

Informe Laboratorio 1

Sección 1

Omar Javier Marca Perez
omar.marca@mail.udp.cl

Marzo de 2024

Índice

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

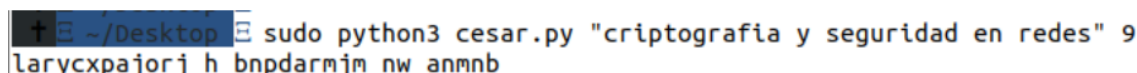
A usted le han encomendado auditar si efectivamente estos sistemas son capaces de detectar las filtraciones a través de tráfico de red. Debido a que el programa ping es ampliamente utilizado desde dentro y hacia fuera de la empresa, su tarea será crear un software que permita replicar tráfico generado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

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.



```

$ sudo python3 cesar.py "criptografia y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb

```

El algoritmo de cifrado cesar esta basado en el ID de los caracteres correspondientes al codigo ASCII desde el 97(a) hasta el 122(z) y es el siguiente:

```

1  import sys
2
3  # python3 cesar.py "criptografia y seguridad en redes" 9
4
5  message = sys.argv[1]  # Mensaje a cifrar
6  n = int(sys.argv[2])
7
8  EncryptedMessage = ""
9  # Recorre cada caracter y le aplica la traslación
10 for c in message:
11     if ord(c) + n >= 97 and ord(c) + n <= 122:
12         EncryptedMessage += chr(ord(c) + n)
13     elif " " == c:
14         EncryptedMessage += " "
15     else:
16         EncryptedMessage += chr(ord(c) + n - 26)
17
18 print(EncryptedMessage) #Imprime el mensaje cifrado

```

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.

A continuación se muestra el código generativo de paquetes ICMP:

```

1  from ping3 import ping
2  from scapy.all import *
3  import time
4  import sys
5  import struct
6
7  # python3 pingv4.py "larycxpajorj h bnpdarmjm nw anmnb"
8
9  EncryptedMessage = sys.argv[1]
10
11  i = 0
12  while i < len(EncryptedMessage):
13      character = format(ord(EncryptedMessage[i]), '02X') # Se obtiene el caracter y luego se inserta en el Data. Modelo tipo linux:
14      Data = bytes.fromhex(character + "608f000000000101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637")
15
16      # Timestamp
17      timestamp = int(time.time()) # Obtenemos el tiempo actual para asignarlo al timestamp
18      timestampBytes = struct.pack('<Q', timestamp) # Lo pasamos a Byte para poder enviarlo y lo ajustamos a <Q
19      #
20      icmp_packet = IP(dst="127.0.0.1") / ICMP(type=8, code=0, id= 4160, seq=1 + i) # Creamos el paquete ICMP
21      #
22      icmp_packet = icmp_packet / timestampBytes / Data # Le anadimos el Timestamp y el valor del campo Data
23
24      send(icmp_packet) # Enviamos un paquete cada segundo
25      time.sleep(1)
26      i += 1

```

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.

El código usado para descifrar el mensaje es el siguiente:

```

1  from scapy.all import *
2  import sys
3
4  # python3 readv2.py cesar.pcapng
5
6  def Decoder(message, n):    # Decifra el mensaje
7      EncryptedMessage = ""  # Segun n traslacion
8      for c in message:
9          if ord(c) - n >= 97 and ord(c) - n <= 122:
10             EncryptedMessage += chr(ord(c) - n)
11         elif " " == c:
12             EncryptedMessage += " "
13         else:
14             EncryptedMessage += chr(ord(c) - n + 26)
15
16     return EncryptedMessage
17
18 # Ruta al archivo pcapng
19 archivo_pcapng = "./" + sys.argv[1]
20
21 # Lee el archivo pcapng
22 paquetes = rdpcap(archivo_pcapng)
23
24 EncryptedMessage = ""
25
26 for paquete in paquetes:
27     if ICMP in paquete: # Verifica si es un paquete ICMP
28         data = paquete[ICMP].payload.load
29         data = data[8:] # Omitir los primeros 8 bytes
30         EncryptedMessage += chr(data[0])
31
32
33 i = 0
34 while i < 26: # Imprime todas las posibles traslaciones
35     print(str(i) + " " + Decoder(EncryptedMessage,i))
36     i+=1

```

Este código es muy similar al de cesar.py pero con la diferencia que tiene que iterar en

todas las combinaciones posibles para encontrar el mensaje original.

