Desensamble de mi propia bomba

Alberto Plaza Montes.

2ºB1 ingeniería informática

Introducción

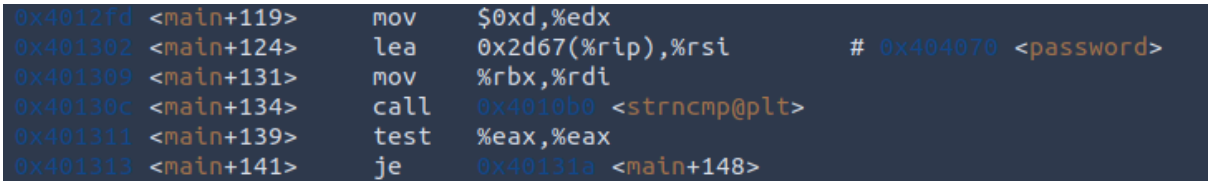
Durante el desarrollo del siguiente trabajo documentaré el desensamble de mi propia bomba, para ello, pasaremos por dos etapas, la encriptación de una contraseña de prueba, por lo que será relativamente arbitraria, con el objetivo de ver cómo se produce el encriptamiento de la misma, y así poder realizar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

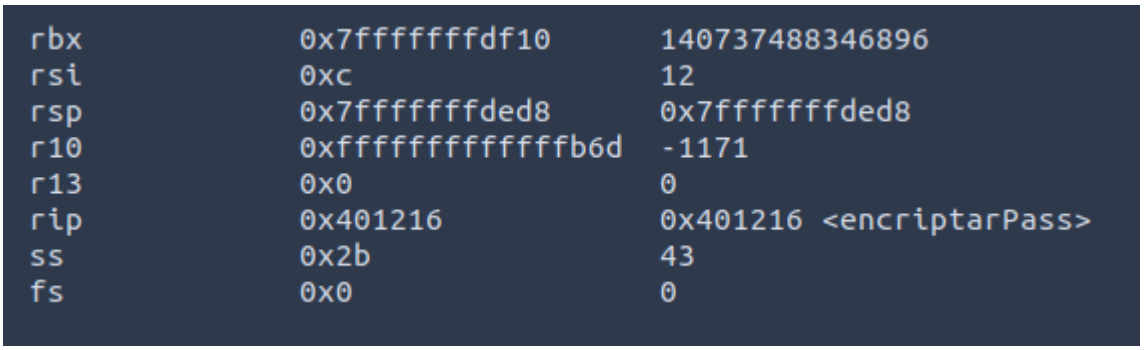
Aprovecharemos el momento en el que compara el encriptado de la contraseña introducida con la contraseña encriptada real, para ver la dirección de esta última.



En la imagen podemos observar como 0x404070 es la dirección de la contraseña original encriptada, por lo que podemos consultarla mediante la orden **p(char*)0x404070** que como resultado nos da: $1 = 0x404070 <password> "**qGt6qxCkGIn\n**". De modo que ya sabemos cual es la contraseña real encriptada, además de su longitud, que es de un total de **11 carácteres** (sin contar el /n). Por lo que nuestra contraseña de prueba debe contar con estos, elegiremos como ejemplo de contraseña a encriptar mis apellidos, “PlazaMontes”, que cuenta exactamente con los 11 carácteres que necesitábamos.

de este modo, pondremos un breakpoint en el comienzo de la función de encriptado, la cual se puede encontrar a golpe de vista en el código ensamblador y comenzaremos la depuración, tras introducir “PlazaMontes” como contraseña de prueba a encriptar.

Nada más comenzar el programa podemos observar información en los dos registros principales, rdi y rsi. Por lo que es probable que la función encriptarPass tenga, al menos, dos parámetros.



%rsi tiene concretamente el número 12, por lo que, teniendo en cuenta que la contraseña real encriptada es (“**qGt6qxCkGln\n”**), es probable que el segundo parámetro sea la longitud de la contraseña.

```
rcx            0x10                16
rdi            0x7fffffffdf10      140737488346896
r8             0x7fffffffdf10      140737488346896
r11            0x7ffff7f44f30      140737353371440
r14            0x0                 0
eflags         0x206               [ PF IF ]
ds             0x0                 0
gs             0x0                 0
```

Respecto al registro rdi no es descifrable a simple vista, pero es obvio que para encriptar la contraseña, habrá que pasarla como parámetro, por lo que comprobaremos si se trata de una cadena de char mediante la orden **p(char*)$rdi**

```
(gdb) p(char*)$rdi
$2 = 0x7fffffffdf10 "PlazaMontes\n"
```

Efectivamente, el primer parámetro, rdi, es el puntero a char.

