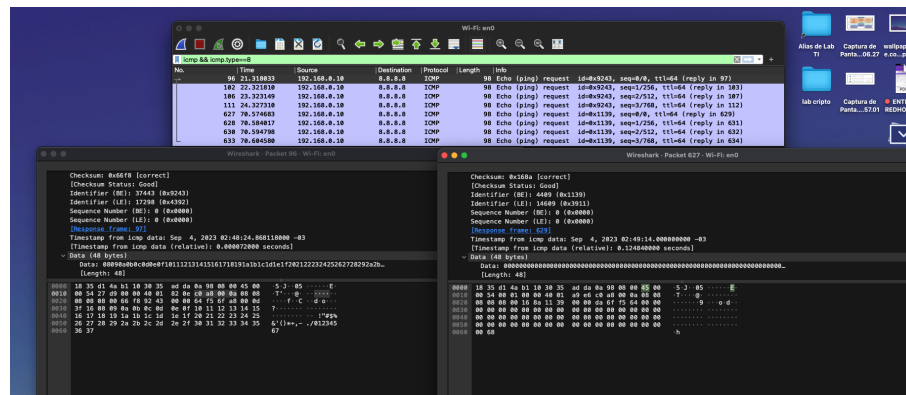


Figura 12: Comparación de paquetes, parte 3

Se demuestra que los paquetes son casi idénticos y que el caracter está en el último byte del payload.

### 3.3. Actividad 3

En la ultima actividad se pide capturar tráfico ICMP que deben corresponder a los enviados en la actividad 2. Capturar el carácter del ultimo byte de cada paquete y guardarlo en un string. Luego se analizan todos los posibles corrimientos cesar que se le pudieron ver aplicado, y con análisis de idioma (Frecuencia de caracteres y palabras en español), se puede encontrar la palabra o frase más probable que haya sido, junto con el corrimiento aplicado. La palabra más probable debe destacarse en verde. Nuevamente, se le pide a Chatgpt que haga un código.

