

Informe Laboratorio 1

Sección 1

Franco Ramírez Molina
e-mail: franco.ramirez1@mail.udp.cl

Marzo de 2024

Índice

1. Descripción	2
2. Actividades	2
2.1. Algoritmo de cifrado	2
2.2. Modo stealth	2
2.3. MitM	4
3. Desarrollo de Actividades	5
3.1. Actividad 1	5
3.2. Actividad 2	7
3.3. Actividad 3	11

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).

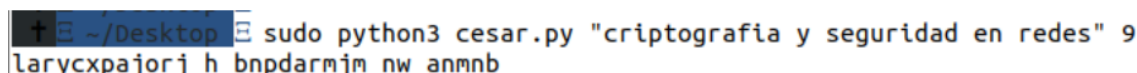
A usted le han encomendado auditar si efectivamente estos sistemas son capaces de detectar las filtraciones a través de tráfico de red. Debido a que el programa ping es ampliamente utilizado desde dentro y hacia fuera de la empresa, su tarea será crear un software que permita replicar tráfico generado 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.

2. Actividades

2.1. Algoritmo de cifrado

1. Generar un programa, en python3, 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, que permita enviar los caracteres del string (el del paso 1) en varios paquetes ICMP request (un caracter por paquete en el byte menos significativo del contador ubicado en el campo data de ICMP) para que de esta forma no se gatillen sospechas sobre la filtración de datos.

Para la generación del tráfico ICMP, deberá basarse en los campos de un paquete generado por el programa ping basado en Ubuntu, según lo visto en el lab anterior disponible acá.

El envío deberá poder enviarse a cualquier IP. Para no generar tráfico malicioso dentro de esta experiencia, se debe enviar el tráfico a la IP de loopback.

```

$ sudo python3 pingv4.py "larycxpajorj h bnpdarmjm nw anmnb"
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.

```

A modo de ejemplo, en este caso, cada paquete transmite un caracter, donde el último paquete transmite la letra b, correspondiente al caracter en plano “s”.

Data (48 bytes)		
Data: 6260090000000000101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637		
[Length: 48]		
0000	ff ff ff ff ff ff 00 00 00 00 00 00 08 00 45 00E.
0010	00 54 00 01 00 00 40 01 76 9b 7f 00 00 01 7f 06	.T....@. v.....
0020	06 06 08 00 56 83 00 01 00 21 64 22 13 05 00 00	...V... !d"....
0030	00 00 62 60 09 00 00 00 00 00 10 11 12 13 14 15	..b`.....
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

2.3. MitM

1. Generar un programa, en python3, 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.

```

$ sudo python3 readv2.py cesar.pcapng
0      larycxpajorj h bnpdarmjm nw anmnb
1      kzqxbwozinqi g amoczqlil mv zmlma
2      jypwavnyhmpf f zlnbypkhk lu yklklz
3      ixovzumxglog e ykmaxojgj kt xkjky
4      hwnuytlwfknf d xjlzwnifi js wjijx
5      gvmtxskvejme c wikyvmeheh ir vihiw
6      fulswrjudild b vhjxulgdg hq uhghv
7      etkrvqitchkc a ugiwtkfcf gp tgfgu
8      dsjquphsbgjb z tfhvsjebe fo sfefst
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     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 htvjgxspc tc gtsth
21     qfwdhcufo two m gsuifwror 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 los 4 mayores problemas o complicaciones que usted tuvo durante el proceso del laboratorio y de qué forma los solucionó.

3. Desarrollo de Actividades

En el presente informe se aplicaran los contenidos aprendidos en clases y laboratorios, tratando temas como protocolo ICMP, algoritmos de cifrado, generación y análisis de tráfico. El informe constará con imágenes y texto explicativo sobre los mecanismos utilizados para completar todos los requerimientos de las actividades.

3.1. Actividad 1

Para el primer ítem se pedía generar un programa utilizando el lenguaje de programación **python**, requiriendo que el programa pudiese cifrar texto mediante el algoritmo de cifrado *Cesar*. Para esto se precisó ayuda de la inteligencia artificial *ChatGPT 3.5* al cual se le pasó el siguiente prompt:

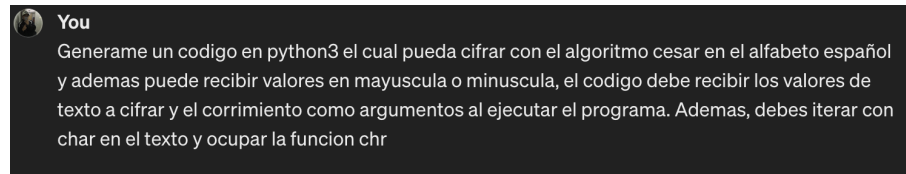


Figura 1: Prompt para algoritmo cesar en python

Para méritos del informe el código se ha modificado con respecto al código original proporcionado por la IA, dado que lo genera con *print()* adicionales los cuales eran innecesarios para el desarrollo del laboratorio. Finalmente así es como queda el código:

