Universidad
Rey Juan Carlos

Práctica 6: Análisis

Realizado por:

Javier Burgos Chamorro

ÍNDICE

A)¿Qué bibliotecas usa el programa y qué funciones importa?

Usare radare 2 para esta parte de la practica ; los pasos a seguir son los siguientes :

Primero ejecuto el comando r2 -d analizame , una vez que ya estoy en r2 uso el comando aaa para que se me analice el binario.

Para sacar las funciones ,ejecuto el comando ii para que me muestre las funciones importadas por otra librería y obtengo este resultado.

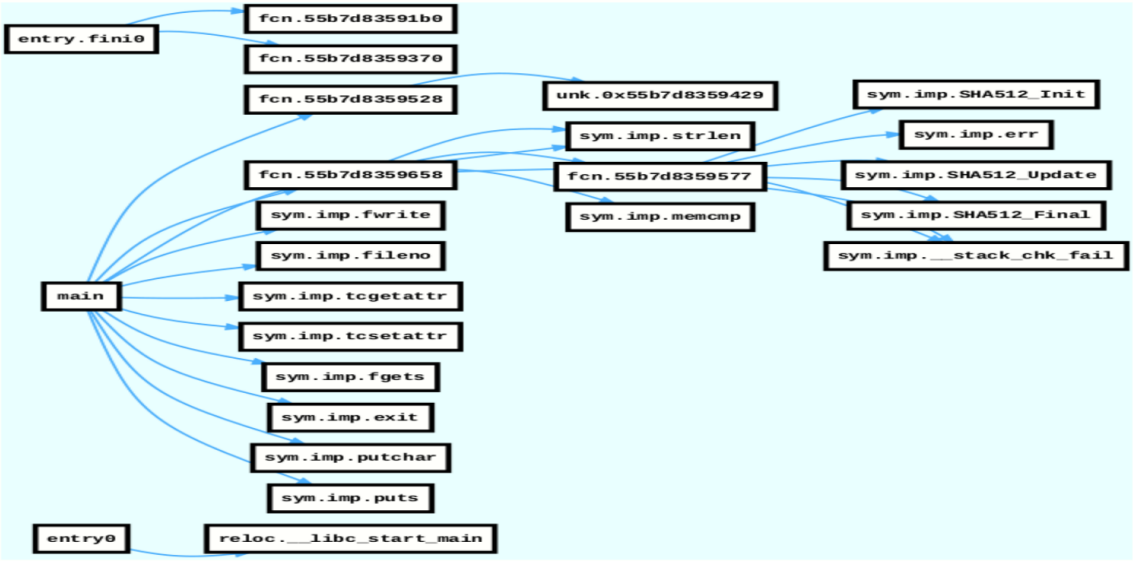
```
----------------------------------------
1   0x55bcdc28d1c0 GLOBAL FUNC       tcsetattr
2   0x55bcdc28d1d0 GLOBAL FUNC       fileno
3   0x55bcdc28d1e0 GLOBAL FUNC       printf
4   0x55bcdc28d1f0 GLOBAL FUNC       memset
5   0x55bcdc28c000 WEAK   NOTYPE     __gmon_start__
6   0x55bcdc28d200 GLOBAL FUNC       puts
7   0x55bcdc28d210 GLOBAL FUNC       exit
8   0x55bcdc28d220 GLOBAL FUNC       putchar
9   0x55bcdc28c000 GLOBAL FUNC       __libc_start_main
10  0x55bcdc28d230 GLOBAL FUNC       BIO_push
11  0x55bcdc28d240 GLOBAL FUNC       fgets
12  0x55bcdc28c000 WEAK   NOTYPE     _ITM_deregisterTMCloneTable
13  0x55bcdc28d250 GLOBAL FUNC       strlen
14  0x55bcdc28c000 WEAK   NOTYPE     _ITM_registerTMCloneTable
15  0x55bcdc28d260 GLOBAL FUNC       BIO_new
16  0x55bcdc28d270 GLOBAL FUNC       BIO_new_mem_buf
17  0x55bcdc28d280 GLOBAL FUNC       err
18  0x55bcdc28d290 GLOBAL FUNC       SHA512_Update
19  0x55bcdc28d2a0 GLOBAL FUNC       tcgetattr
20  0x55bcdc28d2b0 GLOBAL FUNC       BIO_f_base64
21  0x55bcdc28d2c0 GLOBAL FUNC       SHA512_Init
22  0x55bcdc28d2d0 GLOBAL FUNC       __stack_chk_fail
23  0x55bcdc28d2e0 GLOBAL FUNC       BIO_free_all
24  0x55bcdc28d2f0 GLOBAL FUNC       BIO_read
25  0x55bcdc28d300 GLOBAL FUNC       BIO_set_flags
26  0x55bcdc28d310 GLOBAL FUNC       SHA512_Final
27  0x55bcdc28d320 GLOBAL FUNC       memcmp
28  0x55bcdc28d330 GLOBAL FUNC       fwrite
29  0x55bcdc28c000 WEAK   FUNC       __cxa_finalize
```

