Ingeniería de la Ciberseguridad G81

PRÁCTICA 2: TLS intercept

30-Noviembre-2024
Prof: Antonio Nappa
Nombre del grupo: **GrupoAJA**

Alejandro Isla Álvarez (**100472228**)
József Iván Gafo(**100456709**)
Ángel Pérez Navas (**100472200**)

Como hemos conseguido las flags

Para obtener las flags codificadas es necesario interceptar las respuesta a las peticiones HTTP que se realizan de forma continuada durante la ejecución del fichero *down.sh*. Para ello se utiliza mitmproxy con la opción *--ssl-insecure* para que no valide el certificado TLS y nos permita inspeccionar el tráfico. Para automatizar el proceso de decodificación de la flag final escribimos un script en python que se ejecuta para procesar cada respuesta recibida usando *mitmdump*.
A continuación, se describe el proceso llevado a cabo para obtener las flags finales a partir de la respuesta HTTP de las peticiones curl.

Flag type 1

```python
def rotate_hex_char(c, n):
    if c.isdigit():  # Comprobamos si es un número (0-9)
        new_val = (int(c) - n) % 10
        return str(new_val)
    elif 'a' <= c <= 'f':  # Comprobamos si es una letra hexadecimal (a-f)
        new_val = (ord(c) - ord('a') - n) % 6 + ord('a')
        return chr(new_val)
    else:
        return c  # Si no es un carácter válido, se deja igual

def rotate_hex_string(hex_string, n):
    rotated = ''.join(rotate_hex_char(c, n) for c in hex_string)
    return rotated

def response(flow: http.HTTPFlow):
    # Verificamos si la respuesta tiene el texto que buscamos
    if "Hexadecimal Flag" in flow.response.text:
        # Extraemos la línea con la flag
        lines = flow.response.text.splitlines()
        for line in lines:
            if "Hexadecimal Flag" in line and "s:" not in line and "op:" not in line:
                print("-----------------------------------------------------------------")
                # Parseamos la rotación y el valor hexadecimal
                parts = line.split(":")
                rotation = int(parts[1].strip())  # Extraemos rotación
                hex_value = parts[2].strip(" )")  # Extraemos flag en hexadecimal

                # Aplicamos la rotación
                decoded_flag = rotate_hex_string(hex_value, rotation)

                # Imprimimos la flag descifrada en formato hexadecimal
                print(f"Decrypted Flag Type 1: {decoded_flag}")
            print("-----------------------------------------------------------------")
```

Código para obtener la flag de tipo 1.

Con mitmproxy obtenemos el primer challenge del flag que consiste en rotar el ciphertext.

En nuestro programa, hemos decidido automatizar para que, usando mitmdump, el programa coja la respuesta del mitmproxy y aplique la rotación. La rotación se consigue usando una función llamada *rotate_hex_string* que tiene como parámetros el **string hexadecimal** (*hex_string*) y el **número de rotaciones a aplicar** (*n*).

En la función, aplicamos un **join** qué irá añadiendo carácter a carácter del hex_string y a la vez aplicando la función *rotate_hex_char*, en el cual hacemos la rotación del carácter hexadecimal.

La función *rotate_hex_char*,primero hacemos un check si el carácter es un número del 0 al 9, y hacemos una **resta al número de rotaciones** y aplicamos el módulo 10 para tener el resultado de la rotación (en este tipo de rotaciones cuando el carácter es del 0 al 9 no tenemos cuenta los números hexadecimales del a al f). Si el carácter es del a al f, usamos una función llamada **ord** que está encargada de **convertir el carácter al valor numérico de ASCII** (e.g “a” sería el valor 97).

En este condicional, primero obtenemos el valor numérico ASCII del carácter y le restamos “a” que seria 97, a continuación le **restamos el número de rotaciones** y **aplicamos** el **módulo 6** (ya que solo hay 6 caracteres alfabéticos en los números hexadecimales “*abcdef*”) y **finalmente** le **volvemos a sumar** “a” es decir **97** para obtener el carácter hexadecimal después de la rotación y lo convertimos de vuelta en carácter.
Y si el **carácter del parámetro es erróneo** o no válido **retornamos** el **mismo carácter** para evitar fallos.

