Desensamble de la bomba de Jaime Pérez

Alberto Plaza Montes.

2ºB1 ingeniería informática

Introducción

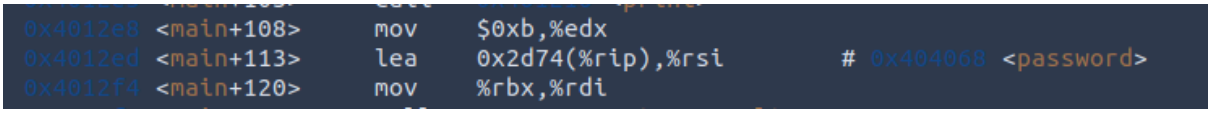
Durante el desarrollo del siguiente trabajo documentaré el desensamble de la bomba de mi compañero Jaime Perez García. Para ello, pasaremos por dos etapas, la encriptación de una contraseña de prueba, por lo que será relativamente arbitraria, con el objetivo de ver cómo se produce el encriptamiento de la misma, y así poder realizar los mismos pasos pero en orden inverso. Después llevaremos a cabo el mismo proceso con el pin para así terminar de resolver la bomba.

Para poder llevar a cabo el desensamble utilizaremos el gdb, depurador por defecto que nos ofrece GNU en la mayoría de plataformas UNIX, en este caso, ubuntu. Además aprovecharemos la opción -tui que nos dotará de una útil interfaz gráfica para poder ver así tanto el código en ensamblador como los registros que va utilizando el programa.

Encriptamiento de la contraseña

Tal y como hemos dicho durante la introducción, para desencriptar la contraseña utilizaremos una contraseña de prueba elegida de forma arbitraria, de modo que podamos rastrear las operaciones que se van realizando sobre la misma y aplicarlas de manera inversa a la contraseña original encriptada. Sin embargo, la arbitrariedad de la contraseña que escojamos no será total, nos conviene utilizar una que tenga la misma longitud, de modo que buscaremos en la función main del ensamblador la longitud de la contraseña encriptada.

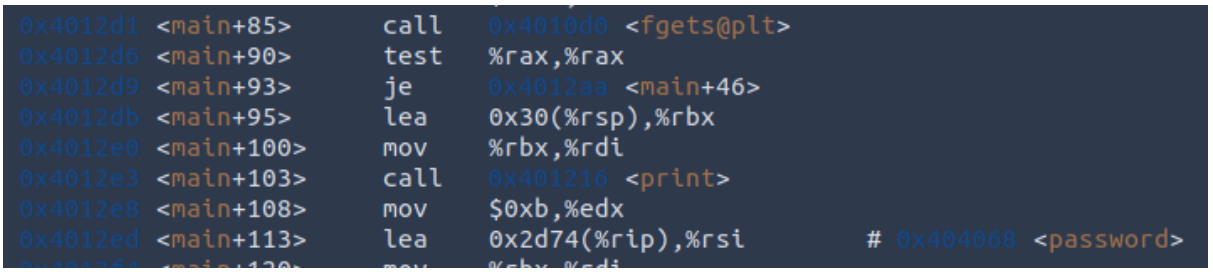
Aprovecharemos el momento en el que compara el encriptado de la contraseña introducida con la contraseña encriptada real, para ver la dirección de esta última.



En la imagen podemos observar como 0x404068 es la dirección de la contraseña original encriptada, por lo que podemos consultarla mediante la orden **p(char*)0x404068** que como resultado nos da: $1 = 0x404068 <password> "**UXVQYRqwy2**" De modo que ya sabemos cual es la contraseña real encriptada, además de su longitud, que es de un total de **10 carácteres**. Por lo que nuestra contraseña de prueba debe contar con estos, elegiremos

como ejemplo de contraseña a encriptar “HolaHolaii”, que cuenta exactamente con los 10 carácteres que necesitábamos.

