

Informe Laboratorio 1

Sección 1

Cristóbal León

e-mail: cristobal.leon1@mail.udp.cl

29 de Marzo de 2024

Índice

1. Descripción	3
2. Actividades	3
2.1. Algoritmo de cifrado	3
2.2. Modo stealth	3
2.3. MitM	5
3. Desarrollo de Actividades	6
3.1. Actividad 1	6
3.2. Actividad 2	8
3.3. Actividad 3	18

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).

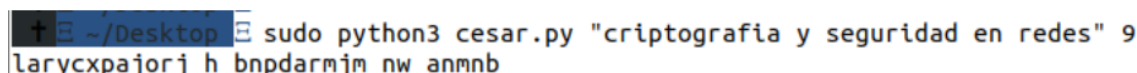
A usted le han encomendado auditar si efectivamente estos sistemas son capaces de detectar las filtraciones a través de tráfico de red. Debido a que el programa ping es ampliamente utilizado desde dentro y hacia fuera de la empresa, su tarea será crear un software que permita replicar tráfico generado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

```

$ sudo python3 readv2.py cesar.pcapng
0      larycxpajorj h bnpdarmjm nw anmnb
1      kzqxbwozinqi g amoczqlil mv zmlma
2      jypwavnyhmpf f zlnbypkhk lu yklklz
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 sfef
9      criptografia y seguridad en redes
10     bqhosnfqzehz x rdftqhczc dm qdcdr
11     apgnrmepdygy w qcespgbyb cl pcbbc
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 htvjgxsp s tc gtsth
21     qfwdhcufo two m gsuifwr or sb fsrsg
22     pevcbtensvn l frthevqnq ra erqrf
23     odubfasdmrum k eqsgdupmp qz dqpqe
24     nctaezrclqtl j dprfctolo py cpopd
25     mbszdyqbksk 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 primera instancia, se procede a desarrollar un script en Python3 que permite cifrar texto utilizando cesar.

Para esto, se le debe dar un input al script de la siguiente manera:

```
sudo python3 cesar.py "<palabra a cifrar>" <numero del corrimiento cesar>
```

Durante todas las actividades del presente laboratorio se usarán 2 ejemplos:

- “criptografía y seguridad en redes” rot-9
- “hola como estas” rot-5

Como se puede observar en las figuras 1 y 2, se han cifrado 2 textos con distintos corrimientos de César, entregando el mensaje correctamente cifrado.

```
[(base) admin@iMac-de-Cristobal Lab1 % sudo python3 cesar.py "criptografia y seguridad en redes" 9 ]  
larycxpajorj h bnpdarmjm nw anmnb  
(base) admin@iMac-de-Cristobal Lab1 % █
```

Figura 1

```
[(base) admin@iMac-de-Cristobal Lab1 % sudo python3 cesar.py "hola como estas" 5 ]  
mtqf htrt jxyfx  
(base) admin@iMac-de-Cristobal Lab1 % █
```

Figura 2

El código de la Figura 3 aplicado anteriormente, implementa un cifrado César. El programa toma dos argumentos de la línea de comandos: el texto a cifrar y el valor de corrimiento que indica cuántas posiciones se moverán las letras en el alfabeto. Luego, utiliza una función llamada `cifrar cesar` para realizar el cifrado. Esta función recorre cada carácter del texto, verificando si es una letra y luego aplicando el corrimiento apropiado, manteniendo el caso (mayúsculas o minúsculas) del carácter original. Finalmente, imprime el texto cifrado en la salida estándar.