Y para las librerías ejecuto el comando il , el cual me da las librerías del binario.

```
[0x7f5253626050]> il
[Linked libraries]
libcrypto.so.1.1
libc.so.6

2 libraries
```

B) Genera y pega en el PDF un grafo de llamadas global del programa (en formato vertical para que se pueda ver fácilmente).



C) Crea una copia del binario y parchea esa copia para que se muestre el mensaje secreto sin tener que saber la contraseña. ¿Cómo has dado con la condición? ¿Cuál es el mensaje secreto? Explica cómo lo has hecho y pega en el PDF una captura del terminal después de ejecutar el binario parcheado.

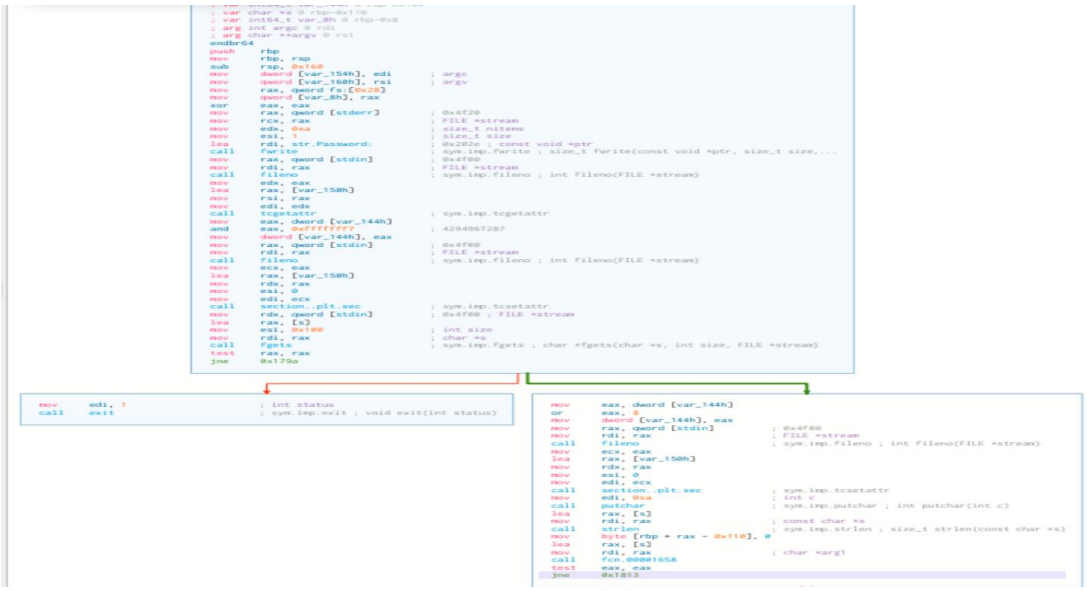
Para este apartado uso cutter por comodidad que aporta a la hora de poder cambiar líneas en el binario ya que deja editar en el propio código directamente .