Y al aplicar todas estas funciones, el challenge obtenido por el mitmproxy podemos obtener la flag y poder subirlo a seceng24.ctfd.io y verificar que es correcta.

Flag type 2

```python
def xor_with_seed(data, seed_str):
    # Realizamos la operación XOR entre data y la semilla.
    seed_bytes = seed_str.encode()
    result = bytes(data[i] ^ seed_bytes[i % len(seed_bytes)] for i in range(len(data)))
    return result

def response(flow: http.HTTPFlow):
    ...
          elif "Hexadecimal Flag" in line:
                # Flag tipo 2
                print("----------------------------------------------------------------")
                # Expresión regular para extraer los valores
                pattern = r"Rotation: (\d+), s:([0-9a-f]+), op:[^:]+: b'([A-Za-z0-9+/=]+)'"

                # Buscamos los valores
                match = re.search(pattern, line)

                rotation = int(match.group(1))  # Rotación
                seed = match.group(2)           # Semilla s
                base64_value = match.group(3)   # Valor en base64

                # Decodificamos la flag base64 a binario
                flag_bin = base64.b64decode(base64_value)
                # Realizamos la operación XOR con la semilla
                xor_result = xor_with_seed(flag_bin, seed).decode()
                # Aplicamos la rotación
                rotated_flag = rotate_hex_string(xor_result, rotation)

                print("Decrypted Flag Type 2:", rotated_flag)

                print("----------------------------------------------------------------")
```

Código parcial para obtener la flag de tipo 2.

Si el mitmproxy obtiene parámetros como puede ser el **Rotation**, **"s" de seed**, **op de operación y el challenge**, estamos ante la flag de tipo 2.

Para obtener la flag del challenge de tipo 2, tenemos primero que decodificarlo en **base64**, después aplicar la función *xor_with_seed*. En esta funcion, primero **codificamos la seed en binario para** permitirnos **hacer** la operación ***XOR*** más adelante. Es muy importante interpretar la **semilla como** una **cadena** en **ASCII** y no hexadecimal, para que el **resultado** del XOR sea una **cadena ASCII** interpretable **de 32 caracteres**.

A continuación, vamos **iterando byte por byte** del challenge, haciendo el ***XOR*** (usando el operador "^") con la semilla seed, donde seleccionamos el byte a hacer el xor. Una vez acabamos con toda la semilla, volvemos a repetirla desde el principio (esto se hace a través del módulo de la longitud de la semilla).

Una vez que ya hemos aplicado la operación ***XOR***, aplicamos la **función *rotate_hex_string*** que ya fue explicado en la flag de tipo 1 para obtener la flag del challenge de tipo 2.

Y finalmente lo podemos subir en *seceng24.ctfd.io* para verificar que la flag es correcta.

Ejemplos de resultados

Aqui podemos ver como nuestro programa coje la respuesta HTTP y automáticamente hace el challenge y nos permite obtener la flag.

```
172.17.0.14:38902: GET https://172.17.0.4:4443/ HTTP/1.1
       << HTTP/1.0 200 OK 218b
[16:37:33.122][172.17.0.14:38902] server disconnect 172.17.0.4:4443
[16:37:33.124][172.17.0.14:38902] client disconnect
[16:37:42.099][172.17.0.14:51988] client connect
[16:37:42.113][172.17.0.14:51988] server connect 172.17.0.4:4443
----------------------------------------------------------------
Decrypted Flag Type 1: 4901744065cafebba4182b22976a1545
----------------------------------------------------------------
172.17.0.14:51988: GET https://172.17.0.4:4443/ HTTP/1.1
       << HTTP/1.0 200 OK 187b
[16:37:42.145][172.17.0.14:51988] server disconnect 172.17.0.4:4443
[16:37:42.146][172.17.0.14:51988] client disconnect
[16:37:43.161][172.17.0.14:51994] client connect
[16:37:43.176][172.17.0.14:51994] server connect 172.17.0.4:4443
----------------------------------------------------------------
Decrypted Flag Type 1: 8355590895cafebb4891390c39442b88
----------------------------------------------------------------
172.17.0.14:51994: GET https://172.17.0.4:4443/ HTTP/1.1
       << HTTP/1.0 200 OK 187b
[16:37:43.207][172.17.0.14:51994] server disconnect 172.17.0.4:4443
[16:37:43.209][172.17.0.14:51994] client disconnect
[16:37:52.181][172.17.0.14:57180] client connect
[16:37:52.194][172.17.0.14:57180] server connect 172.17.0.4:4443
----------------------------------------------------------------
Decrypted Flag Type 2: 5814600756cafezz82903b243286fd39
----------------------------------------------------------------
172.17.0.14:57180: GET https://172.17.0.4:4443/ HTTP/1.1
       << HTTP/1.0 200 OK 218b
[16:37:52.227][172.17.0.14:57180] server disconnect 172.17.0.4:4443
[16:37:52.228][172.17.0.14:57180] client disconnect
[16:37:53.242][172.17.0.14:57184] client connect
[16:37:53.255][172.17.0.14:57184] server connect 172.17.0.4:4443
----------------------------------------------------------------
Decrypted Flag Type 2: 9450902089cafezze4a40f78cb17e9ce
----------------------------------------------------------------
```

