

Informe Laboratorio 1

Sección 2

Alumno Matias Herrera
e-mail: matias.herrera2@mai.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	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).

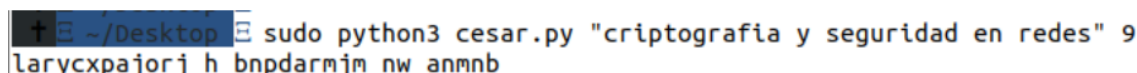
A usted le han encomendado auditar si efectivamente estos sistemas son capaces de detectar las filtraciones a través de tráfico de red. Debido a que el programa ping es ampliamente utilizado desde dentro y hacia fuera de la empresa, su tarea será crear un software que permita replicar tráfico generado 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 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 esta actividad se genera el siguiente código con ayuda de chatGPT, el código es el siguiente, se importa la biblioteca sys para la utilización de la terminal como input al momento de llamar al archivo:

Aquí tienes un programa en Python que cifra un texto utilizando el cifrado César con un desplazamiento dado como parámetro:

```
python
Copy code

import sys

def cifrar_cesar(texto, desplazamiento):
    texto_cifrado = ""
    for caracter in texto:
        # Verifica si el caracter es una letra
        if caracter.isalpha():
            # Obtiene el valor ASCII del caracter
            valor = ord(caracter)
            # Calcula el desplazamiento aplicado al valor ASCII
            nuevo_valor = valor + desplazamiento
            # Verifica si el nuevo valor está fuera del rango de letras en ASCII
            if caracter.islower():
                if nuevo_valor > ord('z'):
                    nuevo_valor -= 26
                elif nuevo_valor < ord('a'):
                    nuevo_valor += 26
            elif caracter.isupper():
                if nuevo_valor > ord('Z'):
                    nuevo_valor -= 26
                elif nuevo_valor < ord('A'):
                    nuevo_valor += 26
            # Agrega el caracter cifrado al texto cifrado
            texto_cifrado += chr(nuevo_valor)
        else:
            # Si el caracter no es una letra, se agrega tal cual al texto cifrado
            texto_cifrado += caracter
    return texto_cifrado
```

Figura 1: ChatGPT generando un programa de cifrado cesar funcion

En la figura 1 podemos apreciar como define la función del cifrado cesar tanto para que la palabra a utilizar tome tanto mayúsculas como minúsculas y verifica si son números o no, en caso de ser números los cuales sean nuestro input se agregan directamente a la palabra cifrada. Para la figura 2 el main verifica que los comandos de la terminal han sido ingresados correctamente a ella para poder ser utilizados en la función anteriormente mencionada para luego retornar el resultado de dicha función que es la palabra cifrada en la terminal.

En la figura 3 utilizamos la palabra Encryptacion con un desplazamiento de 5 caracteres hacia la derecha, resultando en lo que se aprecia en dicha figura.

```

if __name__ == "__main__":
    # Verifica si se proporcionaron los argumentos adecuados
    if len(sys.argv) != 3:
        print("Uso: sudo python3 programa.py 'String a cifrar' <desplazamiento>")
        sys.exit(1)

    # Obtener el texto y el desplazamiento de los argumentos de línea de comandos
    texto = sys.argv[1]
    desplazamiento = int(sys.argv[2])

    # Cifra el texto utilizando el cifrado César
    texto_cifrado = cifrar_cesar(texto, desplazamiento)

    # Imprime el texto cifrado
    print("Texto cifrado:", texto_cifrado)

```

Figura 2: ChatGPT generando un programa de cifrado cesar main

```

use@Linux:~$ sudo python3 C_cesar.py "Encriptacion" 5
Texto cifrado: Jshwnuyfhnts