Para poder escribir en el binario necesitamos ejecutar el cutter en modo escritura , por lo que el comando que uso es este :

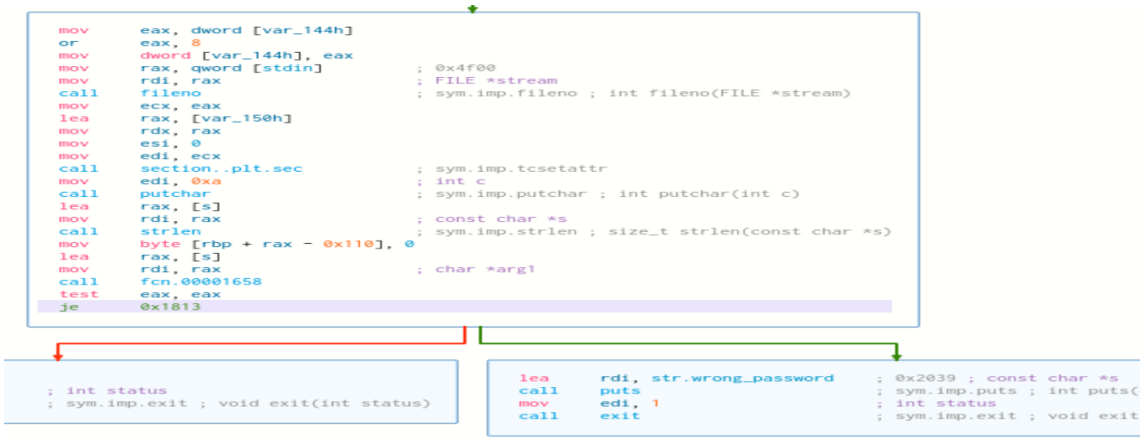
Cutter -w analizame

Con esto ya lo abrimos en modo escritura y pasamos a analizar el main.

Si nos fijamos en el grafico de funciones que nos aporta el cutter podemos ver como el main salta a una función si se cumple el salto .



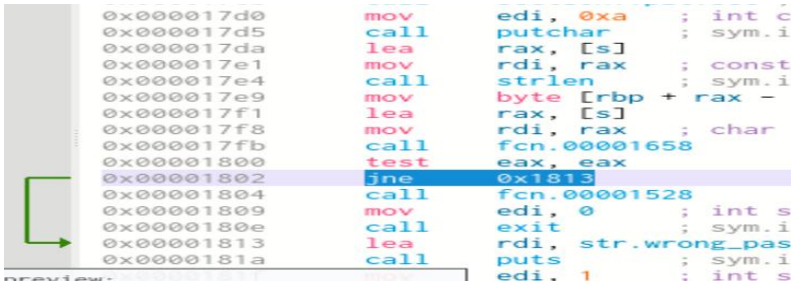
De tal manera que si nos fijamos en la función a la que salta y en el grafico que esta nos aporta , vemos que es la encargada de decir si es wrong password y entonces te saca del programa o si te devuelve al programa principal devolviéndote el secreto.



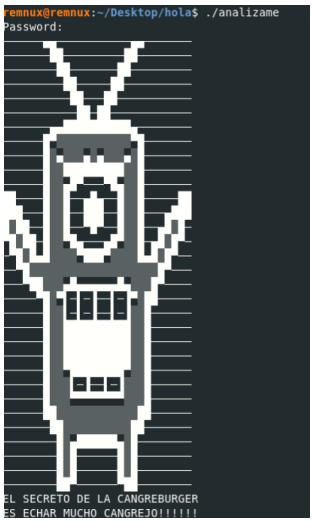
Ahora para evitar esa comprobación es tan sencillo como invertir el salto condicional , esto se hace haciendo click derecho en el salto y dándole a invertir. Al invertirlo conseguimos que no salte a la función y por tanto que no nos compruebe la password.

Básicamente localizo el salto .

Y lo invirto ,quedando el codigo asi.



Ejecuto el analizame y me da la el secreto.



Por tanto nos hemos saltado la comprobación tal y como esperábamos.

