

Fiesta ETSIIT 2022



KERNEL WRITEUP



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

Granada, 11 de mayo de 2022

En este reto se nos entrega un ejecutable llamado *kernel*. Al ejecutarlo vemos que nos pide una clave la cual tenemos que obtener para sacar la flag.

Puesto que no tenemos nada de información, lo primero que debemos hacer es abrir el ejecutable con un desensamblador, en este caso lo haremos con Hopper.

Nada más abrir el ejecutable, si buscamos la palabra *flag*, veremos que está almacenada de forma global en la región de datos de la memoria:

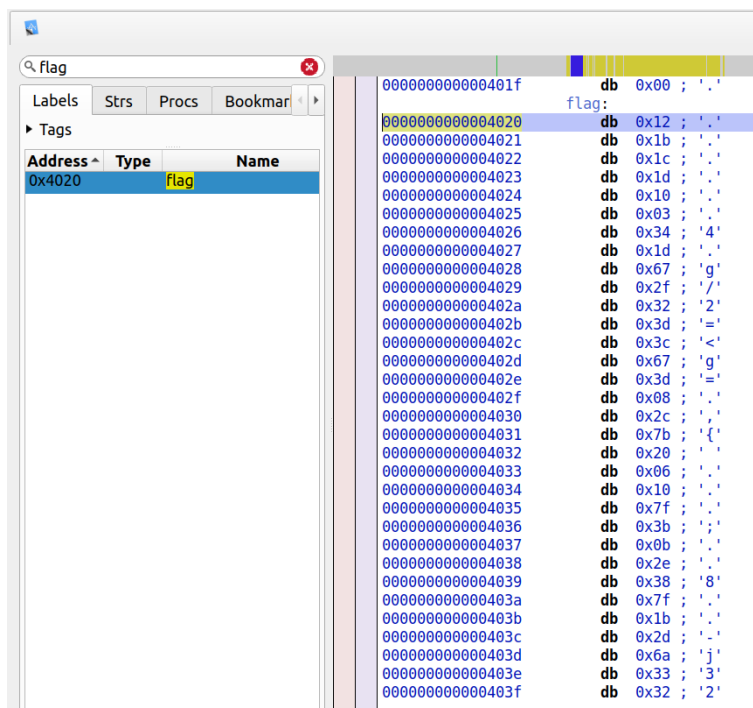
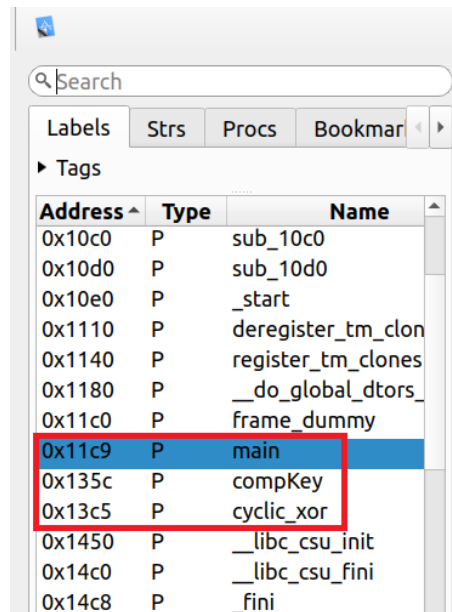


Figura 1: flag en memoria

Vemos que no es legible puesto que está cifrada. Más tarde veremos de qué forma.

Una vez visto esto, pasamos a echarle un vistazo a las etiquetas definidas y vemos tres en concreto que nos llaman la atención, estas son *main*, *compKey* y *cyclic_xor*:



Address	Type	Name
0x10c0	P	sub_10c0
0x10d0	P	sub_10d0
0x10e0	P	_start
0x1110	P	deregister_tm_clones
0x1140	P	register_tm_clones
0x1180	P	__do_global_ctors
0x11c0	P	frame_dummy
0x11c9	P	main
0x135c	P	compKey
0x13c5	P	cyclic_xor
0x1450	P	__libc_csu_init
0x14c0	P	__libc_csu_fini
0x14c8	P	_fini

Figura 2: Etiquetas del programa

Una vez vistas estas etiquetas, podemos hacernos una idea de que el flujo del programa es el siguiente:

1. Introducimos una clave
2. Se compara en la función *compKey*
3. Si la clave es la correcta, se ejecuta la función *cyclic_xor* que descifra la flag almacenada en memoria