```

Figura 3: ChatGPT generando un programa de cifrado cesar

3.2. Actividad 2

El código de esta actividad es el siguiente, se importan las bibliotecas de sys (explicada anteriormente) y scapy con el fin de poder generar tráfico y paquetes en una interfaz de red utilizando dicha biblioteca:

```

1 import sys
2 from scapy.all import *
3
4 def enviar_paquete_icmp(caracter, secuencia, identificacion):
5     # Crea un paquete ICMP con el caracter en el campo de datos
6     relleno = b'!'#$%&'()*+,-./01234567"
7     paquete = IP(dst="127.0.0.1") / ICMP(type="echo-request", id=identificacion, seq=secuencia) / bytes(caracter, 'utf-8') /
8     relleno
9     # Envía el paquete ICMP
10    send(paquete, verbose=True)
11
12 if __name__ == "__main__":
13     # Verifica si se proporcionó el argumento adecuado
14     if len(sys.argv) != 2:
15         print("Uso: sudo python3 archivo.py 'texto cifrado'")
16         sys.exit(1)
17
18     # Obtener el texto cifrado de los argumentos de línea de comandos
19     texto_cifrado = sys.argv[1]
20
21     # Genera secuencia e ID diferentes para cada paquete ICMP
22     secuencia = 1
23     identificacion = 1
24
25     # Envía cada caracter del texto cifrado en un paquete ICMP
26     for caracter in texto_cifrado:
27         enviar_paquete_icmp(caracter, secuencia, identificacion)
28         secuencia += 1
29         identificacion += 1

```

Figura 4: ChatGPT programa de envío de paquetes ICMP con mensajes encriptado

la figura 4 podemos apreciar la definición de la función `enviar_paquete_icmp` en la cual consiste en recibir el texto cifrado por carácter para generar un paquete de tipo ICMP con destino `'127.0.0.1'` y características personalizadas como lo es el id, la secuencia y su información. Ahora ya creado el programa se procede a utilizar el siguiente formato como se puede ver en la figura 5

[illegible]

Figura 5: Terminal comando ping con palabra cifrada

Luego de generar y enviar los paquetes a la interface loopback estas se pueden observar en la figura 6 la cual es el programa wireshark obteniendo los paquetes de tipo ICMP con cada caracter del mensaje encriptado anteriormente.

```
1 0.000000000 127.0.0.1 127.0.0.1 ICMP 65 Echo (ping) request id=0x0001, seq=1/256, ttl=64 (no response found!)
2 0.059706261 127.0.0.1 127.0.0.1 ICMP 65 Echo (ping) request id=0x0002, seq=2/512, ttl=64 (no response found!)
3 0.105339789 127.0.0.1 127.0.0.1 ICMP 65 Echo (ping) request id=0x0003, seq=3/768, ttl=64 (no response found!)
4 0.146864647 127.0.0.1 127.0.0.1 ICMP 65 Echo (ping) request id=0x0004, seq=4/1024, ttl=64 (no response found!)
5 0.196224006 127.0.0.1 127.0.0.1 ICMP 65 Echo (ping) request id=0x0005, seq=5/1280, ttl=64 (no response found!)
6 0.258935438 127.0.0.1 127.0.0.1 ICMP 65 Echo (ping) request id=0x0006, seq=6/1536, ttl=64 (no response found!)
7 0.303112495 127.0.0.1 127.0.0.1 ICMP 65 Echo (ping) request id=0x0007, seq=7/1792, ttl=64 (no response found!)
8 0.347105852 127.0.0.1 127.0.0.1 ICMP 65 Echo (ping) request id=0x0008, seq=8/2048, ttl=64 (no response found!)
9 0.388041524 127.0.0.1 127.0.0.1 ICMP 65 Echo (ping) request id=0x0009, seq=9/2304, ttl=64 (no response found!)
10 0.428398160 127.0.0.1 127.0.0.1 ICMP 65 Echo (ping) request id=0x000a, seq=10/2560, ttl=64 (no response found!)
11 0.487526047 127.0.0.1 127.0.0.1 ICMP 65 Echo (ping) request id=0x000b, seq=11/2816, ttl=64 (no response found!)
12 0.535432502 127.0.0.1 127.0.0.1 ICMP 65 Echo (ping) request id=0x000c, seq=12/3072, ttl=64 (no response found!)

Sequence Number (LE): 3072 (0x0c00)
[No response seen]
[Expert Info (Warning/Sequence): No response seen to ICMP request]
[No response seen to ICMP request]
[Severity level: Warning]
[Group: Sequence]
Data (23 bytes)
Data: 731232425262728292a2b2c2d2e2f3031323334353637
[Length: 23]

0000 ff ff ff ff ff ff 00 00 00 00 00 08 00 45 00 .....E-
0010 00 33 00 01 00 00 40 01 7c c7 7f 00 00 01 7f 00 ....@.|.....
0020 00 01 08 00 94 02 00 0c 00 0c 73 21 23 24 25 26 .....s!$%&
0030 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 '()*+,-./0123456
0040 37 7
```

Figura 6: Captura de paquetes ICMP wireshark

La informacion de la figura 6 se observa que el Id de cada paquete es de forma secuencial, es decir que cada paquete contiene un caracter del mensaje encriptado con el fin de no ser intersectado con facilidad, luego de realizar dicho envio de paquetes, se procede a interceptar los mensajes, actividad que sera comentada en la siguiente seccion.