D) ¿Dónde y cómo está almacenada la información secreta que escribe el binario por su salida cuando la contraseña es correcta? ¿Qué función o funciones se encargan de escribir el mensaje secreto por la salida? Especifica la sección y el offset en el que se encuentran los datos en el binario y si están codificados/ofuscados.

En este apartado hare uso de cutter como de radare.

Al analizar antes el código en cutter pudimos, fijarnos en la función fcn.00001528 pues al verla con calma podemos apreciar un bucle de hasta 3708 ,el cual hace un xor con 0X13 byte a byte , tras esto , llama al la función fcn.00001429

```
79: fcn.00001528 ();
; var int64_t var_4h @ rbp-0x4
0x00001528        endbr64
0x0000152c        push rbp
0x0000152d        mov rbp, rsp
0x00001530        sub rsp, 0x10
0x00001534        mov dword [var_4h], 0
0x0000153b        jmp 0x1565
0x0000153d        mov eax, dword [var_4h]
0x00001540        cdqe
0x00001542        lea rdx, str.c_R_c_R_c_R_c_R_c_R_c_R_c_R_crV_crV_c
0x00001549        movzx eax, byte [rax + rdx]
0x0000154d        xor eax, 0x13
0x00001550        mov ecx, eax
0x00001552        mov eax, dword [var_4h]
0x00001555        cdqe
0x00001557        lea rdx, str.c_R_c_R_c_R_c_R_c_R_c_R_crV_crV_c
0x0000155e        mov byte [rax + rdx], cl
0x00001561        add dword [var_4h], 1
0x00001565        mov eax, dword [var_4h]
0x00001568        cmp eax, 0xe7c
0x0000156d        jbe 0x153d
0x0000156f        call fcn.00001429
0x00001574        nop
0x00001575        leave
0x00001576        ret
```

Si nos fijamos en fcn.00001429. esta función lo que hace es descifrar el secreto en base 64 y tras esto lo pinta por pantalla .

```
255: fcn.00001429 ();
; var int64_t var_1020h @ rbp-0x1020
; var int64_t var_1018h @ rbp-0x1018
; var void *s @ rbp-0x1010
; var int64_t canary @ rbp-0x8
0x00001429       endbr64
0x0000142d       push rbp
0x0000142e       mov rbp, rsp
0x00001431       sub rsp, map.home_remnux_Desktop_hola_analizame.r_x ; 0x1000
                                    ; fcn.00001000
0x00001438       or qword [rsp], 0
0x0000143d       sub rsp, 0x20
0x00001441       mov rax, qword fs:[0x28]
0x0000144a       mov qword [canary], rax
0x0000144e       xor eax, eax
0x00001450       lea rax, [s]
0x00001457       mov edx, map.home_remnux_Desktop_hola_analizame.r_x ; fcn.00001000
                                    ; 0x1000 ; size_t n
0x0000145c       mov esi, 0         ; int c
0x00001461       mov rdi, rax       ; void *s
0x00001464       call memset        ; sym.imp.memset ; void *memset(void *s, int c,
0x00001469       mov esi, 0xffffffff ; -1
0x0000146e       lea rdi, str.c_R_c_R_c_R_c_R_c_R_crV_crV_c_R_c_R_c_R_c_R_c_R_c_
0x00001475       call BIO_new_mem_buf ; sym.imp.BIO_new_mem_buf
0x0000147a       mov qword [var_1020h], rax
0x00001481       call BIO_f_base64  ; sym.imp.BIO_f_base64
0x00001486       mov rdi, rax
0x00001489       call BIO_new       ; sym.imp.BIO_new
0x0000148e       mov qword [var_1018h], rax
0x00001495       mov rdx, qword [var_1020h]
0x0000149c       mov rax, qword [var_1018h]
0x000014a3       mov rsi, rdx
0x000014a6       mov rdi, rax
0x000014a9       call BIO_push      ; sym.imp.BIO_push
0x000014ae       mov qword [var_1020h], rax
0x000014b5       mov rax, qword [var_1020h]
0x000014bc       mov esi, 0x100
0x000014c1       mov rdi, rax
0x000014c4       call BIO_set_flags ; sym.imp.BIO_set_flags
0x000014c9       lea rcx, [s]
```

