

Informe Laboratorio 1

Sección 1

Sergio Saavedra
e-mail: sergio.saavedra1@mail.udp.cl

Marzo 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	5
3.1. Actividad 1	5
3.2. Actividad 2	6
3.3. Actividad 3	9

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).

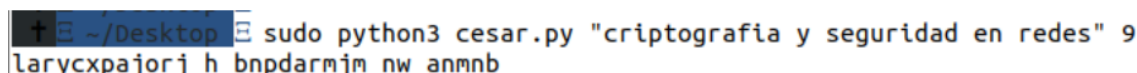
A usted le han encomendado auditar si efectivamente estos sistemas son capaces de detectar las filtraciones a través de tráfico de red. Debido a que el programa ping es ampliamente utilizado desde dentro y hacia fuera de la empresa, su tarea será crear un software que permita replicar tráfico generado 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 sfefst
9      criptografia y seguridad en redes
10     bqhosnfqzehz x rdftqhczc dm qdcdr
11     apgnrmepdygy w qcespgbyb cl pcbbcq
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     qfwdhcufo two m gsuifwror 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

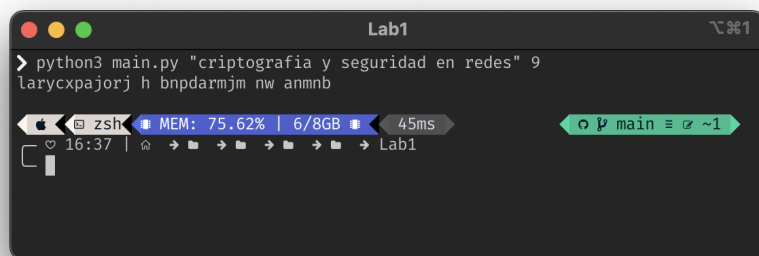
3.1. Actividad 1

Para la primera parte del laboratorio, se solicita generar un código en python que cifre texto con el algoritmo Cesar. Para esto se hizo uso de herramientas externas quedando de la siguiente manera:

```
1 import sys
2
3 def cifrar_cesar(texto, desplazamiento):
4     texto_cifrado = ''
5     for caracter in texto:
6         if caracter.isalpha():
7             mayuscula = caracter.isupper()
8             caracter = caracter.lower()
9             codigo = ord(caracter) + desplazamiento
10            if codigo > ord('z'):
11                codigo -= 26
12            elif codigo < ord('a'):
13                codigo += 26
14            caracter_cifrado = chr(codigo)
15            if mayuscula:
16                caracter_cifrado = caracter_cifrado.upper()
17            texto_cifrado += caracter_cifrado
18        else:
19            texto_cifrado += caracter
20    return texto_cifrado
21
22 def descifrar_cesar(texto_cifrado, desplazamiento):
23     return cifrar_cesar(texto_cifrado, -desplazamiento)
24
25 if __name__ == "__main__":
26     if len(sys.argv) != 3:
27         print("Mal escrito el corrimiento o palabra")
28         sys.exit(1)
29
30     texto_original = sys.argv[1]
31     desplazamiento = int(sys.argv[2])
32     texto_cifrado = cifrar_cesar(texto_original, desplazamiento)
33     texto_decifrado = descifrar_cesar(texto_cifrado, desplazamiento)
34     print(texto_cifrado)
```

Cabe recalcar que para **todos los códigos que reciben parámetros**, es la biblioteca **sys** con su método **argv** los que permiten obtener un arreglo con los parámetros indicados en la ejecución de la terminal.

De este código se obtiene la siguiente respuesta:



```
> python3 main.py "criptografia y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb
```

Figura 1: Terminal primer script (cifrado) con texto y corrimiento de parámetros

Esto se debe a que el algoritmo genera un corrimiento de 9 (valor indicado como parámetro final en la terminal). Se puede entender el corrimiento del algoritmo con la siguiente imagen:

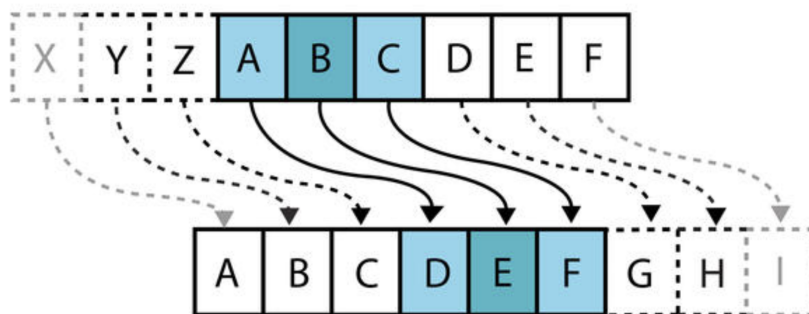


Figura 2: Algoritmo Cesar con corrimiento 3

3.2. Actividad 2

Para la segunda parte del laboratorio, se solicita generar un script en python que envíe los caracteres del cifrado de la actividad pasada (*larycxpajorj h bnpdarmjm nw anmnb*).

Para esto se usa la herramienta scapy con sus métodos **IP**, **ICMP**, **Send**, en el cual **IP** apunta a la ip loopback (interfaz de red virtual del router) **127.0.0.1**, el método **ICMP** asegura que sean secuenciales con sus argumentos ID y SEQ. **Send** emite el tráfico a la red especificada, todo esto **cada un segundo** con la libreria **time** con su método **sleep**. El código empleado fue el siguiente:

```

1 import os
2 import sys
3 import time
4 import struct
5 import datetime
6 from scapy.all import IP, ICMP, send, IPOption_Timestamp
7
8 def buildICMP(data, id, seq):
9     icmp_data = bytes(data + '\x00\x00', 'utf-8')
10    icmp_data += b'\x00\x00\x00\x00\x00'
11
12    # Obtener timestamp actual en formato UTC
13    timestamp_utc = int(time.time())
14
15    # Convertir el timestamp a hexadecimal y luego invertir el orden (big-
16    endian)
17    timestamp_hex = hex(timestamp_utc)[2:]
18    timestamp_bytes = bytes.fromhex(timestamp_hex.rjust(32, '0'))
19    timestamp_bytes = timestamp_bytes[::-1]
20
21    # A adir el timestamp antes del payload
22    icmp_data = timestamp_bytes + icmp_data
23
24    icmp_data += bytes(range(0x10, 0x38))
25    payload = icmp_data
26
27    packetICMP = IP(dst='127.0.0.1') / ICMP(id=id, seq=seq) / payload
28    return packetICMP
29
30 if __name__ == "__main__":
31     if len(sys.argv) != 2:
32         print("Parametros mal ingresados")
33         sys.exit(1)
34     message = sys.argv[1]
35     id_counter = 1
36     seq_counter = 1
37     for char in message:
38         packetICMP = buildICMP(char, id_counter, seq_counter)
39         send(packetICMP)
40         id_counter += 1
41         seq_counter += 1
42         time.sleep(1)

```

De la ejecución se obtiene la siguiente respuesta:

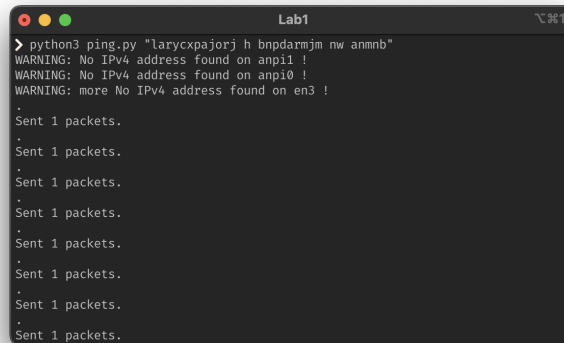


Figura 3: Terminal segundo script (stealth) con envío de paquetes a IP loopback

Se puede verificar que la ejecución del script se ha ejecutado correctamente con **Wireshark**, teniendo la siguiente vista de tráfico:

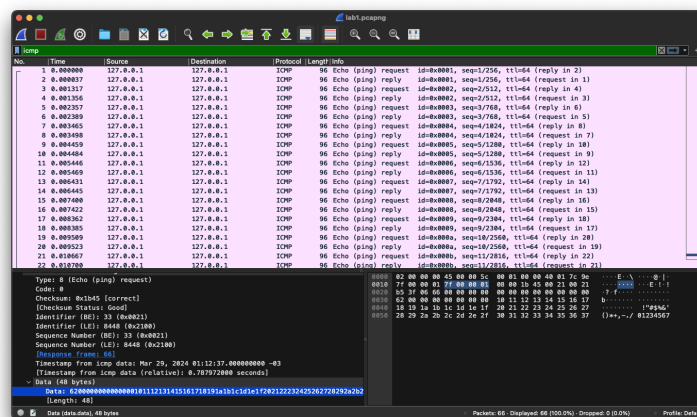


Figura 4: Tráfico total de Wireshark

En el primer paquete se aprecia que en *Data* se está enviando el carácter correspondiente al primero del texto cifrado, además de obtener un **checksum coherente** con status *good*.