3. Desarrollo de Actividades

3.1. Actividad 1

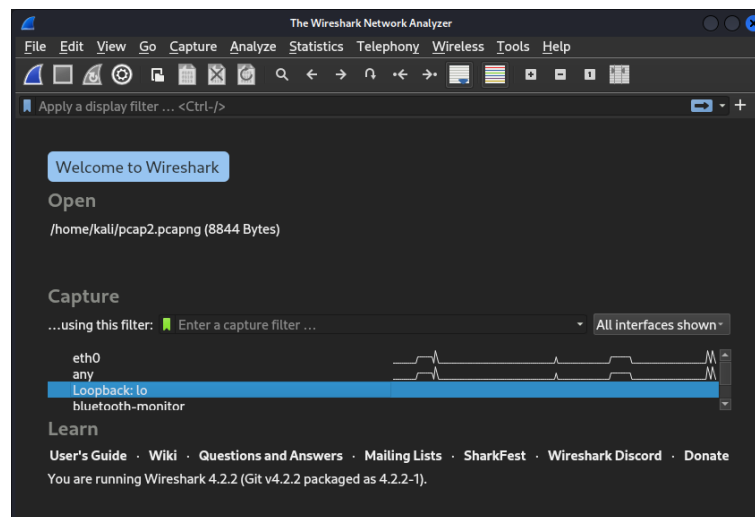
A continuación se muestra el comando que ejecuta el código del algoritmo cesar. Este comando tiene como parámetros el texto que se quiere cifrar y la traslación que se le quiere aplicar.

```
(kali@kali)-[~/Laboratorio1]
$ sudo python3 cesar.py "criptografia y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb
```

Como salida se obtiene una cadena de caracteres con el código cifrado según la traslación aplicada.

3.2. Actividad 2

Antes de iniciar la captura de paquetes, se debe ajustar hacia la interfaz de Loopback de Wireshark para captar los paquetes de nuestro entorno de prueba como se muestra a continuación:



Luego se ejecuta el comando para inyectar tráfico a la red. Este debe ingresar como parámetro el texto cifrado obtenido anteriormente:

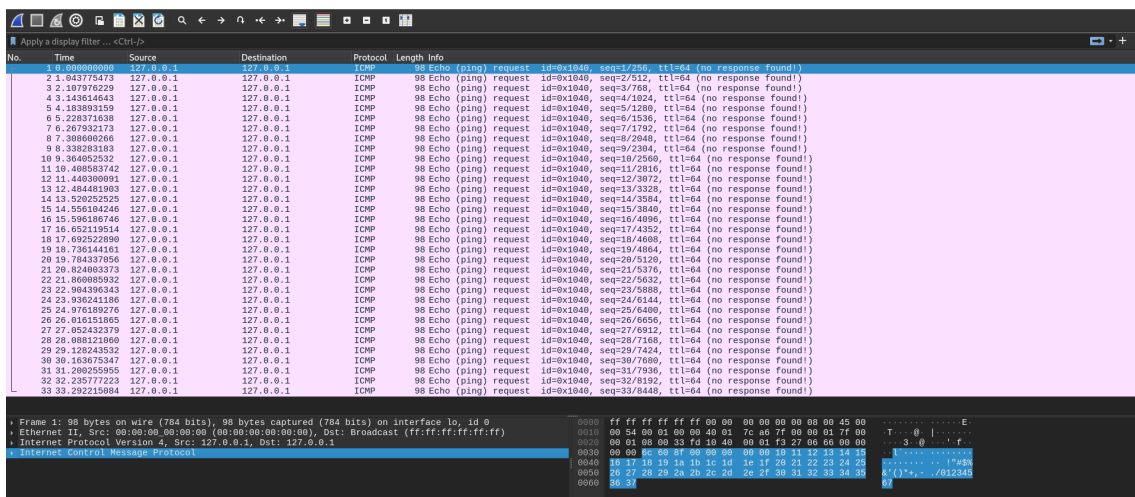
```
(kali㉿kali)-[~/Laboratorio1]
$ sudo python3 pingv4.py "larycxpajorj h bnpdarmjm nw anmnb"

Sent 1 packets.

Sent 1 packets.

Sent 1 packets.
```