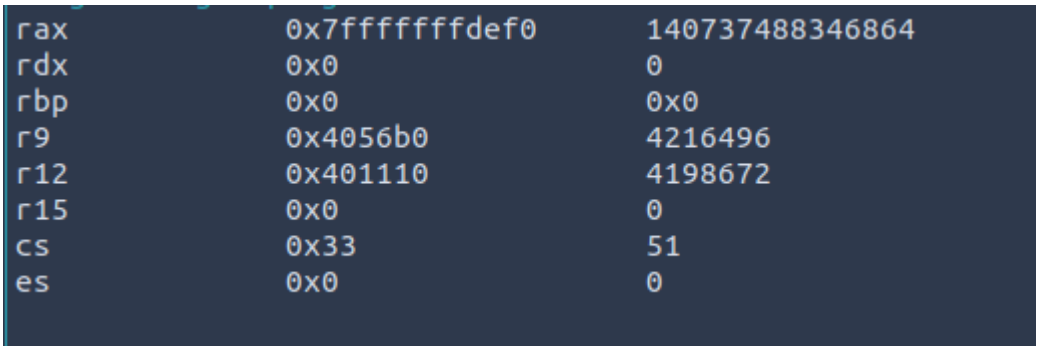
Lo siguiente sería poner un breakpoint al comienzo de la función de encriptado para poder empezar a debuggear, sin embargo no la encontramos por ningún lado. Pero sabemos que esta se tiene que ejecutar en algún momento entre el get y la comparación.



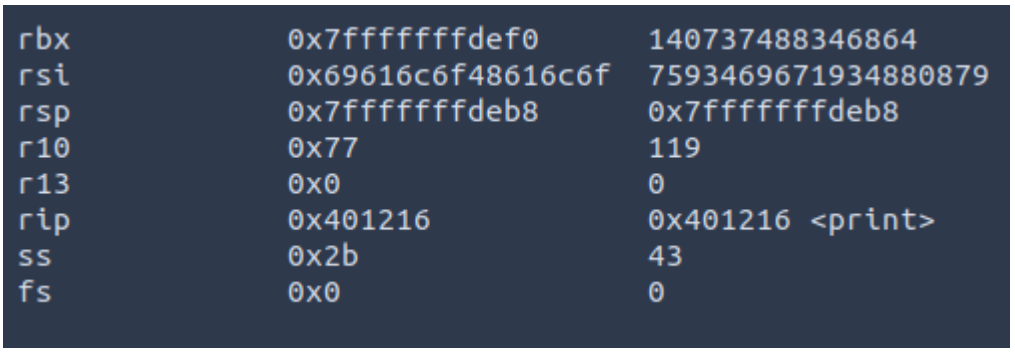
Parece ser que Jaime ha escondido la función para encriptar bajo el nombre de “print”. Ya que estamos aquí, aprovecharemos la comprobación que hace entre la cadena introducida y el código encriptado para obtener este último. Lo haremos con **p(char*)0x404068** que nos devolverá como resultado **$2 = 0x404068 <password> "UXVQYRqwy2".** la contraseña de Jaime encriptada es: “**UXVQYRqwy2”**.

De modo que ahora sí, podemos colocar un breakpoint al comienzo de la función y comenzar a debuggear. comenzaremos la depuración, tras introducir “HolaHolaii” como contraseña de prueba a encriptar.