```
import sys

def cifrar_cesar(texto, corrimiento):
    texto_cifrado = ''
    for caracter in texto:
        # Verificar si el caracter es una letra
        if caracter.isalpha():
            # Obtener el código ASCII del caracter
            codigo = ord(caracter)
            # Aplicar el corrimiento y ajustar según el rango de letras mayúsculas o minúsculas
            if caracter.islower():
                codigo_cifrado = (codigo - ord('a') + corrimiento) % 26 + ord('a')
            elif caracter.isupper():
                codigo_cifrado = (codigo - ord('A') + corrimiento) % 26 + ord('A')
            # Convertir el código ASCII cifrado de nuevo a caracter
            caracter_cifrado = chr(codigo_cifrado)
            # Agregar el caracter cifrado al texto cifrado
            texto_cifrado += caracter_cifrado
        else:
            # Si el caracter no es una letra, simplemente agregarlo sin cifrar
            texto_cifrado += caracter
    return texto_cifrado

if len(sys.argv) != 3:
    print("Uso: python3 cifrado_cesar.py <texto_a_cifrar> <corrimiento>")
    sys.exit(1)

# Obtener los argumentos de la línea de comandos
texto_a_cifrar = sys.argv[1]
corrimiento = int(sys.argv[2])

# Llamar a la función para cifrar el texto
texto_cifrado = cifrar_cesar(texto_a_cifrar, corrimiento)

# Mostrar el texto cifrado
print(texto_cifrado)
```

Figura 3

3.2. Actividad 2

Luego, utilizando Scapy se procede a enviar los strings cifrados del paso anterior dentro de un paquete ICMP, de manera tal, que este no parezca un paquete legítimo, se enviará un paquete ICMP para cada carácter del texto cifrado por César anteriormente. El tráfico descrito anteriormente será enviado a la IP de loopback “127.0.0.1”.

Para esto, se le debe dar un input al script de la siguiente manera:

```
sudo python3 pingv1.py "<texto cifrado en cesar anteriormente>"
```

Como se puede observar en las figuras 4 y 5, se han enviado n paquetes siendo n el largo de caracteres según el mensaje cifrado que se ha enviado, en este caso, se enviaron paquetes con el mensaje cifrado de “criptografia y seguridad en redes” rot-9 (larycxpajorj h bnpdarmjm nw anmnb)y de “hola como estas” rot-5(mtqf htrt jxyfx).


```
informatica@informatica-08:~/lab1/Actividad2$ sudo python3 pingv1.py "larycxpajor
j h bnpdarmjm nw anmnb"
Enviado: l - Timestamp: Mar 27, 2024 19:05:45.000 UTC
Enviado: a - Timestamp: Mar 27, 2024 19:05:46.000 UTC
Enviado: r - Timestamp: Mar 27, 2024 19:05:47.000 UTC
Enviado: y - Timestamp: Mar 27, 2024 19:05:48.000 UTC
Enviado: c - Timestamp: Mar 27, 2024 19:05:49.000 UTC
Enviado: x - Timestamp: Mar 27, 2024 19:05:51.000 UTC
Enviado: p - Timestamp: Mar 27, 2024 19:05:52.000 UTC
Enviado: a - Timestamp: Mar 27, 2024 19:05:53.000 UTC
Enviado: j - Timestamp: Mar 27, 2024 19:05:54.000 UTC
Enviado: o - Timestamp: Mar 27, 2024 19:05:55.000 UTC
Enviado: r - Timestamp: Mar 27, 2024 19:05:56.000 UTC
Enviado: j - Timestamp: Mar 27, 2024 19:05:57.000 UTC
Enviado: - Timestamp: Mar 27, 2024 19:05:58.000 UTC
Enviado: h - Timestamp: Mar 27, 2024 19:05:59.000 UTC
Enviado: - Timestamp: Mar 27, 2024 19:06:00.000 UTC
Enviado: b - Timestamp: Mar 27, 2024 19:06:01.000 UTC
Enviado: n - Timestamp: Mar 27, 2024 19:06:02.000 UTC
Enviado: p - Timestamp: Mar 27, 2024 19:06:03.000 UTC
Enviado: d - Timestamp: Mar 27, 2024 19:06:04.000 UTC
Enviado: a - Timestamp: Mar 27, 2024 19:06:05.000 UTC
Enviado: r - Timestamp: Mar 27, 2024 19:06:06.000 UTC
Enviado: m - Timestamp: Mar 27, 2024 19:06:08.000 UTC
Enviado: j - Timestamp: Mar 27, 2024 19:06:09.000 UTC
Enviado: m - Timestamp: Mar 27, 2024 19:06:10.000 UTC
Enviado: - Timestamp: Mar 27, 2024 19:06:11.000 UTC
Enviado: n - Timestamp: Mar 27, 2024 19:06:12.000 UTC
Enviado: w - Timestamp: Mar 27, 2024 19:06:13.000 UTC
Enviado: - Timestamp: Mar 27, 2024 19:06:14.000 UTC
Enviado: a - Timestamp: Mar 27, 2024 19:06:15.000 UTC
Enviado: n - Timestamp: Mar 27, 2024 19:06:16.000 UTC
Enviado: m - Timestamp: Mar 27, 2024 19:06:17.000 UTC
Enviado: n - Timestamp: Mar 27, 2024 19:06:18.000 UTC
Enviado: b - Timestamp: Mar 27, 2024 19:06:19.000 UTC
informatica@informatica-08:~/lab1/Actividad2$ █
```

Figura 4