Ahora se pueden apreciar los paquetes ICMP generados en Wireshark:



Para una mayor credibilidad, se debe dar un vistazo al apartado de ICMP asegurandose de que todos los elementos esten en su lugar. Los campos a revisar son:

- Type
- Code
- Checksum
- Identifier
- Sequence Number
- Timestamp
- Data

A continuación se muestra una imagen ampliada de Wireshark con dichos campos:

```
▼ Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x33fd [correct]
  [Checksum Status: Good]
  Identifier (BE): 4160 (0x1040)
  Identifier (LE): 16400 (0x4010)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  [No response seen]
  Timestamp from icmp data: Mar 28, 2024 22:31:15.000000000 EDT
  [Timestamp from icmp data (relative): 0.541009521 seconds]
  ▼ Data (48 bytes)
    Data: 6c608f0000000000101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f30313233343...
    [Length: 48]
```

El identificador y la secuencia son usados para asociar cada paquete Request a uno Reply. En este caso se escoge un número como lo es 4160 de identificador siendo que pueda pasar desapercibido entre otros paquetes con identificadores de similar magnitud. A su vez se toma en consideración que como estos paquetes corresponden a un mismo identificador, estos deben estar secuenciados para que no hayan sospechas que estos paquetes actúen con irregularidad a los demás. Es decir, los paquetes generados se deben comportar de la misma forma a como lo hacen los paquetes ICMP generados naturalmente por el comando ping. Por otra parte, el Checksum o comprobación de datos y errores es generado de forma automática por Scapy debido a la forma en la que está hecho el código de Python.

El Timestamp corresponde al tiempo exacto en el que se creó el paquete. No obstante es importante señalar que normalmente estos también se manejan en milisegundos pero por la naturaleza de la librería Struct de Python al trabajar en bytes, no se generan paquetes con decimales y de 16 bytes por lo que estos paquetes se mostrarán como un número entero correspondiendo en este caso a la creación del paquete.

Luego tenemos el campo de Data que por lo general contiene información que no es útil ya que dependerá su contenido del sistema operativo. No obstante, es la que se utiliza para enviar el mensaje cifrado mediante caracteres insertados en el byte menos significativo.

3.3. Actividad 3

Una vez tenemos el archivo pcapng generado por Wireshark, pasamos el nombre del archivo como parámetro en el código de descifrado como se muestra a continuación:

```

(kali㉿kali)-[~/Laboratorio1]
$ 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 wikyvmheh ir vihiw
6 fulswrjudild b vhjxulgdg hq uhghv
7 etkrvqitchkc a ugiwtkfcf gp tgfgu
8 dsjquphsbgjb z tfhvsjebe fo sfteft
9 criptografia y seguridad en redes
10 bqhosnfqzehz x rdftqhczc dm qdcdr
11 apgnrmepydgy w qcespgbyb cl pcbcbq
12 zofmqldoxcfx v pbdrofaxa bk obabp
13 ynelpkcnwbew u oacqnezww aj nazao
14 xmdkojbmadv t nzbpmdivy zi mzyzn
15 wlcjniauzcu 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 qfwdhcufotwo m gsuifwrwr sb fsrsg
22 pevcbtensvn l frthevqnq ra erqrf
23 odubfasdmrum k eqsgdupmp qz dqpqe
24 nctaezrclqtl j dprfctolo py cpopd
25 mbszdyqbksk i coqebnkn ox bonoc

```

Como se puede apreciar, se hacen varias iteraciones con distintas traslaciones de letras para poder encontrar el mensaje descifrado. El mensaje descifrado está marcado en color verde y la traslación es de 9 unidades.

