Universidad Carlos III

Ingeniería de la Ciberseguridad

Curso 2024-25

Práctica 2

Intercepción de tráfico HTTP

Ingeniería Informática, Cuarto curso

Adrián Fernández Galán (NIA: 100472182, e-mail: 100472182@alumnos.uc3m.es)
César López Mantecón (NIA: 100472092, e-mail: 100472092@alumnos.uc3m.es)
Manuel Gómez-Plana Rodríguez (NIA: 100472310, e-mail: 100472310@alumnos.uc3m.es)

Prof. Antonio Nappa
Grupo: 81

1. Introducción

Este documento recoge el desarrollo de la segunda práctica de la asignatura *Ingeniería de la Ciberseguridad*. El objetivo de esta práctica es desencriptar una serie de *flags* contenidas en paquetes `http` que atraviesan un *proxy*.

Exiten dos clases de *flags* contenidas en los paquetes:

- **Tipo 1**: se trata de una cadena hexadecimal con un argumento llamado *rotation*.
- **Tipo 2**: se trata de una cadena de caracteres con 3 argumentos (*Rotation, s* y *op*)

2. Funcionamiento de `mitmproxy`

`Mitmproxy` es una herramienta para la consola de comandos que permite el análisis interactivo y la modificaicón de tráfico `http`. Por defecto, el proxy escucha sobre la dirección *localhost* y puerto *8080*.

Adicionalmente, existe un proceso en constante ejecución llamado *down.sh* que hace una petición http sobre esta misma dirección.

```
   PID TTY          TIME CMD
     1 pts/0    00:00:00 bash
   587 ?        00:00:00 sshd
 26023 ?        00:00:37 down.sh
900981 ?        00:00:00 sshd
900998 ?        00:00:00 sshd
900999 pts/1    00:00:00 bash
901270 ?        00:00:00 sleep
901273 pts/1    00:00:00 ps
```

Listing 1: Resultado de ejecutar ps -e

Se ha utilizado el comando `find` para localizar el archivo. Aprovechando el conocimiento previo que tenemos sobre la máquina, hemos empleado el usuario *admin* para poder lanzar el comando sobre la raíz, ya que este usuario pertenece al grupo de superusuarios:

```
sudo find / -type f -name down.sh
[sudo] password for admin:
./down.sh
```

Listing 2: Resultado de ejecutar find

Finalmente, analizando el contenido del archivo podemos ver el comando utilizado para realizar la petición `http`.

```
    /home/rover/curl-7.88.1/src/curl --insecure --proxy-insecure --
       proxy https://172.17.0.19:8080 https://172.17.0.4:4443/
```

Listing 3: petición http a través del comando curl

Se observa que el comando redirige su petición a la ip y puerto donde escucha `mitmproxy`. Adicionalmente, utiliza las flags `-insecure` y `-proxy-insecure`, por lo que no verificará los certificados del proxy y se permitirá la conexión. Con esto, la conexión se completa sin levantar excepciones y podremos ser capaces de analizar las peticiones y respuestas a través de la interfaz de `mitmproxy`.

3. Descubrimiento de una flag tipo 1

Las *flags* están codificadas en hexadecimal y cuentan con un parámetro llamado *Rotation*. Este parámetro parece indicar que se ha cifrado a través de una rotación en la cadena de texto. Al analizar varios paquetes se aprecia que las flags de igual rotación cuentan con caracteres entre los índices 10 y 15, lo que sugiere la utilización de una misma semilla para todas las flags.

```
3193601502dbafcc86bdb57381081aec
3846408208dbafccc0399a91c4fbf1a3
4745131622dbafcc09340e7b0af6885f
```

Listing 4: Varias flags para rotation 1 donde la semilla es dbafcc

Primeramente se trató de desencriptar las *flags* rotando en ambos sentidos los valores hexadecimales de cada caracter. No obstante, esto no dio resultado. Adicionalmente, se probaron otras codificaciones como `ASCII`.

Posteriormente se observó que las flags con una diferencia de 6 rotaciones mantenían la misma semilla. Dado que la semilla está formada enteramente por letras y que la codificación hexadecimal cuenta con 6 caracteres alfabéticos, esto sugiere que el cifrado se ha aplicado por separado a letras y números.

```
FLAG                                Rot
3193601502dbafcc86bdb57381081aec     1
3846408208dbafccc0399a91c4fbf1a3     1
4745131622dbafcc09340e7b0af6885f     1
9352846549dbafcc965695c25417c6d9     7
7173993205dbafccf2c4f4db398e6687     7
1047974726dbafcc05aacde26d253963     7
7512913585dbafcc9aeb0520366299c2     7
5483152252dbafcc26492c301ca7365e     7
2473284585dbafcc99892775f126bd30    13
0872242899dbafccfc622e537f7ba143    13
9735635169dbafcc24d5a92548bc1e09    13
7646156765dbafccd0beba631d833c5c    13
```

Listing 5: Varias flag con la misma semilla y 6 rotaciones

Finalmente, se ha hecho la prueba obteniendo una *flag* válida a través de un programa de python.

```python
flag = "9443878358dbafcc223a8e670ba03537"
num_rotations = 1
letters_to_number = {'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4, 'f':
    5}
number_to_letters = {0: 'a', 1: 'b', 2: 'c', 3: 'd', 4: 'e', 5: 'f'
    }
new_flag = ""

for i in range(len(flag)):
    c = flag[i]
    if c.isnumeric():
        new_c = (int(c) - num_rotations) % 10
    elif c in letters_to_number.keys():
        new_c = number_to_letters[(letters_to_number[c] -
            num_rotations) % 6]
    else:
        new_c = c
    new_flag += str(new_c)

print(new_flag)
```

Listing 6: Decodificación de una flag de rotación 1

El resultado obtenido ha sido el siguiente: `8332767247cafebb112f7d569af92426`. Este resultado si ha sido aceptado como correcto. La semilla empleada en la generación de las *flags* es `cafebb`.