```
informatica@informatica-08:~/lab1/Actividad2$ sudo python3 pingv1.py "mtqf htrt j
xyfx"
Enviado: m - Timestamp: Mar 27, 2024 19:14:56.000 UTC
Enviado: t - Timestamp: Mar 27, 2024 19:14:57.000 UTC
Enviado: q - Timestamp: Mar 27, 2024 19:14:58.000 UTC
Enviado: f - Timestamp: Mar 27, 2024 19:14:59.000 UTC
Enviado:  - Timestamp: Mar 27, 2024 19:15:00.000 UTC
Enviado: h - Timestamp: Mar 27, 2024 19:15:01.000 UTC
Enviado: t - Timestamp: Mar 27, 2024 19:15:02.000 UTC
Enviado: r - Timestamp: Mar 27, 2024 19:15:03.000 UTC
Enviado: t - Timestamp: Mar 27, 2024 19:15:04.000 UTC
Enviado:  - Timestamp: Mar 27, 2024 19:15:06.000 UTC
Enviado: j - Timestamp: Mar 27, 2024 19:15:07.000 UTC
Enviado: x - Timestamp: Mar 27, 2024 19:15:08.000 UTC
Enviado: y - Timestamp: Mar 27, 2024 19:15:09.000 UTC
Enviado: f - Timestamp: Mar 27, 2024 19:15:10.000 UTC
Enviado: x - Timestamp: Mar 27, 2024 19:15:11.000 UTC
informatica@informatica-08:~/lab1/Actividad2$ █
```

Figura 5

Tráfico a ip de loopback: se procede a dirigir el paquete ICMP construido por el script a la IP loopback “127.0.0.1”.

Como se puede observar en la figura 6, el paquete ha sido enviado efectivamente a la IP loopback “127.0.0.1”.

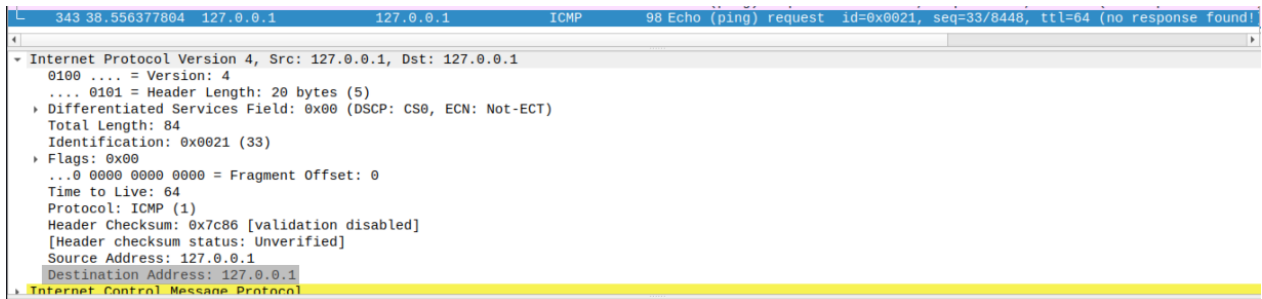


Figura 6

Se inyecta cifrado a tráfico: se oculta cada carácter del texto cifrado en el byte menos significativo del contador ubicado en el campo data de ICMP.

Como se puede observar en la figura 7, el caracter va oculto en el primer byte de Data, en este caso se esta ocultando una b, que al pasarlo a hexadecimal es un 62.

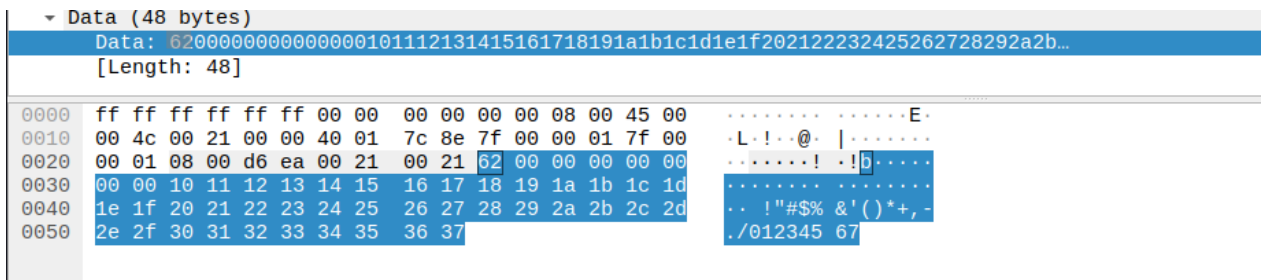


Figura 7

Mantiene ejecución (cada 1 segundo): el script se hizo de manera que los paquetes se envíen cada 1 segundo.

Como se puede observar en la figura 8, el script está hecho de manera que los paquetes se envíen cada 1 segundo, esto se muestra en la línea 78 del código.