Conclusiones y Comentarios

En la experiencia se destaca el manejo de Python junto con Scapy para la generación de paquetes de protocolo ICMP con la finalidad de enviar mensajes cifrados en una secuencia de paquetes. Luego, a partir de otro programa hecho en Python se lee el archivo pcapng para analizar los paquetes que se enviaron y poder descifrar el mensaje aplicando todas las traslaciones posibles del mismo y conocer el contenido original. Otro punto a tomar en consideración es que los paquetes generados en la interfaz de Loopback no reciben ningún tipo de respuesta a estos, por lo que es normal que aparezca el apartado de "No response seen". Los principales postulados a este suceso es que el Timestamp no está alineado correctamente

con el tiempo de salida del paquete, por lo que pudo haber sido ignorado por la misma máquina.

Por otra parte, en esta experiencia se profundizó más en el uso de los paquetes y sus apartados así como en los tamaños en bytes que se manejan entre cada elemento del mismo.

Issues

1) Uno de los problemas que hubo fue la implementación del Timestamp en el código puesto a que este apartado tiene 16 bytes. El Struct con el cual se pasa a bytes el tiempo actual solo soporta 8 bytes, por lo que fue necesario hacer un recorte. No obstante, estos bytes perdidos no fueron significativos puesto a que se trataba de decimales. Una de las soluciones a esto es llenar con ceros para ocupar el espacio de dichos decimales para tener coherencia en el tamaño del paquete.

2) El Timestamp generaba problemas a la hora de extraer información del campo Data. Cuando se extraía información, estaba en formato byte y además, el Timestamp estaba junto con la información Data haciendo que fuese difícil extraer el carácter deseado. Para resolverlo, se implementó en el código una forma de eliminar los primeros 8 bytes de la cadena que llegaba del campo Data. Es decir, se eliminaban bytes hasta que en la primera posición quedaba el carácter deseado listo para analizar.

3) Hubo confusiones entre lo que mostraba el Wireshark en el campo data y lo que se mostraba en el código de Python. Cuando se quería insertar los valores del campo Data en el paquete, estos solían mostrarse como texto plano en Wireshark, por lo que era necesario pasarlos a bytes siendo ese el formato correcto. De la misma forma para la extracción de los datos resultó confuso. Para solucionarlo se diseñaron ciertas reglas de como debían llegar los datos desde el código hasta Wireshark y viceversa. Las reglas antes mencionadas están basadas en el tipo de datos que maneja cada aplicación ya sea String, Byte, Char o Hexadecimal.

4) No hay mucha información accesible sobre el paquete ICMP y las que hay no siempre cuentan con toda la información al respecto. Por ejemplo, es bien sabido que el comando ping, dependiendo del sistema operativo en el que se ejecute dará distintos resultados en el protocolo ICMP. En Windows el paquete ICMP pesa 74 bytes por defecto mientras que en Ubuntu pesa 98 bytes. Esto deja aun más compleja la búsqueda de información debido a que para distinto software requiere de distinta documentación. La única forma encontrada de solucionarlo es a base de prueba y error. Mientras que en el campo Data de Windows se envía el abecedario con repetición, en Linux se envía una secuencia de números del 10 hasta el 37.

Bibliografía

El apoyo y manejo del código se hizo con una combinación entre el uso de herramientas de IA's de código generativas como ChatGPT y Microsoft Bing y a su vez de documentación

formal dada por la pagina oficial de Scapy, Wireshark y Digital Guide Ionos.

Página oficial de Scapy [acá](#)

Página oficial de Ionos sobre el paquete ICMP [acá](#)

Página oficial de Wireshark sobre el Timestamp del paquete ICMP [acá](#)

Codigo cesar.py [acá](#)

Codigo pinv4.py [acá](#)

Codigo readv2.py [acá](#)