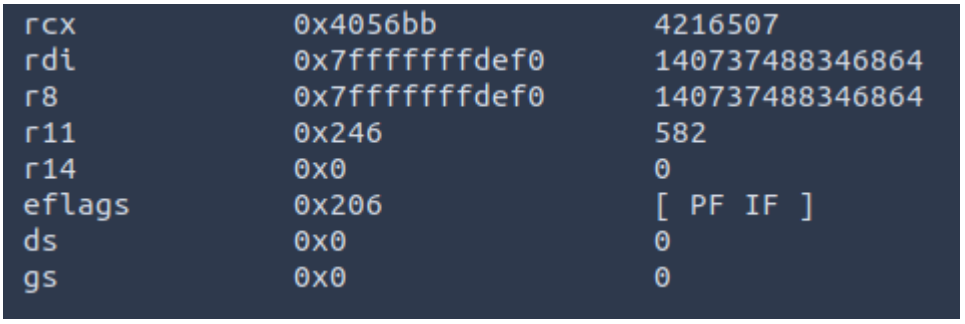
Nada más comenzar el programa podemos observar información en los registros principales



```
rax            0x7fffffffdef0      140737488346864
rdx            0x0                 0
rbp            0x0                 0x0
r9             0x4056b0            4216496
r12            0x401110            4198672
r15            0x0                 0
cs             0x33                51
es             0x0                 0
```



```
rbx            0x7fffffffdef0      140737488346864
rsi            0x69616c6f48616c6f  7593469671934880879
rsp            0x7fffffffdeb8      0x7fffffffdeb8
r10            0x77                119
r13            0x0                 0
rip            0x401216            0x401216 <print>
ss             0x2b                43
fs             0x0                 0
```



```
rcx            0x4056bb            4216507
rdi            0x7fffffffdef0      140737488346864
r8             0x7fffffffdef0      140737488346864
r11            0x246               582
r14            0x0                 0
eflags         0x206               [ PF IF ]
ds             0x0                 0
gs             0x0                 0
```

Al contrario que sucedía en mi bomba, que destacaba especialmente el parámetro rsi, en este caso no hay ningún registro que parezca destacar, por lo que es probable que solo introduzca la cadena como parámetro y esta esté en %rdi, lo comprobaremos.

```
(gdb) p(char*)$rdi
$1 = 0x7fffffffdef0 "HolaHolaii\n"
```

Efectivamente es tal y como habíamos descrito.

Ahora analizaremos el código:

Podemos observar en las primeras líneas nada más comenzar, una estructura que tiene forma de bucle. Esta comienza con una i = 0 (el movimiento de 0 a %eax) y posteriormente el movimiento de %eax a %rdx extendiendo el signo, por lo que parece ser que se usará el registro %rdx a modo de i del bucle. Respecto a la condición del bucle es que se ejecutará mientras el char i de la cadena sea distinto de 0. Esto lo sabemos gracias a las instrucciones +12 y +16.

- La instrucción +12 compara un 0 con (%rdi,%rdx,1), lo que se traduce en 1*i + %rdi (es decir, a partir de la posición de memoria de %rdi, comprueba el carácter i).
- La instrucción +16 se traduce en que realizará el salto siempre y cuando sea distinto de 0.

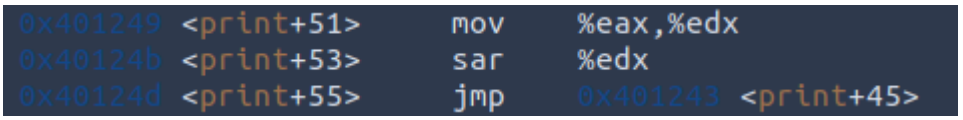
Esta estructura nos da que pensar que se trata de un bucle de la siguiente manera: **while(cadena[i] != 0).** El sentido de esto es aprovechar que las cadenas en C terminan en \0 para ayudarse a recorrer la cadena entera.

A continuación en la instrucción +19 vemos claramente como incrementa en 1 %eax, y por consiguiente la i.

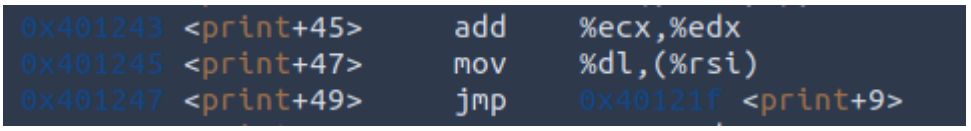
El siguiente paso es realizar un cálculo con la instrucción lea y guardarlo en %rsi. Este cálculo es -0x1(%rdi,%rdx,1), esto es, acceder a al elemento anterior de la cadena en el que se encuentra i, y guardarlo en el registro %rsi. Después el compilador lo pasa a %ecx para operar con él.

Justo después nos encontramos con una comparación en las líneas +33 y +36, estas nos dividen el problema en dos situaciones, los elementos de la cadena anteriores o iguales a 6 y sus posteriores.