Además se puede apreciar que los atributos de ID y SEQ van cambiando a medida que llegan más paquetes, por lo que **se mantiene un SEQ e ICMP identification coherentes**.

Sumado a esto se mantiene el timestamp del paquete, siendo posible verlo en la sección de *Internet Control Message Protocol* en **Timestamp from ICMP data**.

También se puede visualizar en *Data*, que el payload es el indicado en la rúbrica, siendo coherentes los **3 primeros bytes**, los siguientes **5: 0x00**, y desde el **0x10 al 0x37**.

Tabla de valores palabra <i>criptografía</i>							
Caracteres		Checksum	ID(BE)	ID(LE)	Seq. (BE)	Seq. (LE)	Data
c	l	0xa8fa	1	256	1	256	6c2020
r	a	0xb3f8	2	512	2	512	612020
i	r	0xa2f6	3	768	3	768	722020
p	y	0x9bf4	4	1024	4	1024	792020
t	c	0xb1f2	5	1280	5	1280	632020
o	x	0x9cf0	6	1536	6	1536	782020
g	p	0xa4ee	7	1792	7	1792	702020
r	a	0xb3ec	8	2048	8	2048	612020
a	j	0xaaea	9	2304	9	2304	6a2020
f	o	0xa5e8	10	2560	10	2560	6f2020
i	r	0xa2e6	11	2816	11	2816	722020
a	j	0xaae4	12	3072	12	3072	6a2020

3.3. Actividad 3

En la parte final del laboratorio, se pide generar un programa en python que reciba el mensaje transmitido en el paso anterior y decifrarlo, generando todos los posibles corrimientos e indicar en color verde la opción más probable de ser el mensaje original (**criptografía y seguridad en redes**). Para eso se empleó el siguiente código:

```

1 import sys
2 from scapy.all import *
3 from colorama import init, Fore
4
5 def decrypt_cesar(message, shift):
6     decrypted_message = ""
7     for char in message:
8         if char.isalpha():
9             char_code = ord(char)
10            if char.isupper():
11                decrypted_char = chr(((char_code - 65 - shift) % 26) + 65)
12            else:
13                decrypted_char = chr(((char_code - 97 - shift) % 26) + 97)
14            decrypted_message += decrypted_char
15        else:
16            decrypted_message += char
17    return decrypted_message
18
19 # Comprobar si se proporciona la ubicación del archivo pcapng
20 if len(sys.argv) != 2:
21     print("Uso: python3 decrypt_pcap.py archivo.pcapng")
22     sys.exit(1)
23
24 # Leer el archivo pcapng
25 file_path = sys.argv[1]
26 packets = rdpcap(file_path)

```

```

27
28 icmp_packets = [pkt for pkt in packets if ICMP in pkt and pkt[ICMP].type
    == 0]
29
30 # Obtener el primer byte de los paquetes ICMP y concatenarlos
31 message_bytes = b"".join([pkt[Raw].load[16:17] for pkt in icmp_packets])
32
33 init()
34 init(autoreset=True)
35
36 for shift in range(26):
37     decrypted_message = decrypt_cesar(message_bytes.decode(), shift)
38     if(decrypted_message == "criptografia y seguridad en redes"):
39         print( Fore.GREEN + f"{shift}: {decrypted_message}")
40     else:
41         print(f"{shift}: {decrypted_message}")

```

Se puede apreciar que fue necesario agregar en el paso 2, el timestamp en el payload al principio del mismo, por lo que al momento de realizar la lectura del archivo *.pcapng* fue necesario especificar que leyera el byte 16 del payload.