```
--
49 if __name__ == "__main__":
50     if len(sys.argv) != 2:
51         print("Uso: python3 enviar_caracteres_icmp.py <texto>")
52         sys.exit(1)
53
54     # Obtener el texto a enviar desde los argumentos de línea de comandos
55     texto = sys.argv[1]
56
57     # Dirección IP destino (loopback)
58     ip_destino = "127.0.0.1"
59
60     # Iniciar el ID de paquete en 1
61     id_paquete = 1
62
63     # Iniciar el número de secuencia en 1
64     secuencia_paquete = 1
65
66     # Enviar cada carácter del texto
67     for caracter in texto:
68         # Enviar el carácter ICMP
69         enviar_caracter_icmp(caracter, ip_destino, id_paquete, secuencia_paquete)
70
71         # Incrementar el ID de paquete
72         id_paquete += 1
73
74         # Incrementar el número de secuencia
75         secuencia_paquete += 1
76
77         # Esperar 1 segundo antes de enviar el siguiente carácter
78         time.sleep(1)
```

Figura 8

Mantiene timestamp (ICMP): Se han agregado los 8 bytes correspondientes al Timestamp antes de los bytes correspondientes a la Data y después de los bytes correspondientes a Sequence Number, sin embargo, Wireshark no lo detectó como campo Timestamp y lo sumó a Data, pero está correctamente ubicados los 8 bytes de Timestamp, además de indicar correctamente la hora en el formato pedido (se puede ver en las figuras 4 y 5), tal como se puede ver en la figura 9:

Sequence Number (LE): 8448 (0x2100)										
▶ [No response seen]										
▼ Data (56 bytes)										
Data: 000000000660477e76200000000000000101112131415161718191a1b1c1d1e1f										
[Length: 56]										
000	ff	ff	ff	ff	ff	ff	00	00	00 00 00 00 08 00 45 00E.
010	00	54	00	21	00	00	40	01	7c 86 7f 00 00 01 7f 00	.T!..@.
020	00	01	08	00	f8	fe	00	21	00 21 00 00 00 00 66 04! .!....f.
030	77	e7	62	00	00	00	00	00	00 00 10 11 12 13 14 15	w·b.....
040	16	17	18	19	1a	1b	1c	1d	1e 1f 20 21 22 23 24 25 !"#\$\$%
050	26	27	28	29	2a	2b	2c	2d	2e 2f 30 31 32 33 34 35	&'()*+,- ./012345
060	36	37								67

Figura 9

Mantiene IP identification coherente: Como se puede observar en la figura 10, se mantiene IP Identification coherente.

Mantiene seq number coherente: Como se puede observar en la figura 10, se mantiene seq number coherente.

Mantiene ICMP identification coherente: Como se puede observar en la figura 10, se mantiene ICMP identification coherente.

Mantiene payload ICMP (3 bytes) coherente: Como se puede observar en la figura 10, se mantiene payload ICMP (3 bytes) coherente.

Mantiene payload ICMP (5 bytes 0x00): Como se puede observar en la figura 10, se mantiene payload ICMP (5 bytes 0x00).

Mantiene payload ICMP (desde 0x10 a 0x37): Como se puede observar en la figura 10, se mantiene payload ICMP (desde 0x10 a 0x37).

Mantiene checksum coherente: Como se puede observar en la figura 10, se mantiene checksum coherente.

No.	Time	Source	Destination	Protocol	Length	Info
5	5.571220785	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0x0001, seq=1/256, ttl=64 (no response found!)