Si queremos obtener el offset es tan sencillo como ejecutar en radare el comando iS el cual nos da las direcciones virtuales entre otras cosas ,nosotros suponemos que el secreto se cargara en el . data de tal manera que la dirección virtual de data es 0x55ebc90f1000 .

```
[0x55ebc90ee549]> iS
[Sections]

nth paddr        size vaddr            vsize perm name
0   0x00000000     0x0 0x00000000        0x0 ----
1   0x00000318    0x1c 0x55ebc90ed318   0x1c -r-- .interp
2   0x00000338    0x20 0x55ebc90ed338   0x20 -r-- .note.gnu.proper
3   0x00000358    0x24 0x55ebc90ed358   0x24 -r-- .note.gnu.build_
4   0x0000037c    0x20 0x55ebc90ed37c   0x20 -r-- .note.ABI_tag
5   0x000003a0    0x30 0x55ebc90ed3a0   0x30 -r-- .gnu.hash
6   0x000003d0   0x300 0x55ebc90ed3d0  0x300 -r-- .dynsym
7   0x000006d0   0x193 0x55ebc90ed6d0  0x193 -r-- .dynstr
8   0x00000864    0x40 0x55ebc90ed864   0x40 -r-- .gnu.version
9   0x000008a8    0x50 0x55ebc90ed8a8   0x50 -r-- .gnu.version_r
10  0x000008f8    0xf0 0x55ebc90ed8f8   0xf0 -r-- .rela.dyn
11  0x000009e8   0x240 0x55ebc90ed9e8  0x240 -r-- .rela.plt
12  0x00001000    0x1b 0x55ebc90ee000   0x1b -r-x .init
13  0x00001020   0x190 0x55ebc90ee020  0x190 -r-x .plt
14  0x000011b0    0x10 0x55ebc90ee1b0   0x10 -r-x .plt.got
15  0x000011c0   0x180 0x55ebc90ee1c0  0x180 -r-x .plt.sec
16  0x00001340   0x565 0x55ebc90ee340  0x565 -r-x .text
17  0x000018a8     0xd 0x55ebc90ee8a8    0xd -r-x .fini
18  0x00002000    0x48 0x55ebc90ef000   0x48 -r-- .rodata
19  0x00002048    0x64 0x55ebc90ef048   0x64 -r-- .eh_frame_hdr
20  0x000020b0   0x188 0x55ebc90ef0b0  0x188 -r-- .eh_frame
21  0x00002cf0     0x8 0x55ebc90f0cf0    0x8 -rw- .init_array
22  0x00002cf8     0x8 0x55ebc90f0cf8    0x8 -rw- .fini_array
23  0x00002d00   0x200 0x55ebc90f0d00  0x200 -rw- .dynamic
24  0x00002f00   0x100 0x55ebc90f0f00  0x100 -rw- .got
25  0x00003000   0xefd 0x55ebc90f1000  0xefd -rw- .data
26  0x00003efd     0x0 0x55ebc90f1f00   0x30 -rw- .bss
27  0x00003efd    0x2a 0x00000000       0x2a ---- .comment
28  0x00003f27   0x10a 0x00000000      0x10a ---- .shstrtab
```

Ahora ejecutamos dr y vemos que el registro rdx donde se almacena la variable es 0x55ebc90f1080 .

```
[0x55ebc90ee549]> dr
rax = 0x00000000
rbx = 0x55ebc90ee830
rcx = 0x00000000
rdx = 0x55ebc90f1080
r8 = 0x2a3646d9426771d8
r9 = 0x218c5954e2540b80
r10 = 0x895cd3d94d9fba89
r11 = 0x287f7f4604d889e6
r12 = 0x55ebc90ee340
r13 = 0x7ffe6a46ee90
r14 = 0x00000000
r15 = 0x00000000
rsi = 0x55ebc90f1020
rdi = 0x7ffe6a46ebe0
rsp = 0x7ffe6a46ec20
rbp = 0x7ffe6a46ec30
rip = 0x55ebc90ee549
rflags = 0x00000297
orax = 0xffffffffffffffff
[0x55ebc90ee549]> iS
```

Si a continuación cogemos el campo paddr de .data (0x00003000) y le sumamos a este la diferencia de las direcciones virtuales anteriores , nos da el offset del secreto ; en este caso 0x00003080.