```

1  import sys
2
3  def cifrado_cesar(texto, corrimiento):
4      texto_cifrado = ""
5      for char in texto:
6          if char.isalpha():
7              if char.islower():
8                  nuevo_char = chr((ord(char) - ord('a') + corrimiento) % 26 + ord('a'))
9              else:
10                 nuevo_char = chr((ord(char) - ord('A') + corrimiento) % 26 + ord('A'))
11                 texto_cifrado += nuevo_char
12             else:
13                 texto_cifrado += char
14         return texto_cifrado
15
16     # Obtener los argumentos de la línea de comandos
17     if len(sys.argv) != 3:
18         sys.exit(1)
19
20     texto = sys.argv[1]
21     corrimiento = int(sys.argv[2])
22
23     # Aplicar el cifrado César e imprimir el texto cifrado
24     texto_cifrado = cifrado_cesar(texto, corrimiento)
25     print(texto_cifrado)

```

Como se puede apreciar en el código 3.1, se define una función llamada ***cifrado_cesar()*** que recibe como parámetros el texto a cifrar y el corrimiento a aplicar, recordando que el cifrado Cesar consiste en sustituir cada letra de la cadena de texto por una desplazada en el abecedario según el corrimiento dado. Es justamente esto lo que hace la línea de código **8** y **10**, las cuales por cada iteración del ciclo toma un carácter del texto a cifrar y aplica el siguiente proceso:

- Convierte el carácter a su respectivo código ASCII
- Lo deja como un valor entre 0 y 25
- Aplica el corrimiento dado verificando que no se pase del 25
- Lo vuelve a dejar a su valor en ASCII
- Lo vuelve a pasar a carácter

Así, ya podemos ejecutar el código el cual responde de la siguiente manera:

```
~/Documents/Universidad/Cripto/lab01 (0.074s)
python3 cesar.py "criptografía y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb
```

Figura 2: Respuesta del código con el texto cifrado

Se puede observar en la figura 2 que la respuesta del código es exitosa, se ha aplicado un desplazamiento 9 en cifrado Cesar a la frase *"criptografía y seguridad en redes"* dando como resultado la frase *"larycxpajorj h bnpdarmjm nw anmnb"*.

3.2. Actividad 2

En la segunda actividad y luego de haber cifrado el código, se precisaba enviar este código secreto mediante generación de paquetes ICMP. Estos paquetes no debía levantar sospechas con respecto al trafico general de la empresa, es decir, se debían recrear paquetes ICMP completos, cosa que no se distinguiera entre el trafico normal.

Análisis de trafico

Antes de empezar a generar trafico y enviar nuestros códigos cifrados, se debió hacer un análisis de trafico ICMP para así ver su comportamiento y los campos requeridos que se usaran para generar el trafico posteriormente. Para esto, se realizo un **ping** a la IP de loopback y capturada con wireshark, dando como resultado los siguientes tipos de paquetes.

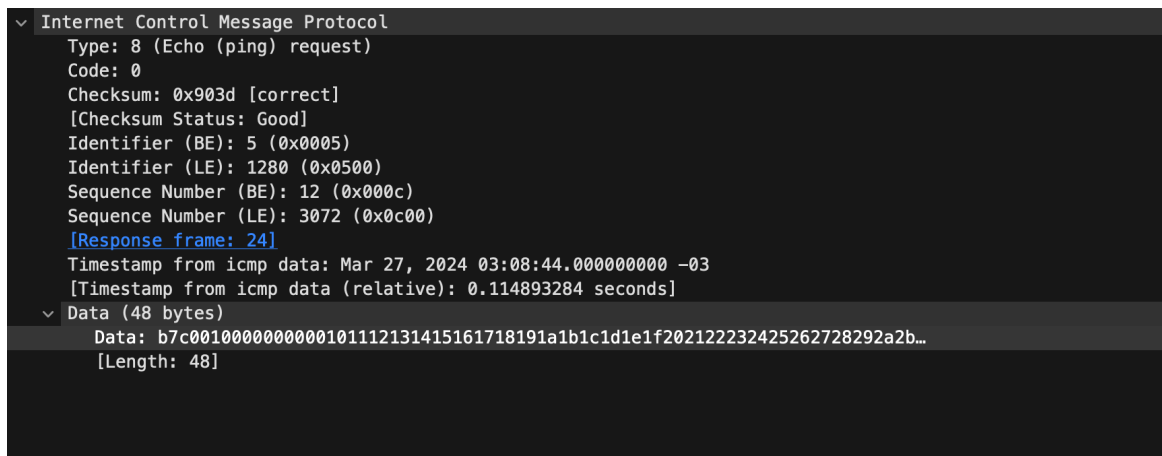


Figura 3: Paquete ICMP

En la figura 3 se pueden observar los distintos campos a replicar en el paquete ICMP con el código, entre ellos se encuentra el **type**, **identififer**, **sequence** y **data**, siendo este último el mas importante dado que en este campo se introducirán los caracteres cifrados para poder enviarlos a través de la red. Esto se hará ocupando el byte menos significativos del contador, es decir, reemplazar cada carácter en ASCII por el byte que esta mas a la izquierda. En el caso de la figura 3 el contador es **b7c001**, teniendo que reemplazar el valor **b7** por el carácter cifrado correspondiente.

Creación de Paquete

Para la generación de tráfico ICMP se ocupó la herramienta de manipulación de paquetes **Scapy**, con la que mediante un programa escrito en python se pudo diseñar paquetes ICMP con los campos respectivos, como se vera a continuación.