10	6.623241779	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0x0002, seq=2/512, ttl=64 (no response found!)
11	7.685905474	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0x0003, seq=3/768, ttl=64 (no response found!)
12	8.750529683	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0x0004, seq=4/1024, ttl=64 (no response found!)
17	9.817874806	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0x0005, seq=5/1280, ttl=64 (no response found!)
18	10.866189317	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0x0006, seq=6/1536, ttl=64 (no response found!)
25	11.914378729	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0x0007, seq=7/1792, ttl=64 (no response found!)
26	12.966341312	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0x0008, seq=8/2048, ttl=64 (no response found!)
27	14.022427229	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0x0009, seq=9/2304, ttl=64 (no response found!)
32	15.082338427	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0x000a, seq=10/2560, ttl=64 (no response found!)
33	16.142660006	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0x000b, seq=11/2816, ttl=64 (no response found!)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1						
Internet Control Message Protocol						
Type: 8 (Echo (ping) request)						
Code: 0						
Checksum: 0xef5f [correct]						
[Checksum Status: Good]						
Identifier (BE): 1 (0x0001)						
Identifier (LE): 256 (0x0100)						
Sequence Number (BE): 1 (0x0001)						
Sequence Number (LE): 256 (0x0100)						

Figura 10

El código de la figura 11 envía caracteres individuales de un texto dado como paquetes ICMP de solicitud (echo request) a una dirección IP de destino específica. El texto se pasa como argumento en la línea de comandos al ejecutar el script.

1. Formateo del timestamp: La función `formatear timestamp` convierte un timestamp de tipo `int` en una cadena de texto formateada en el formato 'Mes Día, Año Hora:Minuto:Segundo.Milisegundo UTC'.

2. Envío de caracteres ICMP: La función `enviar caracter icmp` envía un carácter específico como un paquete ICMP de solicitud a la dirección IP de destino proporcionada. Construye el paquete ICMP con el carácter, un timestamp, y datos hexadecimales adicionales. Luego, utiliza `Scapy` para enviar el paquete ICMP a través de la función `send`.

3. Proceso principal:

- Verifica que se proporcione el texto como argumento en la línea de comandos. Si no se proporciona, muestra un mensaje de uso y sale del script.
- Obtiene el texto de la línea de comandos y establece la dirección IP de destino como `localhost` (127.0.0.1).
- Inicializa el ID de paquete y el número de secuencia en 1.
- Itera sobre cada carácter del texto: Llama a la función `enviar caracter icmp` para enviar el carácter como un paquete ICMP, incrementa el ID de paquete y el número de secuencia y espera 1 segundo antes de enviar el siguiente carácter, utilizando `time.sleep(1)`.

En resumen, este script utiliza paquetes ICMP para transmitir un texto carácter por carácter a una dirección IP específica, con un intervalo de 1 segundo entre cada carácter.

