

Informe Laboratorio 1

Sección 2

Alumno Sebastián Chávez
e-mail: sebastian.chavez1@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	8

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).

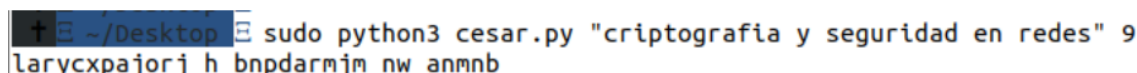
A usted le han encomendado auditar si efectivamente estos sistemas son capaces de detectar las filtraciones a través de tráfico de red. Debido a que el programa ping es ampliamente utilizado desde dentro y hacia fuera de la empresa, su tarea será crear un software que permita replicar tráfico generado 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 pcabcq
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     qfwdhcufotwo m gsuifwr or sb fsrsg
22     pevcbtensvn l frthevq nq ra erqrf
23     odubfasdmrum k eqsgdupmp qz dqpqe
24     nctaezrclqtl j dprfctolo py cpopd
25     mbszdyqbksk i coqebnkn ox bonoc

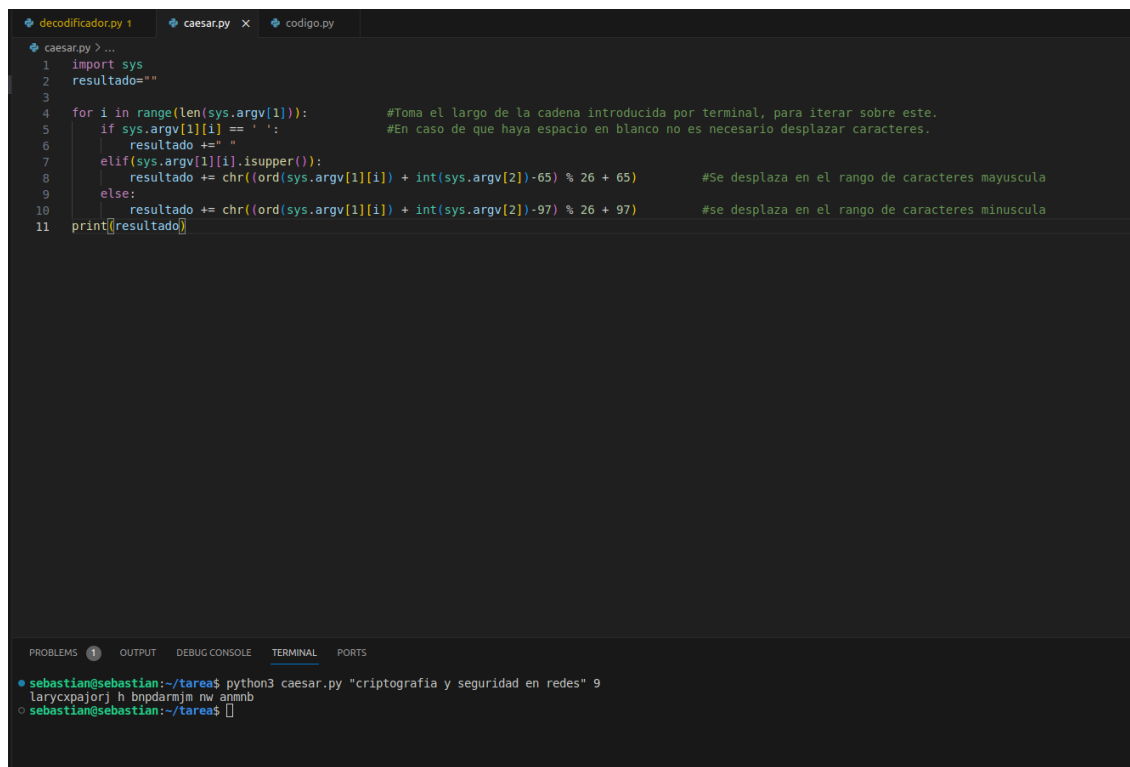
```

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

Para la primera actividad se realizó un script de python el cual emula el cifrado cesar, el cual puede ser apreciado en la siguiente imagen:



```
decodificador.py 1 | caesar.py x | codigo.py
caesar.py > ...
1 import sys
2 resultado=""
3
4 for i in range(len(sys.argv[1])):      #Toma el largo de la cadena introducida por terminal, para iterar sobre este.
5     if sys.argv[1][i] == ' ':          #En caso de que haya espacio en blanco no es necesario desplazar caracteres.
6         resultado += " "
7     elif sys.argv[1][i].isupper():
8         resultado += chr((ord(sys.argv[1][i]) + int(sys.argv[2]) - 65) % 26 + 65)    #Se desplaza en el rango de caracteres mayuscula
9     else:
10        resultado += chr((ord(sys.argv[1][i]) + int(sys.argv[2]) - 97) % 26 + 97)    #se desplaza en el rango de caracteres minuscula
11 print(resultado)
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
sebastian@sebastian:~/tarea$ python3 caesar.py "criptografia y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb
sebastian@sebastian:~/tarea$
```

Figura 1: desarrollo script y resultado

El cual se trata de un sencillo algoritmo Caesar, el cual permite codificar una string ingresada por la terminal luego de además agregar el corrimiento. En la imagen, además, se puede apreciar el funcionamiento así como también el resultado.

3.2. Actividad 2

Para la segunda actividad se tenía que enviar el string resultante de la actividad anterior utilizando scapy. Para lograr el cometido se realizo el siguiente script en python:

```

pingv4.py - tarea - Visual Studio Code
Run Terminal Help
readv2.py 1 caesar.py pingv4.py X
pingv4.py > ...
4 import struct
5
6 from scapy.all import sr1, IP, ICMP, send, Ether
7
8 #a=IP()/ICMP()
9 #print(sys.argv[1])
10 id_paquete=os.getpid() & 0xFFFF #manera de generar un identificador unico utilizando el pid
11 tiempo=time.time() #este valor se utiliza para el timestamp
12 for z in range(len(sys.argv[1])):
13     timestamp_bytes=struct.pack('<Q', int(tiempo)) #pasa el timestamp a a bytes con en Little Endian
14     a=IP()/ICMP(seq=(z+1))/bytes(timestamp_bytes) #arma el paquete, con la condición de que la secuencia aumenta en uno durante el ciclo,
15     caracter=sys.argv[1][z] #toma el string pasado como argumento de la terminal
16     a.ttl=64 #setea el time to live
17     a.src="127.0.0.1" #ip de fuente
18     a.dst="127.0.0.1" #ip de destino(se escoge esta para loopback)
19     a[ICMP].id=id_paquete #se carga la id generada con el pid al paquete
20     a[Raw].load=f"{caracter}\x0e\x0e\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17" \
21     "\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27" \
22     "\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37" #Info de payload sacada de wireshark, con el añadido del caracte
23     #a.show() #esta info se copio de un ping cualquiera enviado de forma norma
24     send(a)
25     time.sleep(1)
26
27
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
22 pevcbtensvn l frthevqng ra ergrf
23 odubfasmrum k eqsgdupmp qz dqpgq
24 nctaezrlqtl j dprfctolo py cpopd
25 mbszdyqbksk l coqebnskn ox bonoc
sebastian@sebastian:~/tarea5$ sudo python3 pingv4.py "larycxpajorj h bnpdarmjm nw anmb"
[sudo] password for sebastian:
Sent 1 packets.
Sent 1 packets.
Ln 20, Col 35 (3 selected) Spaces: 4 UTF-8 LF Python 3.10.12 64-bit