```

1  from scapy.all import *
2  import os
3  import sys
4  import time
5
6  # Loopback IP
7  ip_loopback = '127.0.0.1'
8
9  def convert_payload(data, index):
10     icmp_data = bytes(data, 'utf-8') + os.urandom(1) + bytes([index])
11     icmp_data += bytes('\x00\x00\x00\x00\x00', 'utf-8')
12     icmp_data += bytes(range(0x10, 0x38))
13
14     return icmp_data
15
16 def enviar_icmp(payload_data, index):
17     paquete = IP(src=ip_loopback, dst=ip_loopback)/ICMP(type=8, id=1, seq=index)/Raw(load=payload_data)
18     send(paquete)
19     time.sleep(1)
20
21
22 if len(sys.argv) != 2:
23     sys.exit(1)
24
25 texto = sys.argv[1]
26
27 for i in range(len(texto)):
28     caracter = texto[i]
29     payload_full = convert_payload(caracter, i)
30     enviar_icmp(payload_full, i)

```

Como se aprecia en la código 3.2 se pueden encontrar 2 funciones que manejan la lógica del código. En primer lugar se itera sobre el texto recibido en el argumento del código, esto se hace dado que el requerimiento es enviar un paquete por cada carácter del texto, luego se envía el carácter y el índice de cada iteración a la primera función llamada **convert_payload()** la cual genera el campo *data* a enviar en el payload del paquete ICMP. El algoritmo implementado en esta función es bastante simple dado que sigue el estándar del parámetro *data* de un paquete ICMP visto en la figura 3, siguiendo la siguiente regla:

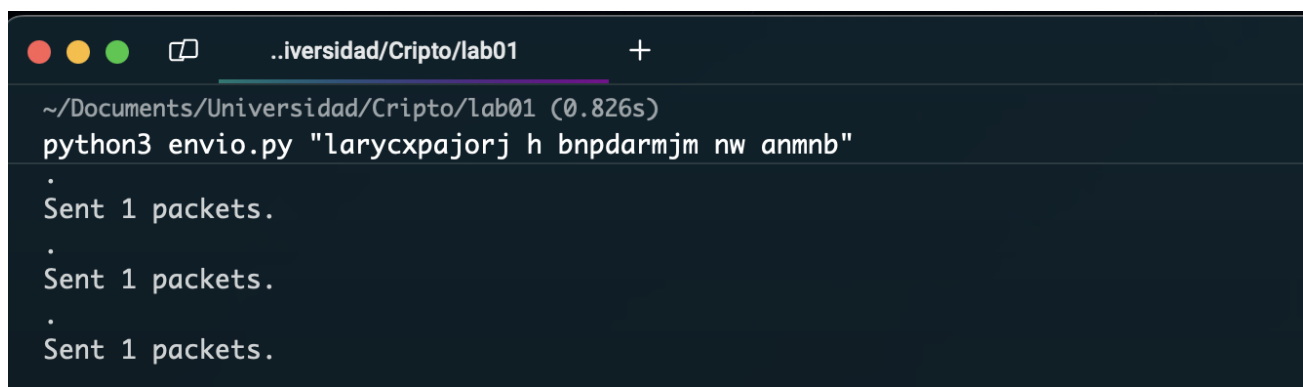
- Crea el contador con 1 Byte de carácter cifrado, 1 Byte de carácter random y 1 Byte de contador. (3 Bytes)
- Genera 5 Bytes de valor x00. (5 Bytes)
- Genera 40 Bytes desde el x10 al x37. (40 Bytes)

Así es como genera un campo data ICMP con 48 Bytes de tamaño, agregando el carácter cifrado.

Envío de paquetes

Luego de generar el valor en Bytes del campo data se precisaba enviar este valor en un paquete ICMP, como se menciono anteriormente se haría uso de Scapy para generar este tipo de trafico, pudiendo encontrar este envío de trafico en la función **enviar_icmp()** vista en la figura 3.2. Esta función netamente recibe como parámetros el valor del payload a enviar y el índice de la iteración en cada carácter, esto para generar el trafico de la siguiente manera:

- Se utiliza la función IP() para asignar origen y destino del paquete, en este caso se utilizo la IP de *loopback* en ambos parámetros.
- En la función ICMP() se asigna el type, identifier y sequence de cada paquete, siendo 8 para el tipo (ICMP request), 0x0001 para el id de los paquetes y el sequence number ira incrementando con cada iteración y envío.
- Finalmente mediante la función Raw() se envía el payload, en este caso el valor data generado anteriormente.