```

1 import sys
2 import time
3 import datetime
4 import struct
5 from scapy.all import IP, ICMP, send
6
7 def formatear_timestamp(timestamp):
8     """
9     Formatea el timestamp en el formato deseado.
10    """
11    return datetime.datetime.utcnow().strftime('%b %d, %Y %H:%M:%S.%f')[:-3] + ' UTC'
12
13 def enviar_caracter_icmp(caracter, ip_destino, id_paquete, secuencia_paquete):
14     """
15     Función que envía un carácter en un paquete ICMP request individual.
16     :param caracter: str, el carácter a enviar
17     :param ip_destino: str, la dirección IP destino
18     :param id_paquete: int, el identificador del paquete
19     :param secuencia_paquete: int, el número de secuencia del paquete
20     """
21     # Generar timestamp actual
22     timestamp = int(time.time())
23
24     # Formatear timestamp
25     timestamp_str = formatear_timestamp(timestamp)
26
27     # Guardar el caracter oculto
28     caracter_oculto = caracter
29
30     # Datos hex luego del caracter oculto
31     datos_hex = "00000000000000101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637"
32
33     # Obtener el timestamp como bytes (8 bytes)
34     timestamp_bytes = struct.pack("iQ", timestamp)
35
36     # Crear datos ICMP con el primer caracter oculto del texto de entrada, timestamp y datos hexadecimales
37     datos_icmp = bytes([ord(caracter_oculto)]) + bytes.fromhex(datos_hex)
38
39     # Crear el paquete ICMP echo request con timestamp y datos
40     paquete_icmp = IP(src="127.0.0.1", dst=ip_destino, id=id_paquete, ttl=64) / ICMP(type=8, code=0, id=id_paquete,
41     seq=secuencia_paquete) / timestamp_bytes / datos_icmp
42
43     # Enviar paquete
44     send(paquete_icmp, verbose=False)
45
46     # Mostrar caracter enviado y timestamp
47     print(f"Enviado: {caracter_oculto} - Timestamp: {timestamp_str}")
48
49 if __name__ == "__main__":
50     if len(sys.argv) != 2:
51         print("Uso: python3 enviar_caracteres_icmp.py <texto>")
52         sys.exit(1)
53

```

Figura 11


```
52     sys.exit(1)
53
54     # Obtener el texto a enviar desde los argumentos de línea de comandos
55     texto = sys.argv[1]
56
57     # Dirección IP destino (loopback)
58     ip_destino = "127.0.0.1"
59
60     # Iniciar el ID de paquete en 1
61     id_paquete = 1
62
63     # Iniciar el número de secuencia en 1
64     secuencia_paquete = 1
65
66     # Enviar cada carácter del texto
67     for caracter in texto:
68         # Enviar el carácter ICMP
69         enviar_caracter_icmp(caracter, ip_destino, id_paquete, secuencia_paquete)
70
71         # Incrementar el ID de paquete
72         id_paquete += 1
73
74         # Incrementar el número de secuencia
75         secuencia_paquete += 1
76
77     # Esperar 1 segundo antes de enviar el siguiente carácter
78     time.sleep(1)
```

Figura 12

3.3. Actividad 3

Finalmente, se procede a guardar el tráfico generado anteriormente, para luego crear un script en Python3 que permite leer la capturar PCAPNG y obtener el carácter cifrado de cada paquete, de esta manera, se reconstruye el mensaje cifrado, para finalmente iterar en cada unas de las posibles ROT-N (26). Si en alguna de esas rotaciones se logra percibir algún mensaje en claro del habla español, este será el candidato a ser el mensaje en claro, por lo que será un mensaje en verde, seguido del ROT-Corrimiento correspondiente.

Para esto, se le debe dar un input al script de la siguiente manera:

```
sudo python3 readv1.py <nombre de la captura ICMP de wireshark>.pcapng
```

Como se puede observar en las figuras 13 y 14, se han descifrado los 2 textos cifrados anteriormente, a partir de todas las combinaciones de las rotaciones de César(26) seguido de su respectivo Rot-n con n el corrimiento correspondiente, el texto descifrado se indica en color verde.