De la ejecución se obtiene la siguiente respuesta:

```

> python3 decifrar.py lab1.pcapng
WARNING: No IPv4 address found on anpi1 !
WARNING: No IPv4 address found on anpi0 !
WARNING: more No IPv4 address found on en3 !
0: larycxpajorj h bnrdarmjm nw anmnb
1: kzqxbwozinqi g amoczqll mv zmlma
2: jypwavnymhph f zlnbyphkh lu yklz
3: ixovzumxglog e ykmaxojgj kt xkjkj
4: hwnuytlwfknf d xjzwnifi js wjijx
5: gvmtxskvejme c wikyvmeh ir vihiw
6: fulswrjudild b vhxulgdg hq uhghv
7: etkrvqitchkc a ugiwtkfcf gp tfggu
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: zofnqldoxcfx v pbdrfakx bk obabp
13: ynelpkcnwbew u oacqnezvz aj nazao
14: xmdkojbmadv t nzbpmdyvy z1 mzyzn
15: wlejnialuzcu s myaolexux yh lyxym
16: vkbimhzytbt r lxznkbtw xg kxwxl
17: ujahlgysxas q kwmjxvsv wf jwvwx
18: tizgkfxirwz p jwllizuru ve iuvvj
19: shyfjewhvyo o iuwkhytat ud hutui
20: rgxeidvgpuxp n htvjxspz tc gtsth
21: qfwdhcfutwo m gsuifwrwr sb fsrsg
22: pevcbtensvn l frthevqmq ra erqrf
23: odubfasdtrum k eqsgdupmp qz dqape
24: nctaezrcqltl j dprfctolo py cpopd
25: mbszdyqbksk i coqebnkn ox bonoc

```

Figura 5: Lista corrimientos con posible respuesta

Una vez visualizada en la terminal la respuesta más probable de todos los corrimientos posibles, se identifica el texto original cifrado en la actividad 1, finalizando la actividad actual.

Conclusiones y Comentarios

Una vez finalizado el experimento, se aprecia que fue posible realizar el laboratorio con éxito.

Es de gran utilidad el comprender el funcionamiento de este tipo de algoritmos, para tener una visión más avanzada sobre los tipos de cifrados y los eventos en los que está presente la herramienta, tanto a nivel computacional como en la vida real, y el cómo identificarlos haciendo uso de herramientas como Wireshark, las cuales jugarán un papel importante en nuestra carrera profesional.

Además, en criptografía el cifrado o descifrado son una serie de pasos bien definidos que se pueden seguir como un procedimiento. Su aplicación se extiende a todo tipo de casos, incluso los más sutiles como pueden ser los idiomas o lenguas habladas en distintos lugares.

En conclusión, la exitosa implementación este proyecto refleja la aplicación efectiva de los conocimientos adquiridos en el presente laboratorio. Este proyecto no solo fortaleció la comprensión de la lógica de cifrado y el uso de algoritmos, sino que también destacó la importancia de estos principios en el área de criptografía y seguridad.

Issues

1. **Desconocimiento para realizar el algoritmo Cesar:** Para poder realizar y emplear el algoritmo Cesar se tuvo que recurrir a diferentes lecturas, pudiendo así implementar el cifrado de manera exitosa.
2. **Mal envío de paquetes (no seq, sin payload):** Al principio no se visualizaba la data de los paquetes en el tráfico, esto debido a que solo se estaba usando el método ICMP(), cuando había que definir la secuencia de cada carácter con su id, una vez realizado esto se pudo ejecutar esa parte de la actividad (Stealth) sin problemas.
3. **No visualizar tráfico en ip loopback:** Al revisar la rubrica de evaluación se dio cuenta de que el tráfico apuntaba a ip de loopback, por lo que se tuvo que cambiar de 1.1.1.1 a 127.0.0.1. Sin embargo no se visualizaban los paquetes, por lo que al buscar en la documentación de wireshark se indicó que para poder revisar el tráfico de este medio era necesario indicarlo en la captura en la página principal.
4. **Visualizar el doble de paquetes:** Al momento de realizar el ping del paso dos (Stealth), se podían visualizar el doble de paquetes, siendo la única diferencia el atributo *Info* donde la mitad eran reply y otros request. Por lo que al hacer el descifrado se tuvo que indicar que solo considerará los que fueran de tipo reply, aunque se intentó cambiar en el código para solo emitir con sr (Send and receive packets) el problema se mantuvo, por lo que fue necesario realizar lo indicado anteriormente.