```
..iversidad/Cripto/lab01 +
~/Documents/Universidad/Cripto/lab01 (0.826s)
python3 envio.py "larycxpajorj h bnpdarmjm nw anmnb"
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

Figura 4: Envío paquetes ICMP con cifrado

Ya sabiendo como funcionan los algoritmos implementados, se da paso a ejecutar el código pasándole como argumento la frase ya cifrada. Como se puede observar en la imagen el código responde correctamente, enviando los paquetes por la red.

Captura con wireshark

Con el código cifrado y el algoritmo de envío de paquetes listos para ejecutar, se dio paso a ocupar la herramienta **Wireshark** para la captura y análisis del tráfico generado en la red, entre este tráfico se encontrarán los paquetes enviados con el código cifrado. Al poner a capturar el tráfico a Wireshark y luego enviar los paquetes se encontró con lo siguiente.

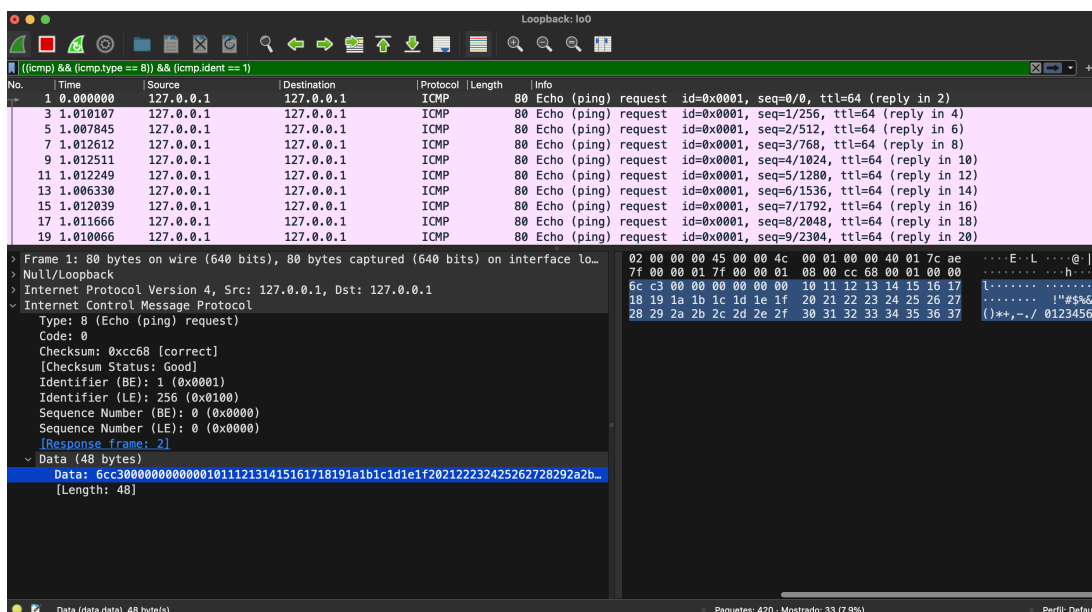


Figura 5: Captura de paquetes con Wireshark

Observando la captura del Wireshark, se puede apreciar que los paquetes ICMP se han enviado correctamente, se encuentra con todos sus campos requeridos para pasar desapercibido como cualquier otro ping ya que cuenta con su respectivo **type**, **identifier**, **checksum**, **sequence** y por supuesto el mas importante valor en este laboratorio, el campo **data**. Si prestamos atención al valor de data, se puede ver que cumple con lo que se envió desde el código en la figura 3.2, ya que el primer byte corresponde al carácter cifrado, los siguientes 2 bytes al resto del contador, luego los 5 bytes de valores x00 y finalmente los últimos 40 bytes que recorren desde el x10 al x37.

3.3. Actividad 3

Para la ultima actividad de laboratorio, se pedía leer la captura de wireshark mediante un programa en python, el cual fuese capaz mediante fuerza bruta descifrar el mensaje cifrado en la actividad 1. A continuación se muestra el código con los algoritmos implementados para cumplir los requerimientos de la actividad.

```

1  from scapy.all import *
2  from colorama import Fore
3
4  texto = ""
5
6  def decifrar_cesar(texto_cifrado, corrimiento):
7      texto_descifrado = ""
8      for char in texto_cifrado:
9          if char.isalpha():
10             if char.islower():
11                 nuevo_char = chr((ord(char) - ord('a') - corrimiento) % 26 + ord('a'))
12             else:
13                 nuevo_char = chr((ord(char) - ord('A') - corrimiento) % 26 + ord('A'))
14             texto_descifrado += nuevo_char
15         else:
16             texto_descifrado += char
17     return texto_descifrado
18
19 # Función para procesar los paquetes ICMP recibidos
20 def procesar_paquete(paquete):
21     global texto