E) ¿Sabrías extraer la información secreta (decodificarla/desofuscarla) sin tener que parchear el binario y/o ejecutarlo? Si es así, indica cómo lo harías paso a paso (herramientas, etc.) y muestra el resultado.

Para sacar la información secreta , están sencillo como usar este comando.

```
remnux@remnux:~/Desktop/hola$ xxd -s 0x00003080 -l 3708 -ps analizame
2763405227634052276340522763405227634052276340522763725627637256276340522763
7256276340522763405227634052276340522763405227634052276340522763405227634052
2763405227634052276340522763405227637256276372562763405227634052276340522763
4052276340522763405227634052276340522763405227634052506658467 45c5846
745c5846745c5846745c5846745c5846745c5846745c58447a5c58447a5c
5846745c5846745c5846745c5846745c5846745c5846745c5846745c5846
745c5846745c58447a5c58447a5c5846745c5846745c5846745c5846745c
5846745c5846745c58467452617a7f5a577a7f5a577a7f5a577a
7f5a577a7f5a577a7f5a577a7f5a577a7f5a577a7f7c797a7f7c797a7f5a
577a7f5a577a7f5a577a7f5a577a7f5a577a7f5a577a7f7c797a
7f7c797a7f5a577a7f5a577a7f5a577a7f5a577a7f5a577a7f5a577a7f5a
577a7f5a577a7f5a5258276340522763405227634052276340522763405227634052
2763405227634052276340522763405227634052276340522763725a2763725a2763405227634052
4052276340522763405227634052276340522763725a2763725a2763405227634052
2763405227634052276340522763405227634052276340522763405227634052276340522763
4052506658467 45c5846745c5846745c5846745c5846745c5846745c5846745c5846
745c5846745c5846745c5846745c58447a5c58447a5c5846745c5846745c
5846745c58447a5c58447a5c5846745c5846745c5846745c5846745c5846
745c5846745c5846745c5846745c5846745c5846745c58467452617a7f5a
577a7f5a577a7f5a577a7f5a577a7f5a577a7f5a577a7f5a577a
7f5a577a7f5a577a7f5a577a7f7c797a7f7c797a7f5a577a7f7c797a7f7c
797a7f5a577a7f5a577a7f5a577a7f5a577a7f5a577a7f5a577a
7f5a577a7f5a577a7f5a577a7f5a577a7f5a5258276340522763405227634052276340522763
```

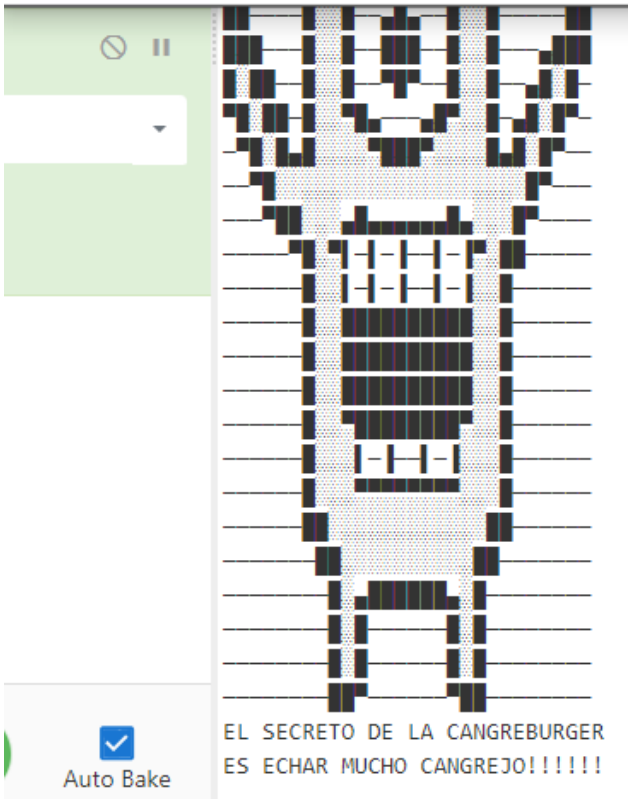
Y si vemos nos tira el mensaje ofuscado. Esto mensaje es posible sacarlo debido a que tenemos el offset , y el tamaño del mensaje que va a ser el número de interacciones del bucle de la función que se encarga de descifrar antes mencionado 3708.