3.3. Actividad 3

En esta actividad generamos un código el cual con ayuda de scapy se analizará el archivo pcapng de wireshark donde tenemos los paquetes ICMP hechos en la anterior actividad para mostrar todos los posibles descifrados de la palabra creada y determinar cuál es la más probable, para esto el código es el siguiente:

```

1 import sys
2 import os
3 from scapy.all import *
4
5 def obtener_mensaje(pcapng_file):
6     mensaje = ""
7     # Lee el archivo pcapng
8     for pkt in rdpcap(pcapng_file):
9         # Verifica si es un paquete ICMP
10        if ICMP in pkt:
11            # Extrae el byte de la data del paquete ICMP y lo agrega al mensaje
12            data = pkt[ICMP].load
13            mensaje += chr(data[0])
14    return mensaje
15
16 def descifrar_cesar(texto, desplazamiento):
17     texto_descifrado = ""
18     for caracter in texto:
19         # Verifica si el caracter es una letra
20         if caracter.isalpha():
21             # Obtiene el valor ASCII del caracter
22             valor = ord(caracter)
23             # Calcula el desplazamiento aplicado al valor ASCII
24             nuevo_valor = valor - desplazamiento
25             # Ajusta el valor si se sale del rango de letras
26             if caracter.islower():
27                 if nuevo_valor < ord('a'):
28                     nuevo_valor += 26
29             elif caracter.isupper():
30                 if nuevo_valor < ord('A'):
31                     nuevo_valor += 26
32             # Agrega el caracter descifrado al texto descifrado
33             texto_descifrado += chr(nuevo_valor)
34         else:
35             # Si el caracter no es una letra, se agrega tal cual al texto descifrado
36             texto_descifrado += caracter
37     return texto_descifrado

```

Figura 7: Código de lectura Parte 1

Tenemos la función `obtener_mensaje` la cual toma como valores el archivo pcapng para realizar la captura de el mensaje encriptado y retornar la palabra completa como string, luego se define la función de descifrado de cesar el cual obtendrá todos los desplazamientos posibles en lo cual retornará el texto descifrado en dicho desplazamiento ya sea desde 0 a 25 desplazamientos posibles, luego se define la función `cargar_diccionario` la cual entrega un archivo txt con todas las palabras del diccionario español para verificar que la palabra descifrada sea coherente y así determinar que dicha palabra o mensaje tenga sentido, luego tenemos la función `imprimir_opciones` la cual en conjunto con la función anterior determinará si la palabra obtenida en dicho desplazamiento tiene sentido, si fuera el caso imprimirá dicha palabra en verde enmarcando que es la posible palabra que era en un principio, pero en caso contrario simplemente la imprimirá en color normal. Para entender mejor el funcionamiento recomiendo observar las figuras 7, 8, 9.

Como resultado de este código a la hora de llamar al archivo nos retorna lo siguiente (figura 10):


```

39 def cargar_diccionario():
40     # Carga un diccionario de palabras en español
41     with open("Diccionario.txt", "r", encoding="utf-8") as file:
42         return set(word.strip() for word in file)
43
44 def imprimir_opciones(texto_original, opciones):
45     diccionario_espanol = cargar_diccionario()
46     mejor_opcion = None
47     max_palabras_en_diccionario = 0
48     i = 0
49     for opcion in opciones:
50         palabras_en_diccionario = sum(1 for palabra in opcion.split() if palabra.lower() in diccionario_espanol)
51         if palabras_en_diccionario > max_palabras_en_diccionario:
52             mejor_opcion = opcion
53             max_palabras_en_diccionario = palabras_en_diccionario
54
55     for opcion in opciones:
56         if opcion == mejor_opcion:
57             print("\033[92m" + str(i) + ': ' + opcion + "\033[0m") # Impresión en verde para la opción más probable
58             i += 1
59         else:
60             print(str(i) + ': ' + opcion)
61             i += 1
62
63 if __name__ == "__main__":
64     # Verifica si se proporcionó el argumento adecuado
65     if len(sys.argv) != 2:
66         print("Uso: python3 programa.py archivo.pcapng")
67         sys.exit(1)
68
69     # Obtiene el nombre del archivo pcapng de los argumentos de línea de comandos
70     archivo_pcapng = sys.argv[1]
71
72     # Obtiene el mensaje transmitido del archivo pcapng
73     mensaje_transmitido = obtener_mensaje(archivo_pcapng)
74     print("Mensaje transmitido:", mensaje_transmitido)