```

(base) admin@iMac-de-Cristobal Lab1 % sudo python3 readv1.py cesar.pcapng
larycxpajorj h bnpdarmjm nw anmnb ROT- 26
mbszdyqbksk i coqebnskn ox bonoc ROT- 25
nctaezrclqtl j dprfctolo py cpopd ROT- 24
odubfasdmrum k eqsgdupmp qz dqpqe ROT- 23
pevcgbtensvn l frthevqnq ra erqrf ROT- 22
qfwdhcufofwo m gsuifwrro sb fsrsg ROT- 21
rgxeidvgpuxp n htvjgxspc tc gtsth ROT- 20
shyfjewhqvyq o iuwkhytqt ud hutui ROT- 19
tizgkfxirwzr p jvxlizuru ve ivuvj ROT- 18
ujahlgyjsxas q kwymjavsv wf jwvwk ROT- 17
vkbimhzktybt r lxznkbwtw xg kxwxl ROT- 16
wlcjnlialuzcu s myaolcxux yh lyxym ROT- 15
xmdkojbmadv t nzbpmdyvy zi mzyzn ROT- 14
ynelpkcnwbew u oacqnezwz aj nazao ROT- 13
zofmqldoxcfx v pbdrofaxa bk obabp ROT- 12
apgnrmepydgy w qcespgbyb cl pcbcq ROT- 11
bqhosnfqzehz x rdftqhczc dm qdcdr ROT- 10
criptografia y seguridad en redes ROT- 9
dsjquphsbgjb z tfhvsjebe fo sfefr ROT- 8
etkrvqitchkc a ugiwtkfcf gp tgfgu ROT- 7
fulswrjudild b vhjxulgdg hq uhghv ROT- 6
gvmtxskvejme c wikyvmheh ir vihiw ROT- 5
hwnuytlwfkf d xjlzwnifi js wjiix ROT- 4
ixovzumxglog e ykmaxojgj kt xkjky ROT- 3
jypwavnymph f zlnbypkhh lu ylkly ROT- 2
kzqxbwozinqi g amoczqlil mv zmlma ROT- 1
(base) admin@iMac-de-Cristobal Lab1 %

```

Figura 13

```
[(base) admin@iMac-de-Cristobal Lab1 % sudo python3 readv1.py cesar2.pcapng ]
mtqf htrt jxyfx ROT- 26
nurg iusu kyzgy ROT- 25
ovsh jvtv lzahz ROT- 24
pwti kwuw mabia ROT- 23
qxuj lxvx nbcjb ROT- 22
ryvk mywy ocdkc ROT- 21
szwl nzxz pdeld ROT- 20
taxm oaya qefme ROT- 19
ubyn pbzb rfgnf ROT- 18
vczo qcac sg hog ROT- 17
wdap rdbd thiph ROT- 16
xebq sece uijqi ROT- 15
yfcr tfdf vjkrj ROT- 14
zgds ugeg wklsk ROT- 13
ahet vhf h xlm t l ROT- 12
bifu wigi ymnum ROT- 11
cjgv xjhj znovn ROT- 10
dkhw ykik aopwo ROT- 9
elix zljl bpqxp ROT- 8
fmjy amkm cqryq ROT- 7
gnkz bnl n drs z r ROT- 6
hola como estas ROT- 5
ipmb dpnp ftubt ROT- 4
jqnc eqoq guvcu ROT- 3
krod frpr hvwdv ROT- 2
lspe gsqs iwxew ROT- 1
(base) admin@iMac-de-Cristobal Lab1 %
```

Figura 14

El código de la Figura 20 aplicado anteriormente, este realiza un análisis de un archivo de captura de red en formato pcapng, en busca de paquetes ICMP de tipo request”. Luego, extrae los dos primeros bytes del campo data de estos paquetes, los decodifica como texto y los almacena en una lista llamada ciphertexts.

A continuación, intenta descifrar estos textos cifrados utilizando el cifrado César con todos los posibles corrimientos, es decir, corrimientos de 0 a 25. Para cada corrimiento, verifica si el texto descifrado contiene palabras en español comparándolas con un conjunto de palabras en español previamente cargadas desde un archivo de texto llamado "spanish words.txt".

Si se encuentra al menos una palabra en español en el texto descifrado, se imprime el texto en verde junto con el valor del corrimiento César utilizado (la clave). Si no se encuentran palabras en español, se imprime el texto descifrado normalmente.

En resumen, este código intenta descifrar posibles mensajes cifrados presentes en un archivo de captura de red utilizando el cifrado César y muestra los resultados resaltando aquellos que contienen palabras en español.