Ahora vamos al ciber chef y hacemos varias transformaciones para sacar el mensaje.

1=La inversa de hexadecimal dando como resultado un string cifrado en base 64.

2=Sobre este cifrado en base 64 hacemos un xor del 0x13 ;como hacia la función de descifrar del programa , dando como resultado solo el cifrado en base 64.

3=Por ultimo hacemos la inversa de base 64, de tal manera que nos arroja el mensaje secreto.



F) ¿Y realizando un análisis dinámico? Si es así, indica cómo lo harías paso a paso (herramientas, etc.) y muestra el resultado.

Para mí esto, es lo que estoy haciendo en el apartado c , pues a fin de cuentas es modificar el código para analizar que salidas me da según que modifique o entradas le pase.

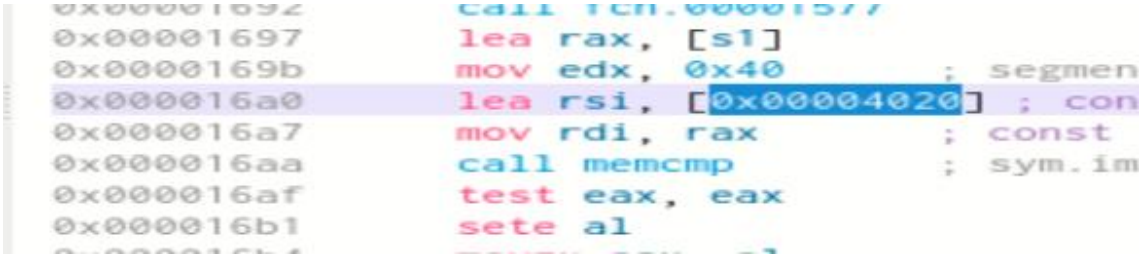
G) ¿Sabes cómo se comprueba si la contraseña introducida es correcta? ¿Qué función o funciones se encargan de esto? Indica en qué sección y offset está la contraseña almacenada, su formato (si está cifrada, etc.) y qué harías para conseguir esa contraseña en claro.

En este apartado hare uso de cutter como de radare.

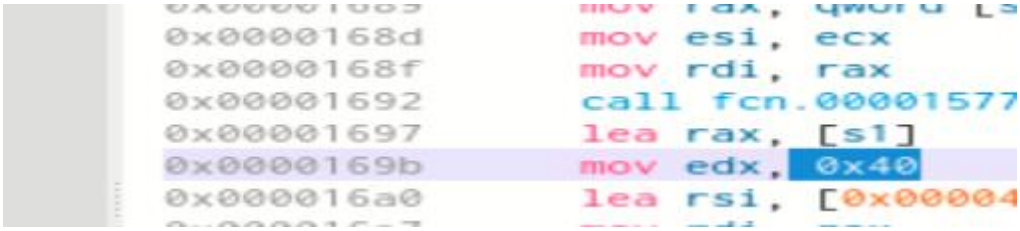
La función encargada de comprobar la contraseña fcn.00001658 ,si nos fijamos tiene la instrucción memcmp que usa para comparar.

```
117: fcn.00001658 (char *arg1);
; var char *s @ rbp-0x58
; var void *s1 @ rbp-0x50
; var int64_t canary @ rbp-0x8
; arg char *arg1 @ rdi
0x00001658      endbr64
0x0000165c      push    rbp
0x0000165d      mov     rbp, rsp
0x00001660      sub     rsp, 0x60
0x00001664      mov     qword [s], rdi  ; arg1
0x00001668      mov     rax, qword fs:[0x28]
0x00001671      mov     qword [canary], rax
0x00001675      xor     eax, eax
0x00001677      mov     rax, qword [s]
0x0000167b      mov     rdi, rax        ; const char *s
0x0000167e      call    strlen          ; sym.imp.strlen ; size_t strlen(const char *s)
0x00001683      mov     ecx, eax
0x00001685      lea     rdx, [s1]       ; int64_t arg3
0x00001689      mov     rax, qword [s]
0x0000168d      mov     esi, ecx        ; int64_t arg2
0x0000168f      mov     rdi, rax        ; int64_t arg1
0x00001692      call    fcn.00001577
0x00001697      lea     rax, [s1]
0x0000169b      mov     edx, 0x40       ; segment.PHDR ; size_t n
0x000016a0      lea     rsi, [0x00004020] ; const void *s2
0x000016a7      mov     rdi, rax        ; const void *s1
0x000016aa      call    memcmp          ; sym.imp.memcmp ; int memcmp(const void *s1, cor
0x000016af      test    eax, eax
0x000016b1      sete    al
0x000016b4      movzx   eax, al
```