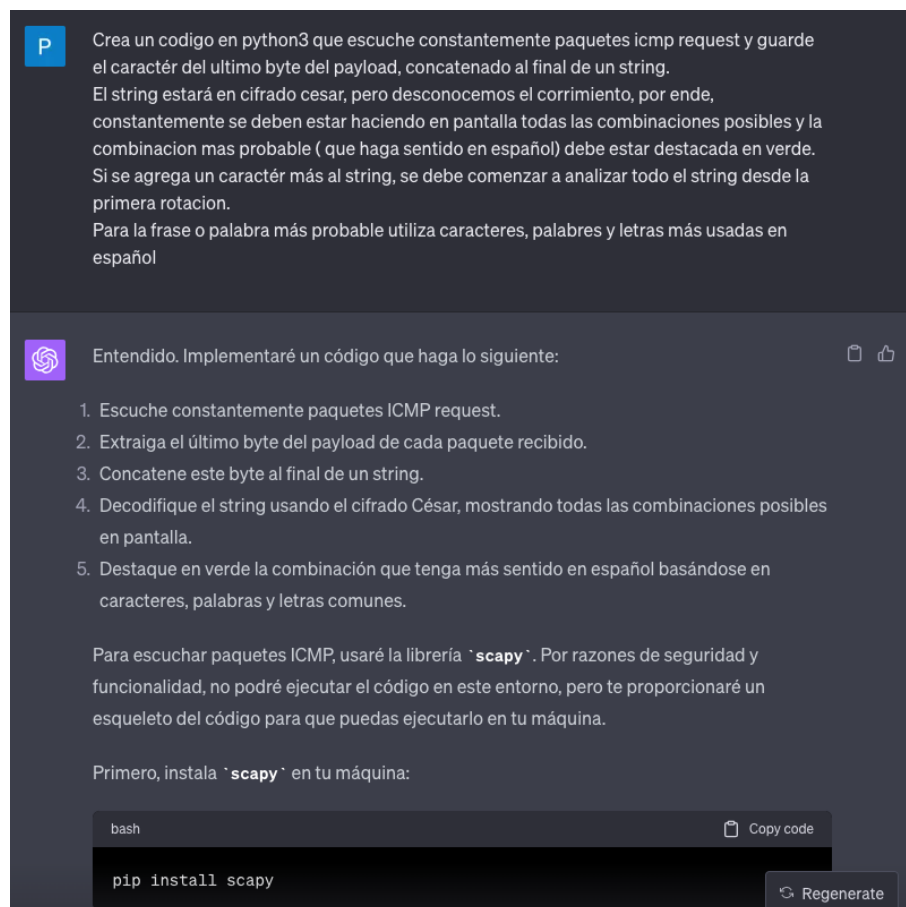


Figura 13: Prompt MitM, parte 1

```
pip install scapy
```

Luego, puedes usar el siguiente código:

```
python
```

```
import scapy.all as scapy
from collections import Counter
import string

# Diccionario de frecuencias de letras en español
freq_espanol = {
    'a': 12.53, 'b': 1.42, 'c': 4.68, 'd': 5.86, 'e': 13.68,
    'f': 0.69, 'g': 1.01, 'h': 0.70, 'i': 6.25, 'j': 0.44,
    'k': 0.02, 'l': 4.97, 'm': 3.15, 'n': 6.71, 'o': 8.68,
    'p': 2.51, 'q': 0.88, 'r': 6.87, 's': 7.98, 't': 4.63,
    'u': 3.93, 'v': 0.90, 'w': 0.01, 'x': 0.22, 'y': 0.90, 'z': 0.52
}

def cesar_cipher(text, shift):
    """Decodificar un texto usando el cifrado César con un corrimiento espec
    alphabet = string.ascii_lowercase
    table = str.maketrans(alphabet, alphabet[shift:] + alphabet[:shift])
    return text.translate(table)

def compute_score(text):
    """Calcular un score basado en las frecuencias de letras en español"""
    count = Counter(text)
    score = sum([freq_espanol[char] for char in text if char in freq_espanol])
    return score

def best_shift(text):
    """Determinar el mejor corrimiento para el cifrado César basado e
    best_score = -1
```

Copy code

Regenera

Figura 14: Prompt MitM, parte 2

```
best_score = -1
best_shift = -1
for shift in range(26):
    decoded = cesar_cipher(text, shift)
    score = compute_score(decoded)
    if score > best_score:
        best_score = score
        best_shift = shift
return best_shift

def packet_handler(pkt):
    """Manejador de paquetes para extraer el último byte del payload de un p
    global accumulated_string

    if pkt.haslayer(scapy.ICMP) and pkt[scapy.ICMP].type == 8: # ICMP reque
        payload = pkt[scapy.Raw].load
        last_byte = chr(payload[-1])
        accumulated_string += last_byte
        print_all_shifts(accumulated_string)

def print_all_shifts(text):
    """Imprimir todas las decodificaciones posibles, destacando la más proba
    probable_shift = best_shift(text)
    for shift in range(26):
        decoded = cesar_cipher(text, shift)
        if shift == probable_shift:
            print(f"\033[92mShift {shift}: {decoded}\033[0m") # verde
        else:
            print(f"Shift {shift}: {decoded}")

accumulated_string = ""

# Escuchar paquetes ICMP
scapy.sniff(filter="icmp", prn=packet_handler)
```

[Regenerate](#)