Analicemos primero los elementos anteriores a 7, que suponen el salto a la línea +51. después desplaza los bits de %edx hacia la derecha. El bit del signo no se desplaza y además se copia en el bit inmediatamente a la derecha. Esto permite dividir entre dos un operando con signo.

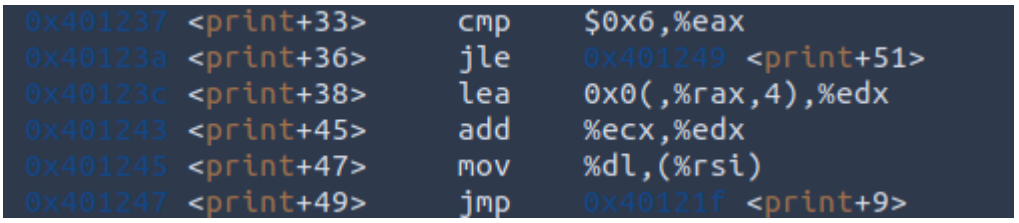


Finalmente suma la letra que había almacenado anteriormente con el resultado de la división y la sustituye en su posición pertinente tal y como vemos en la línea +47. Y comienza una nueva iteración.



Siguiente este procedimiento de manera iterativa podemos obtener los primeros 6 carácteres, aplicando la misma lógica pero a la inversa en la contraseña encriptada, obtenemos los 6 primeros dígitos que serán: “UWUOWO”

Respecto al elemento 7 y posteriores:



podemos ver como calcula 4*i y lo mete en %edx, para después sumar la letra y el resultado de 4*i.

Siguiendo de nuevo la lógica inversa de forma iterativa, podemos obtener las últimas letras que nos quedaban para desencriptar la palabra. De modo que el resultado final es: **“UWUOWOUWU\n”.**

Encriptamiento del pin

De la misma forma que antes, buscaremos cuál es el passcode encriptado mediante **p*(int)0x404060** y obtenemos la respuesta: **$1 = 19790** Por lo que ya sabemos cual es el pass code encriptado. Al igual que antes Jaime ha escondido de nuevo la función para encriptar el passcode bajo otro nombre que esta vez vemos rápidamente, se trata de la función pritf. Así que vamos hacia ella para ver el código.

```
0x401367 <main+235>    call   0x4010f0 <__isoc99_scanf@plt>
0x40136c <main+240>    cmp    $0x1,%ebx
0x40136f <main+243>    jne    0x401329 <main+173>
0x401371 <main+245>    lea    0xc(%rsp),%rdi
0x401376 <main+250>    call   0x40124f <pritf>
0x40137b <main+255>    mov    0x2cdf(%rip),%eax        # 0x404060 <passcode>
```

Observamos como primero se mueve el contenido de %rdi, que probablemente sea el número a encriptar, al registro %eax.

Después opera con el, multiplicando el número por sí mismo y después sumándole 20 en hexadecimal (32 en decimal). De este modo, podemos plantear una ecuación con el pin encriptado. 2x+32=19790. Al resolverla obtenemos que el pin desencriptado es **9879**.

```
0x40124f <pritf>       endbr64
0x401253 <pritf+4>     mov    (%rdi),%eax
0x401255 <pritf+6>     lea    0x20(%rax,%rax,1),%eax
0x401259 <pritf+10>    mov    %eax,(%rdi)
0x40125b <pritf+12>    ret
```

Conclusión

Finalmente gracias a algunos procedimientos de inversión y el análisis del código en ensamblador hemos podido obtener la contraseña y el pin de la bomba desencriptadas, los cuales son:

Contraseña: UWUOWOUWU

Pin: 9879

```
albertoplaza@AlbertoUbuntu:~/Universidad/EC/bomba jaime$ ./bomba_jaime

Introduce la contraseÃ±a: UWUOWOUWU

Introduce el pin: 9879

Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·
Â·Â·Â· bomba desactivada Â·Â·Â·
Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·Â·
```