El offset de la contraseña es este 0x00004020 pues lo está cargando en el programa justo en la instrucción que esta encima del memcmp



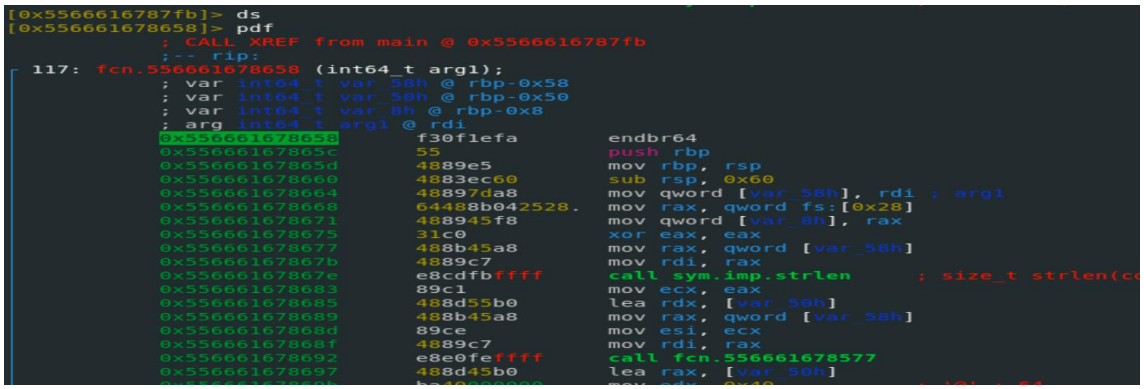
El tamaño de la contraseña se puede sacar por la línea de arriba justo antes de la instrucción que carga la contraseña. En este caso 0x40 ;es decir 64.



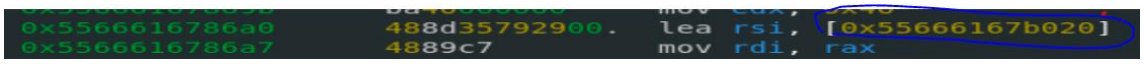
El problema que se nos plantea es que no tenemos su dirección , por tanto lo que hago es ir a radare ir poniendo break point a lo largo del programa para ir debuggenado hasta llegar la instrucción que carga la password.

Primero pongo un break point en el main.

Tras esto pongo un break point en la función que gestiona la comparación de la contraseña y entro dentro de esta función para si cargar la contraseña en memoria.



Encuentro en la función la dirección virtual , la cual carga la contraseña



Y por último hago un volcado

Sacando el string de la password ofuscado.

Para poder descifrarlo , primero supondría que el cifrado que usa es SHA 512, pues en el primer apartado vemos que importa la librería de cifrado de SHA 512,el tamaño ya lo tenemos que es 64 .Sabiendo esto podríamos hacer un ataque de diccionario.