A modo de curiosidad, dado que el cifrado de rotación es modular, la semilla se mostrará en claro para una rotación igual al periodo. Es por esto que se puede apreciar la semilla en las flags con rotación 6:

```
3786566793 cafebb39330ab797fbecfa
6350772345 cafebb35f756efb7386a9d
0706541085 cafebbeaf7963399a4d327
2847807583 cafebbd1a054ded7579a71
1613995163 cafebb528dd12a5f194769
```

Listing 7: Flags para rotacion 6

4. Descubrimiento de una flag tipo 2

Las *flags* de tipo 2 cuentan con un cifrado más complejo que requiere de varios pasos para su descrubrimiento. Los paquetes cuentan con 3 argumentos (*Rotation, op* y *s*) y una cadena de caracteres codificada en base 64.

```
Hexadecimal Flag (Rotation: 15, s:1c7767, op:x): b'
    BlAEAAAGBVUHDlBTUgFNTQEEAwIAUwYHAQVVBlcDAFc='
```

Listing 8: Ejemplo de flag tipo 2

Los argumentos se han interpretado de la siguiente forma:

- **Rotation**: al igual que en las flags de tipo 1, se asume que indica la rotación en uno o varios diccionarios de caracteres.
- **s**: se interpreta como una semilla o contraseña usada en el cifrado.
- **op**: se interpreta como la operación usada en el cifrado. Dado que tiene un valor constante `x`, se asume que se trata de una operación `XOR` con la semilla, de forma similar a un *cifrado Vernam*.

Para descubrir la *flag* se han seguido los pasos detallados a continuación.

4.1. Exploración de la máquina y conversión a hexadecimal

Al explorar la máquina se ha encontrado un directorio oculto en la raíz llamado `.ingsec2425` con un archivo `xor.c`. Este archivo cuenta con una función para aplicar una operación `XOR` sobre una cadena en hexadecimal y una contraseña. Con esta información, se ha codificado la *flag* en hexadecimal y se ha aplicado esta función sobre ella y la semilla, obteniendo el siguiente resultado:

```
    echo BlAEAAAGBVUHDlBTUgFNTQEEAwIAUwYHAQVVBlcDAFc= | base64 -d |
        xxd -p
    > 0650040000060555070
        e505352014d4d0104030200530607010555065703005 7 # flag en hex

    /ingsec2425/xor lc7767 0650040000060555070
        e505352014d4d0104030200530607010555065703005 7
    > 373333373631343630396664636 27
        a7a373332613764303030666231613431 34
```

Listing 9: XOR sobre la flag

Es importante destacar que para este proceso hemos usado el usuario *admin* para modificar los permisos del directorio y el ejecutable y poder lanzarlo desde el usuario *rover*.

4.2. Rotación

Primeramente se probó a hacer la rotación descubierta en las *flags* de tipo 1 sobre el resultado de la operación XOR. Esto no resultó en una *flag* correcta.

Analizando más profundamente el resultado, la salida de la operación XOR cuenta con 64 caracteres, mientras que las flags de tipo 1 cuentan con 32 caracteres. Sabiendo que la cadena está codificada en hexadecimal, se transformó en `ASCII` para lograr una cadena con la longitud final

esperada. Sobre la nueva cadena se volvió a aplicar la rotación, obteniendo el resultado final. Se ha usado el código de la parte anterior junto con el siguiente comando de linux para decodificar finalmente la flag:

```
echo "373333373631343630396664636 27
    a7a373332613764303030666231613431 34" | xxd -r -p
> 7337614609fdcbzz732a7d000fb1a414

python3 decode_flag.py 7337614609fdcbzz732a7d000fb1a414
> 6226503598ecbazz621f6c999ea0f303
```

Listing 10: Optención de la flag en claro

4.3. Pipeline de decodificación

Se incluye este apartado a modo de resumen del proceso seguido en la decodificación de las *flags* tipo 2.

1. Traducción de la *flag* de base 64 a hexadecimal.
2. Operación XOR entre *flag* y semilla.
3. Codificación en `ASCII` del resultado anterior.
4. Aplicación de la rotación descubierta para las *flags* tipo 1.

5. Conclusión

El análisis de tráfico de red es una de las principales actividades relacionadas con la ciberseguridad. La importancia de unas comunicaciones seguras a través de la red es vital no sólo para la protección de información crítica, sino también para el respecto de los derechos a la privacidad de los usuarios. Esta práctica nos ha permitido explorar el análisis e intercepción de tráfico web y familiarizarnos con herramientas como `mitmproxy`.

Trabajar con dos clases de *flags* nos ha permitido experimentar como un cifrado algo más complejo multiplica en gran medida el esfuerzo necesario para la vulneración de la confidencialidad de un mensaje. Las flags de tipo 2 han requerido un esfuerzo mucho mayor para descubrir su descifrado. Al igual que vimos en cursos pasados, es importante que los algoritmos criptográficos reduzcan al máximo la información contenida en un mensaje cifrado para impedir el reconocimiento de patrones que le permitan al atacante descubrir su contenido. Sin embargo, en ambos tipos de *flags* ha sido posible obtener información sobre el cifrado a través del análisis de distintas ocurrencias.

Para solucionar las vulnerabilidades existentes en el contexto de la práctica es necesario evitar el uso de marcadores como `-ssl-insecure` o `-proxy-insecure` que permiten la comunicación con proxys no autenticados. Adicionalmente, se sugiere usar algoritmos de cifrado probados y estándar como AES256.