```

Figura 2: desarrollo script

En la figura se puede apreciar tanto la ejecución así como también los mensajes de feedback de que se están enviando los paquetes. Se envía un paquete por carácter de la string encriptada. El script en sí se basa en crear con ayuda de scapy un paquete, tratando que sea lo más similar posible a uno normal. En la siguiente figura se podrá apreciar como llegan estos en wireshark:

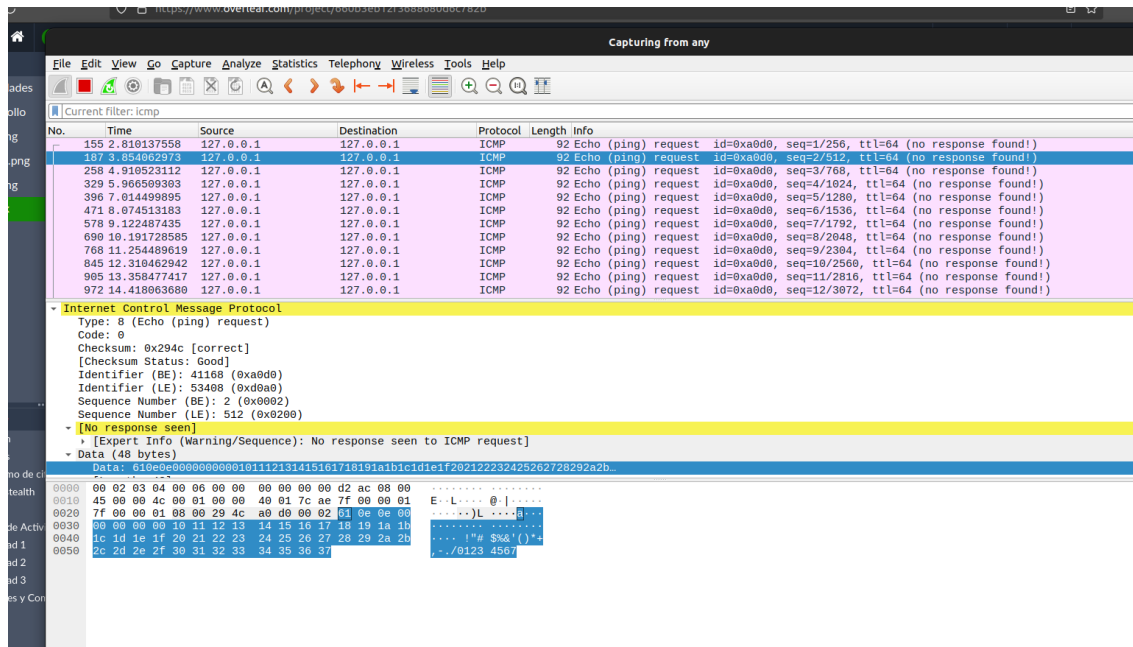


Figura 3: paquetes en wireshark

Primero, se aprecia que en los elementos de protocolo, destino, ttl, info, etc. son tal cual como un paquete normal generado cuando uno hace un ping con dirección loopback. También, si se observa el primer byte este contiene el primer carácter de la string encriptada que fue enviada. El checksum también está correcto. Así como también los bytes solicitados. También se logra el enviar los paquetes con un segundo de diferencia entre ellos.

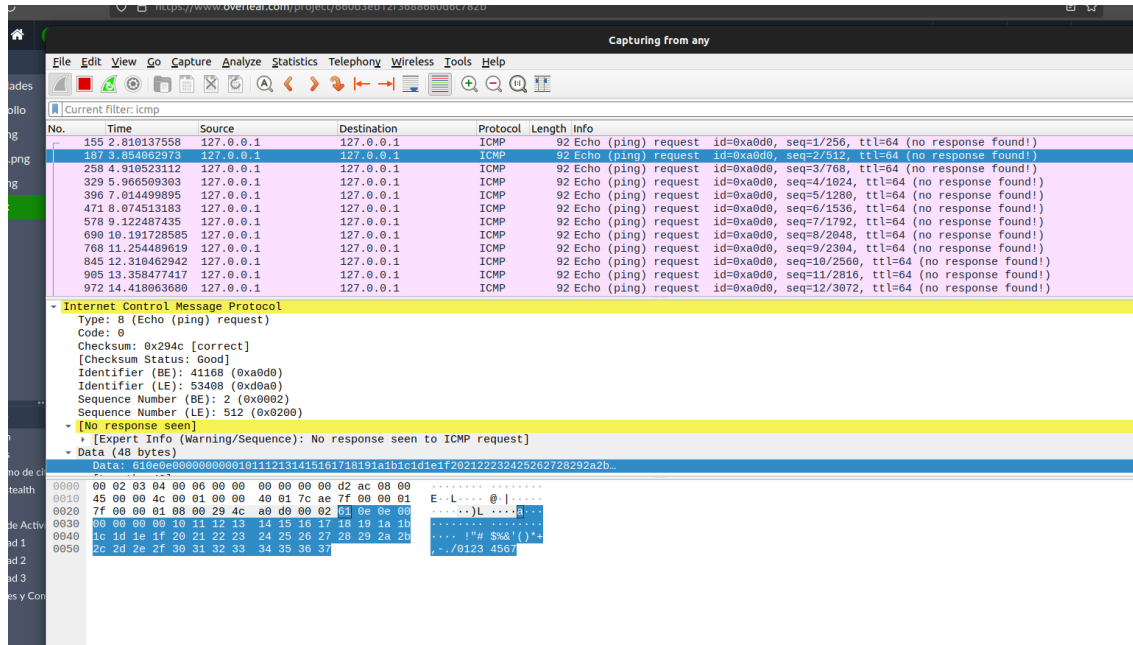


Figura 4: seq en wireshark

En esta figura también se aprecia que la secuencia efectivamente funciona como fue explicada y solicitada previamente. Así mismo, también llega inyectado el segundo carácter del string encriptado(a). Adicionalmente esta captura se guardó como archivo pcapng para ser utilizado en la siguiente sección.

3.3. Actividad 3

Para esta sección era necesario crear un script el cual, en primer lugar, fuera capaz de extraer de un archivo pcapng los caracteres de la string encriptada los cuales fueron inyectados en cada uno de los paquetes enviados.

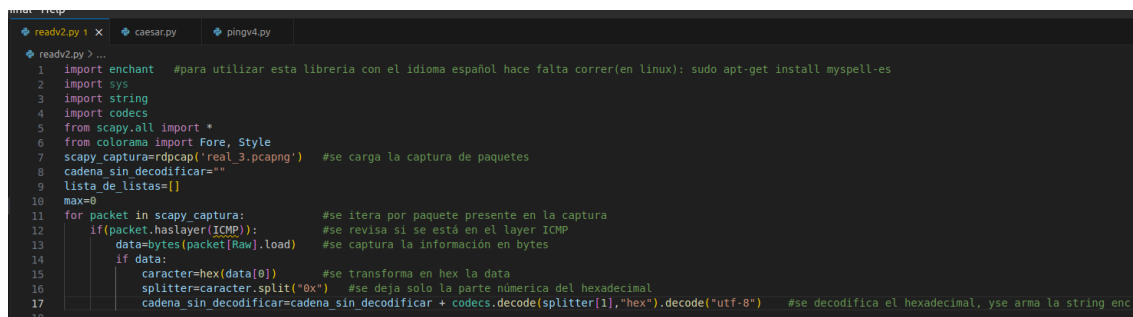


Figura 5: script sección extraer bytes paquete

En esta sección del código se extrae de la captura los bytes inyectados previamente, además de decodificarlos.