Para comprobarlo comenzamos analizando el código del main:

```

main:
00000000000011c9    endbr64                                ; Begin of unwind
00000000000011cd    push    rbp
00000000000011ce    mov     rbp, rsp
00000000000011d1    sub     rsp, 0x20
00000000000011d5    mov     rax, qword [fs:0x28]
00000000000011de    mov     qword [rbp-8], rax
00000000000011e2    xor     eax, eax
00000000000011e4    mov     dword [rbp-0x18], 0x544f4f57
00000000000011eb    mov     word [rbp-0x14], 0x59
00000000000011f1    lea     rdi, qword [_IO_stdin_used+8]    ; 0x2008
00000000000011f8    call    sub_1090                        ; sub_1090
00000000000011fd    lea     rdi, qword [_IO_stdin_used+848]  ; 0x2350
0000000000001204    call    sub_1090                        ; sub_1090
0000000000001209    lea     rdi, qword [aNDN00n]            ; "
0000000000001210    call    sub_1090                        ; sub_1090
0000000000001215    lea     rdi, qword [aNDN00n+232]        ; 0x2508
000000000000121c    call    sub_1090                        ; sub_1090
0000000000001221    lea     rdi, qword [aNclave]           ; "\nClave: "
0000000000001228    mov     eax, 0x0
000000000000122d    call    sub_10b0                        ; sub_10b0
0000000000001232    lea     rax, qword [rbp-0x12]
0000000000001236    mov     rsi, rax
0000000000001239    lea     rdi, qword [aNclave+9]          ; 0x2606
0000000000001240    mov     eax, 0x0
0000000000001245    call    sub_10d0                        ; sub_10d0
000000000000124a    lea     rdx, qword [rbp-0x12]
000000000000124e    lea     rax, qword [rbp-0x18]
0000000000001252    mov     rsi, rdx                        ; argument #2 for
0000000000001255    mov     rdi, rax                        ; argument #1 for
0000000000001258    call    compKey                        ; compKey
000000000000125d    test    al, al
000000000000125f    je      loc_126b

```

Figura 3: main

En el rectángulo marrón vemos que se llama cuatro veces a una etiqueta llamada *sub_1090*, si vamos hasta esa etiqueta, veremos que esta llama a la función *puts* de C, por lo que sabemos que en esa parte del código se está imprimiendo algo por pantalla. En este caso, esas cuatro llamadas corresponden a la impresión de “ETSIIT_CTF”, al primer texto que aparece, a los dibujos en ASCII art y al segundo texto que aparece antes de pedirnos la clave.

En el rectángulo verde vemos que se llama a la etiqueta *sub_10b0*, si vamos hasta esta etiqueta veremos que lo que hace es llamar a la función *printf*, en este caso se está imprimiendo el mensaje que nos pide la clave (se puede ver a la derecha).

En el rectángulo rojo se está llamando a la etiqueta *sub_10d0*, si nos vamos hasta esta etiqueta, veremos que lo que hace es llamar a la función *scanf*. Como ya sabemos, los dos primeros parámetros se pasan a las funciones mediante los registros *%rdi* y *%rsi*, en este orden. De esta forma, vemos que en *%rdi* se está almacenando la posición de memoria de *[aNclave+9]* (que si vamos hasta esa etiqueta veremos que su contenido es “%s”, es decir, le indica a *scanf* que lo que se va a almacenar en el segundo parámetro es un string) mientras que en *%rsi* se está almacenando una posición de memoria, concretamente *[rbp-0x12]*, en esta posición se almacenará la clave introducida por teclado.

Por último, en el rectángulo naranja vemos que se está llamando a la función *compKey* con los argumentos `[rbp-0x12]` y `[rbp-0x18]`, anteriormente hemos visto que `[rbp-0x12]` contiene lo que introducimos por teclado, por lo que deducimos que en `[rbp-0x18]` se encuentra la clave a adivinar, una vez vemos esto, nos damos cuenta de que `[rbp-0x18]` se rellena al inicio del `main`:

```

main:
00011c9          endbr64
00011cd          push     rbp
00011ce          mov      rbp, rsp
00011d1          sub      rsp, 0x20
00011d5          mov      rax, qword [fs:0x28]
00011de          mov      qword [rbp-8], rax
00011e2          xor      eax, eax
00011e4          mov      dword [rbp-0x18], 0x544f4f57
00011eb          mov      word [rbp-0x14], 0x59

```

Figura 4: Clave

Una vez visto esto solo hace falta pasar `0x59544f4f57` a texto, y veremos que es igual a `YTOOW`, puesto que x86 emplea little-endian, le damos la vuelta a la palabra y obtenemos `WOOTY`, que es la clave para obtener la flag.

Una vez la introducimos, *compKey* se encarga de llamar a la función *cyclic_xor*, que a su vez se encarga de aplicar un xor cíclico a la flag cifrada que vimos al principio, completando de esta forma el reto:

```

Clave: WOOTY

¡Enhorabuena, salvaste al gato Kernel!, antes de irse
te ha dejado un regalo ;D

ETSIIT{R3vers3d_c4t_G0t_wo0Ty3d}

```

Figura 5: Clave