22     if ICMP in paquete and paquete[ICMP].type == 8:
23         # Obtener el payload del paquete ICMP
24         payload = paquete[Raw].load.hex()
25
26         # Obtener el caracter cifrado
27         caracter_cifrado = payload[:2]
28
29         texto += chr(int(caracter_cifrado, 16))
30
31 if len(sys.argv) != 2:
32     sys.exit(1)
33
34 archivo_pcap = sys.argv[1]
35
36 # Cargar el archivo pcapng
37 paquetes = rdpcap(archivo_pcap)
38
39 # Iterar sobre cada paquete del archivo
40 for paquete in paquetes:
41     procesar_paquete(paquete)
42
43 for i in range(26):
44     cesar = decifrar_cesar(texto, i)
45     if (cesar == "criptografia y seguridad en redes"):
46         print(Fore.GREEN + f'{i}    {cesar}')
47     else:
48         print(Fore.WHITE + f'{i}    {cesar}')
```

De la figura 3.3 podemos notas que nuevamente se utilizó la herramienta Scapy, mediante sus múltiples funciones para leer archivos *.pcap* y analizar campos de un paquete ICMP. En primer lugar se lee el archivo *.pcap* que llega como argumento al programa, luego se itera por cada paquete capturado en Wireshark y enviando a la función **procesar_paquete()**,

la cual se asegura que el paquete corresponda a un ICMP request dado que es el tipo de paquete que necesitamos analizar y guardar los primero 2 bytes correspondientes al carácter cifrado, concatenándolos en una variable de texto para obtener la frase completa cifrada. Finalmente y dado que no se cuenta con el valor de corrimiento de cifrado, se procede a aplicar el algoritmo cesar, esta vez para descifrar los valores, probando todos los posibles desplazamientos que en este caso eran 26, obteniendo como resultado lo siguiente.

```
~/Documents/Universidad/Cripto/lab01 (0.813s)
python3 recibido.py "prueba_wire.pcapng"
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  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 sfef
9  criptografia y seguridad en redes
10 bqhoshnfqzehz x rdftqhczc dm qdcdr
11 apgnrmepdygy w qcespgbyb cl pcbbq
12 zofmqldoxcfx v pbdrofaxa bk obabp
13 ynelpkcnwbew u oacqnezvz 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 jwvkw
18 tizgkfxirwzr p jvxlizuru ve ivuvj
19 shyfjewhqvyq o iuwkhytqt ud hutui
20 rgxeidvgpuxp n htvjgxsp s tc gtsth
21 qfwdhcufo two m gsuifwror sb fsrsg
22 pevcbtensvn l frthevqnq ra erqrf
23 odubfasdmrum k eqsgdupmp qz dqpqe
24 nctaezrclqtl j dprfctolo py cpopd
25 mbszdyqbksk i coqebnskn ox bonoc
```

Figura 6: Respuesta con descifrado por fuerza bruta

Es así como por fuerza bruta se pudo obtener el código descifrado, teniendo un corrimiento de 9 y siendo la palabra **criptografia y seguridad en redes**, las cuales concuerdan con la figura 2.