Figura 15: Prompt MitM, parte 3

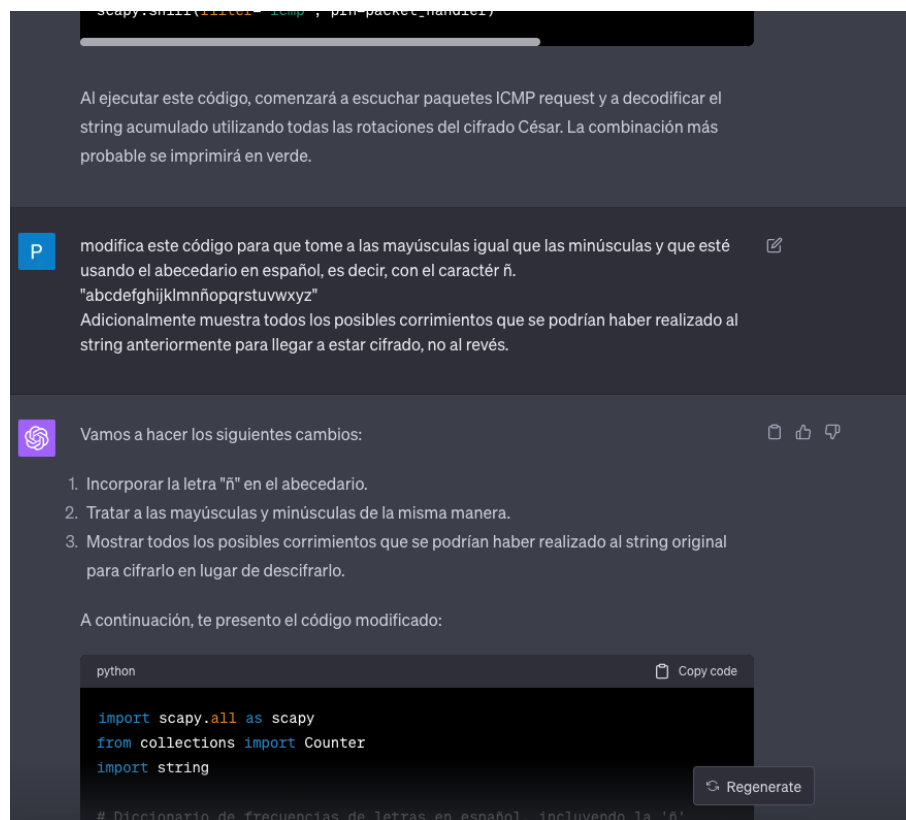


Figura 16: Prompt MitM, parte 4

Una vez está ejecutado el código, se ejecuta el código de la actividad 2. En este caso se enviará "Tengo mucho sueño y mucha hambre", string que al pasar por la primera actividad, con un corrimiento igual a 8, queda así: "bmuñw tckow acmvw g tckoi oitjzm".

Se pone a prueba el código:

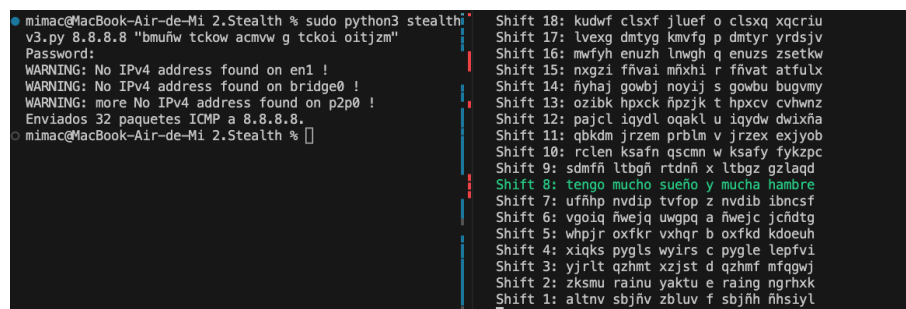


Figura 17: Funcionamiento MitM

## Conclusiones y comentarios

La actividad de laboratorio fué un éxito, se pudo cifrar en cesar, enviar paquetes personalizados para simular tráfico real, y se pudo atajar y predecir cual era el mensaje correcto. Trabajar con ChatGpt creo que fué lo más difícil de esta experiencia. El hacer que haga el prompt correcto, quita demasiado tiempo, quizás hubiera sido más fácil todo hacerlo manualmente. Mi primer issue fue en el cifrado cesar, que el chat no pensaba en un abecedario con ñ. El segundo fue al enviar los paquetes ICMP, ya que nunca ordenaba los bytes del paquete como yo quería y tampoco nunca añadía el timestamp. El tercero fue que muchas veces no respondía ni cerca de lo que quería, lo que me hacía perder mucho tiempo. El cuarto fue en casi todas las actividades, que me tiraba que no podía realizar la petición por afectar al orden y a la moral y me bloqueaba el input de texto.

Todos los códigos e imágenes están de manera ordenada en este enlace: Repositorio Github