```

19 alfabeto=string.ascii_lowercase #alfabeto a ser utilizado para desencriptar el cifrado Caesar
20 mas_probable=[] #lista encargada de mantener cuenta sobre la string desencriptada más probable a ser la real
21 diccionario=enchant.Dict("es_CL") #se instancia enchant, que se utiliza como diccionario para comparar las palabras desencriptadas
22
23 mensaje_encryptado=(cadena_sin_decodificar).strip() #se eliminan los espacios en blanco de la string encryptada
24 mensaje_descriptado=""
25
26 for a in range(26): #se itera por el rango del alfabeto
27     for x in range(len(mensaje_encryptado)): #se itera por el largo del mensaje encryptado
28         if mensaje_encryptado[x] in alfabeto: #se revisa si el char está dentro del alfabeto
29             posicion=alfabeto.find(mensaje_encryptado[x])
30             nuevaposicion=(posicion-a)%26 #se usa de corrimiento el rango del alfabeto, a, en este caso
31             nuevocaracter=alfabeto[nuevaposicion]
32             mensaje_descriptado+=nuevocaracter
33         else:
34             mensaje_descriptado += mensaje_encryptado[x] #arriba se mueve por el alfabeto haciendo el algoritmo inverso.
35
36 revisar=mensaje_descriptado.split() #se separa la string desencriptada, tomando como separador espacio en blanco
37 for i in range(len(revisar)): #se itera por el largo de la string desencriptada
38     if diccionario.check(revisar[i]): #se revisa si cada una de las palabras desencriptadas matchea una del diccionario en español
39         mas_probable.append(revisar[i]) #en caso de que resulta en un match, se suma la palabra a la lista.
40 if len(mas_probable)>max:
41     max=len(mas_probable) #se deja un max global, equivalente a la mayor cantidad de palabras reales post desencripción
42 lista_de_listas.append([mensaje_descriptado, len(mas_probable)]) #lista de lista donde se agrega la string, y el número de matches en el diccionario
43 mensaje_descriptado=""
44 mas_probable.clear() #se limpia el contador de matches para el siguiente ciclo
45 for b in range(len(lista_de_listas)):
46     if max < lista_de_listas[b][1]:
47         print(Fore.GREEN + f"{b} " + lista_de_listas[b][0] + Style.RESET_ALL) #se pinte en verde la string deco con más matches en el diccionario
48     else:
49         print(f"{b} " + lista_de_listas[b][0]) #en otro caso se pinte sin color.

```

Figura 6: seq en wireshark

Luego, en esta sección se construye nuevamente la string encryptada a partir de los caracteres extraídos de los paquetes. Posteriormente, se revisa con la función `enchant` por palabras para así determinar si cada una de las palabras desencriptadas están presentes en el diccionario. Finalmente, se hacen contadores en base a la cantidad de palabras desencriptadas con sentido y se pinte en verde la que tiene mayor cantidad. Ese algoritmo previamente descrito fue el procedimiento utilizado para poder determinar que corrimiento es el correcto para desencriptar. Finalmente, en la figura a continuación se puede ver como desencripta e imprime en verde el resultado adecuado.

```

1 import enchant #para utilizar esta libreria con el idioma español hace falta correr(en linux): sudo apt-get install myspell-es
2 import sys
3 import string
4 import codecs
5 from scapy.all import *
6 from colorama import Fore, Style
7 scapy_capture=rdpcap('real_3.pcapng') #se carga la captura de paquetes
8 cadena_sin_decodificar=""
9 lista_de_listas=[]
10 max=0
11 for packet in scapy_capture: #se itera por paquete presente en la captura
12     if(packet.haslayer(ICMP)): #se revisa si se está en el layer ICMP
13         data=bytes(packet[Raw].load) #se captura la información en bytes
14         if data:
15             caracter=hex(data[0]) #se transforma en hex la data
16             splitter=caracter.split("0x") #se deja solo la parte numérica del hexadecimal
17             cadena_sin_decodificar=cadena_sin_decodificar + codecs.decode(splitter[1],"hex").decode("utf-8") #se decodifica el hexadecimal, y se arma la string enc
18
19 alfabeto=string.ascii_lowercase #alfabeto a ser utilizado para desencriptar el cifrado Caesar

```