```

Figura 8: Código de lectura Parte 2

```

75
76 # Genera todas las posibles combinaciones de descifrado utilizando el cifrado César
77 opciones_descifrado = []
78 for i in range(26):
79     opcion = descifrar_cesar(mensaje_transmitido, i)
80     opciones_descifrado.append(opcion)
81
82 # Imprime las opciones de descifrado, indicando la más probable en verde
83 imprimir_opciones(mensaje_transmitido, opciones_descifrado)

```

Figura 9: Código de lectura Parte 3

Como se puede observar en la figura 10 nos retorna el mensaje transmitido por los paquetes ICMP obtenidos, luego de eso nos retornas todos los posibles mensajes descifrados y resaltando en verde cual es la palabra o mensaje mas coherente, en este caso el mensaje era Encriptacion.

```
use@Linux:~$ sudo python3 lectura.py cap_stealth.pcapng
Mensaje transmitido: Jshwnuyfhnts
0: Jshwnuyfhnts
1: Irgvmtxegmsr
2: Hqfulswdflrq
3: Gpetkrvcekqp
4: Fodsjqubdjpo
5: Encriptacion
6: Dmbqhoshzbhnm
7: Clapgnryagml
8: Bkzofmqxzflk
9: Ajyne1pwyekj
10: Zixmdkovxdji
11: Yhwlcjnuwcih
12: Xgvkbimtvbhg
13: Wfujahlsuagf
14: Vetizgkrtzfe
15: Udshyfjqsyed
16: Tcrgxeiprxdc
17: Sbqfwdhoqwcb
18: Rapevcgnpvba
19: Qzodubfmouaz
20: Pynctaelntzy
21: Oxmbzskmsyx
22: Nwlarycjlrxw
23: Mvkzqxbikqvv
24: Lujypwahjpvu
25: Ktixovzgiout
```

Figura 10: Resultado de terminal

Conclusiones y Comentarios

En conclusion el funcionamiento de encriptacion cesar es una forma de encriptacion clasica pero para estandares actuales de seguridad de mensajes son faciles de descifrar que alguien poco experimentado podria realizar con ayuda de un codigo generado por IA, en lo cual si uno mendara informacion sensible a travez de paquetes ICMP por la red y genera trafico encriptado a travez del metodo anteriormente mencionado seria intersectado facilmente y descifrado en cosa de minutos.

Issues

Dentro de las dificultades en estas 3 actividades fueron los siguientes. Primero, en la actividad 2 hubo dificultades a la hora de generar los paquetes ICMP de forma correcta para tener todos los parametros coherentes como lo es el ID y seq que son parametros que deben generarse de forma coherente a la hora de mandar un mensaje como era solicitado y para solucionarlo genere 2 parametros que tanto el id como la secuencia fueran aumentado en 1 al momento de enviar un paquete ICMP por la interface. Segundo, tambien existio un problema a la hora de generar el payload para cada paquete con el fin de insertar el mensaje encriptado en por caracter en cada uno de los paquetes en lo cual entendi que tenia que definir un array para que cada caracter de la palabra entregada por terminal tenia que tener un valor individual para poder ser entregado de forma consecutiva a la hora de ingresar al for de la figura 4. Tercero en la actividad 3 existieron dificultades a la hora de obtener el mensaje transmitido en la actividad 2 ya que si intentaba obtener un payload mas completo me generaria

complicaciones a la hora de trabajar con dichos datos, por lo que tome la decision de solo transmitir el caracter en dicho payload y nada mas. Y cuarto a la hora de obtener el mensaje o la opcion mas probable no me retornaba la opcion correcta o la mas coherente, asi que por teminos del ejercicio cree la funcion para obtener un diccionario de palabras en español para realizar un analisis previo a cada iteracion de la opcion descifrada en cada desplazamiento y verificar que dicho mensaje descifrado posible tenga sentido.