Conclusiones y Comentarios

Luego de haber realizado todas las actividades del laboratorio se pudo observar que hubo un éxito en todas estas, pasando por un correcto cifrado mediante el algoritmo de cifrado *Cesar*, generación de paquetes ICMP con todos sus campos, captura de los paquetes mediante Wireshark y finalmente el descifrado del mensaje mediante fuerza bruta.

En este laboratorio se pueden concluir dos puntos importantes en el ámbito de la criptografía y la seguridad en redes. El primer punto corresponde a lo simple que es la comunicación en cuanto al envío y recepción de datos en la red, dado que solo generando trafico ICMP se pudo enviar un mensaje de 5 palabras con 29 letras en total, pudiendo ser incluso un mensaje aun mas largo dado que el programa *ping* es altamente utilizado dentro de la empresa a la cual se le hizo la auditoría.

El segundo punto a destacar es la poca seguridad que genera el algoritmo de cifrado cesar y la increíble facilidad con la que se puede realizar un ataque a este tipo de cifrado, a pesar de que el auditor ya sabia en que byte del campo data iba el mensaje cifrado, tan solo eran 3 los bytes que variaban en este campo, es decir, un atacante solo le seria necesario analizar el paquete ICMP e ir probando algoritmos de cifrado para poder encontrar el mensaje, es mas, en la misma auditoría se pudo demostrar que no era ni siquiera necesario conocer el desplazamiento utilizado ya que aplicando todos los posibles corrimientos se pudo obtener fácilmente el mensaje.

En conclusión, las actividades de laboratorio se realizaron con éxito, pero se pudo dar pruebas de las brechas de seguridad que existen en este tipo de casos.

Issues

Generación de código mediante IA

La primera complicación al desarrollar el laboratorio fue que al desconocer como implementar el algoritmo de cifrado cesar y generar trafico mediante python se utilizo chatGPT, siendo esta una herramienta útil ya que proporciona código en cosa de segundos. Lamentablemente el código que proporcionan este tipo de herramientas es bastante genérico, incluso habiendo veces que se negaba a realizar tareas por el tipo de prompt que se le escribía, dado que lo relacionaba con conceptos de hacking e información poco ética. Es por esto que a pesar de que si proporcionaba código útil en este caso, fue necesario la intervención humana para que funcionara de una manera correcta y óptima. Pudiendo reflejar así lo mucho que le falta por avanzar a las inteligencias artificiales.

Timestamp from icmp data

El mayor de los retos fue que Wireshark pudiese capturar el *Timestamp from icmp data* en los paquetes, dado que aun enviando este campo mediante el código en python, Wireshark no reflejaba esto en la captura. Esto finalmente no se pudo realizar a pesar de intentarlo con distintos mecanismos, reflejando una gran brecha en la seguridad de la actividad dado que los paquetes enviados con el cifrado eran los únicos que no contaban con este campo Timestamp.

Generación de payload

Un problema que se presento durante el desarrollo de la actividad 2 fue la creación del campo data a enviar en el paquete, dado que al principio se utilizo una variable que contenía un "data type" genérico con todo el campo data de un ICMP capturado del programa ping. Esto se pudo solucionar gracias a la función `byte()` que convertía los hex en bytes, concatenándolos en una variable llamada `icmp_data` la cual se podía enviar el payload perfecto para que no levantara sospecha entre el trafico ICMP.

Texto en verde

Para finalizar, la ultima de las complicaciones fue poner el texto descifrado en verde. Este problema surgió dado que generar un algoritmo que detectara palabras coherentes era bastante complicado, ya que se precisaba de crear un diccionario con palabras recurrentes en el español, cosa que se haría casi imposible por lo extenso que seria. Esto se pudo solucionar provisoriamente gracias a que se conocía la palabra a descifrar, teniendo que poner solamente un condicional para imprimir en verde cuando encuentre la frase igual a la que se esperaba obtener, siendo esta una solución poco óptima pero para méritos del informe, mas que suficiente.