```

File "/home/sebastian/tarea/readv2.py", line 1, in <module>
import enchant #para utilizar esta libreria con el idioma español hace falta correr(en linux): sudo apt-get install myspell-es
ModuleNotFoundError: No module named 'enchant'
sebastian@sebastian:~/tareas$ python3 readv2.py
0 larycpajorj h bnpdarmjm mw anamb
1 kqzxbwozinqi g amocqllil mv zmlma
2 jypwavyhnpf f zlnbypkpk lu ykliz
3 ixovzumglog e ykmaxogj kt xkiky
4 hnuyltlwknf d xjlnwnifi js wjijx
5 qvmtkskvejme c wikymneh ir vihiw
6 fulswrjudild b vjhxludg hq uhghv
7 etkrvqitckk a uqwkifer gp tglgu
8 dsjqphsbjzb z tlnvsjebe fo sfiwt
9 criptografia y seguridad en redes
10 bqhosnfqzehz x rdtqhczz dm qdcdr
11 apgnrmepdyg w qcespgbyb cl pcbbq
12 zofnqldoxcix v pndofaxa bk obabp
13 ynelpkcnubew u oacqnezvz aj nazao
14 xmdkojbmadv t nzbpmdivy zi mzyzn
15 wlcjnlaluccu s myaolcxux yh lyxym
16 wkzahnzkytbt r lxznkbtv xg kowxl
17 ujahlgysxas q kwmjavsv wf jwvkk
18 tizgkfxirwzr p jvxlizuru ve ivuvj
19 shyfjewhqvya o luwkyhtat ud hutui
20 rpxaidguxpa n hrvjxps ts qstsh
21 qfwdhucufotwo m gsulfrwr sb fsrsg
22 pevcbtensvn l frthevqnp ra erqrf
23 odubfasdrum k eqsgdupmp qz ddpqe
24 nctezrcratl j dprictelo py cpodd
25 mszdyqbkpsk l cceqbskn ox bonoc
sebastian@sebastian:~/tareas$

```

Figura 7: resultado script desencriptar

Conclusiones y Comentarios

Para concluir se puede decir que se lograron de manera exitosa los objetivos impuestos inicialmente, además de que se logró durante la realización de las tareas lo maleable y mutables que son los campos con los cuales se envía paquetes por internet. De la misma forma, también se logró apreciar en primera persona lo complicado que puede llegar a ser monitorear correctamente debido a la facilidad que con la que se pueden fabricar desde 0 paquetes que son prácticamente iguales a uno generado de forma legítima. Por otra parte, también se pudo aprender muchísimo sobre el tipo de data que va incluida dentro de los campos de data.

Issues

Las 4 mayores complicaciones que surgieron durante el proceso de realización del proceso fueron:

En primera instancia sería el timestamp, y más en específico, el formato pues para representar correctamente el formato es necesario pasar a little endian y no es muy claro el como lograrlo dentro de python. Para resolver este problema fue necesario buscar y encontrar la librería struct, y a continuación leer como manejar los bytes e inyectarlos dentro del paquete en el formato deseado.

En segunda instancia otro error que surgió fue el de imitar de forma correcta los bytes de data, para que fueran iguales a los de un paquete creado enviando un ping. Para solucionar esta sección fue necesario entrar a wireshark y copiar esa sección de un ping previamente enviado para luego inyectar en los paquetes fabricados.

Otra gran problemática surgida durante la realización de los procedimientos fue el uso de las librerías de scapy en general, pues si bien hay documentos y bastantes comandos de ayuda, aún así era difícil entender el cómo aplicar correctamente debido a la falta de ejemplos en muchos casos. Para solucionar este problema fue de mucha ayuda stackoverflow así como también la inteligencia artificial chatgpt, de estas dos fuentes era posible encontrar ejemplos así como también estrategias y guías de librerías a utilizar.

Finalmente, la gran otra problemática fue el cómo extraer de forma exitosa el carácter inyectado dentro del primer byte de data. En general ocurrió que era complicado entender, en primer lugar, sobre qué iteraba o qué elementos del payload estaba tratando de extraer. A lo anterior se suma también la necesidad constante de cambiar ya sea de bytes a hex o ascii, etc. Para solucionar esto se recurrió a ayuda de inteligencia artificial junto con stackoverflow, donde fue posible idear la forma utilizada para finalmente extraer el byte de forma correcta y así lograr el procedimiento final.