```
import sys
import pyshark
from termcolor import colored

# Verificar si se proporciona un archivo pcapng como argumento
if len(sys.argv) != 2:
    print("Uso: python3 readv1.py <archivo.pcapng>")
    sys.exit(1)

# Obtener el nombre del archivo pcapng del argumento de la línea de comandos
pcap_file = sys.argv[1]

# Diccionario de palabras en español
with open("spanish_words.txt", "r", encoding="utf-8") as file:
    spanish_words = set(word.strip().lower() for word in file)

# Cargar el archivo pcapng
cap = pyshark.FileCapture(pcap_file)

# Lista para almacenar todos los textos cifrados
ciphertexts = []

# Iterar sobre todos los paquetes en el archivo
for packet in cap:
    # Verificar si es un paquete ICMP de tipo "request"
    if "ICMP" in packet and packet.icmp.type == "8":
        # Acceder a los dos primeros bytes del campo data_data de ICMP
        first_two_bytes_hex = packet.icmp.data_data[:2]

        # Convertir los dos primeros bytes de hexadecimal a una cadena de bytes
        bytes_data = bytes.fromhex(first_two_bytes_hex)

        # Convertir la cadena de bytes a texto y agregarlo a la lista
        message = bytes_data.decode('utf-8')
        ciphertexts.append(message)

# Iterar sobre los corrimientos de César de 0 a 25
for shift in range(26):
    # Aplicar el corrimiento de César a cada texto cifrado y agregarlo a la lista de resultados
    shifted_text = ''
    for text in ciphertexts:
        shifted_message = ''
        for char in text:
            if char.isalpha():
                shifted_char = chr(((ord(char.lower()) - 97 + shift) % 26) + 97)
                if char.isupper():
                    shifted_char = shifted_char.upper()
                shifted_message += shifted_char
```

Figura 15

```
        shifted_message += shifted_char
    else:
        shifted_message += char
    shifted_text += shifted_message

# Buscar palabras en español en el texto cifrado
found_spanish_word = any(word.lower() in spanish_words for word in shifted_text.split())

# Imprimir el texto cifrado en verde si se encontró al menos una palabra en español
if found_spanish_word:
    print(colored(shifted_text, "green"), "ROT-", 26 - shift) # indica el corrimiento cesar(l>
else:
    print(shifted_text)
```

Figura 16

Conclusiones y Comentarios

Issues

- Para la actividad 2, el tema de posicionar los bytes ha sido un problema, ya que en Wireshark no se logró mostrar adecuadamente el Timestamp a pesar de estar bien puesto.
- En la actividad 2, ha sido difícil ocultar los caracteres en el primer byte de Data ya que el largo de bytes sobrepasaba 48, pero finalmente se pudo concretar.
- En la actividad 2, ha sido difícil colocar el correcto Checksum, ya que cuando no estaba correcto no llegaban Reply, finalmente se pudo concretar los Checksum adecuados y el paquete parece genuino.
- Finalmente, en la actividad 3, fue difícil leer el primer byte de Data del PCAP, finalmente se pudo lograr, mostrando todas las combinaciones y el mensaje descifrado.