Anexo: código completo

Se incluye el **código completo** para facilitar la reproducción de resultados usando mitmdump si fuera necesario.

Al ejecutar este código en la terminal (*./mitmdump --ssl-insecure -s decrypt_flags.py*), los paquetes irán apareciendo y gracias al programa que hemos diseñado, seremos capaces de descifrar automáticamente las flags a medida que estos van apareciendo.

```python
from mitmproxy import http
import re
import base64

def rotate_hex_char(c, n):
    if c.isdigit():  # Comprobamos si es un número (0-9)
        new_val = (int(c) - n) % 10
        return str(new_val)
    elif 'a' <= c <= 'f':  # Comprobamos si es una letra hexadecimal (a-f)
        new_val = (ord(c) - ord('a') - n) % 6 + ord('a')
        return chr(new_val)
    else:
        return c  # Si no es un carácter válido, se deja igual

def rotate_hex_string(hex_string, n):
    rotated = ''.join(rotate_hex_char(c, n) for c in hex_string)
    return rotated

def xor_with_seed(data, seed_str):
    # Realizamos la operación XOR entre data y la semilla.
    seed_bytes = seed_str.encode()
    result = bytes(data[i] ^ seed_bytes[i % len(seed_bytes)] for i in range(len(data)))
    return result

def response(flow: http.HTTPFlow):
    # Verificamos si la respuesta tiene el texto que buscamos
    if "Hexadecimal Flag" in flow.response.text:
        # Extraer la línea con la flag
        lines = flow.response.text.splitlines()
        for line in lines:
            if "Hexadecimal Flag" in line and "s:" not in line and "op:" not in line:
                # Flag tipo 1
                print("----------------------------------------------------------------")
                # Parseamos la rotación y el valor hexadecimal
                parts = line.split(":")
                rotation = int(parts[1].strip())  # Extraer rotación
                hex_value = parts[2].strip(" )")  # Extraer flag en hexadecimal

                # Aplicamos la rotación
                decoded_flag = rotate_hex_string(hex_value, rotation)

                # Imprimimos la flag descifrada en formato hexadecimal
```

```
            print(f"Decrypted Flag Type 1: {decoded_flag}")
            print("-----------------------------------------------------------------")
        elif "Hexadecimal Flag" in line:
            # Flag tipo 2
            print("-----------------------------------------------------------------")
            # Expresión regular para extraer los valores
            pattern = r"Rotation: (\d+), s:([0-9a-f]+), op:[^:]+: b'([A-Za-z0-9+/=]+)'"

            # Buscamos los valores
            match = re.search(pattern, line)

            rotation = int(match.group(1))  # Rotación
            seed = match.group(2)           # Semilla s
            base64_value = match.group(3)   # Valor en base64

            # Decodificamos la flag base64 a binario
            flag_bin = base64.b64decode(base64_value)
            # Realizamos la operación XOR con la semilla
            xor_result = xor_with_seed(flag_bin, seed).decode()
            # Aplicamos la rotación
            rotated_flag = rotate_hex_string(xor_result, rotation)

            print("Decrypted Flag Type 2:", rotated_flag)

            print("-----------------------------------------------------------------")
```

decrypt_flags.py