Ahora que tenemos claro que es cada registro, analizaremos los primeros movimientos del código en ensamblador el cual es el siguiente:

```
+>0x401216 <encriptarPass>        endbr64
  0x40121a <encriptarPass+4>      addb   $0x1,(%rdi)
  0x40121d <encriptarPass+7>      movslq %esi,%rax
  0x401220 <encriptarPass+10>     subb   $0x1,-0x2(%rdi,%rax,1)
  0x401225 <encriptarPass+15>     mov    $0x1,%eax
```

Podemos observar como lo primero que hace es sumar 1 en el contenido de %rdi, es decir, al comienzo del vector a char. Por lo que la “P” de nuestra contraseña “PlazaMontes” pasará a ser una “Q”.

Después mueve la longitud de la cadena, almacenada en %esi a %rax mediante un mov, con el añadido slq para llevar a cabo la extensión de signo. A continuación lo usa en una operación de resta, en la que resta 1 en -0x2(%rdi,%rax,1). Traduciremos esto para ver de qué posición se trata. la parte de (%rdi,%rax,1) se puede traducir como 1*%rax + %rdi, es decir accede al comienzo de la posición del puntero a char, para sumarle 12 posiciones (la longitud de vector que pasamos por 2º parámetro y después movimos a %rax) y colocarse así una posición más adelante del \n. para después restarle 2, de modo que, teniendo en cuenta el \n, nos hemos colocado al final de la cadena, es decir en la “s” de “PlazaMontes”. Finalmente la orden resta un 1 al contenido de esa posición, es decir a la “s”, por lo que pasaría a ser una “r”.

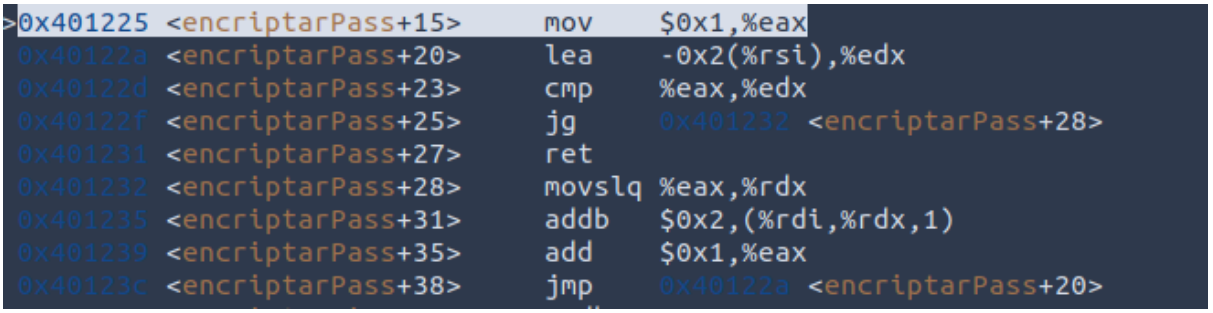
Corroboramos si la cadena va tal y como decimos:

```
(gdb) p(char*)$rdi
$4 = 0x7fffffffdf10 "QlazaMonter\n"
```

De este modo hemos descifrado que el primer bloque de código a la hora de encriptar la cadena introducida lo que hace es sumar 1 al primer elemento y restar 1 a último elemento.

Y Por tanto, aplicando la lógica inversa en la contraseña real encriptada, restando 1 al primero y sumando 1 al último, tenemos la contraseña desencriptada tal que así: "**p??????????o\n".**

Analicemos ahora la segunda parte del algoritmo de encriptación, que es la siguiente:



En este fragmento podemos darnos cuenta de que lo que se realiza es algún tipo de bucle ya que en la +23 se realiza una comprobación que en caso de que se cumpla lleva a cabo acciones y si no finaliza. Además si vamos hasta las acciones que nos manda esa comprobación (+28) en ese fragmento de código nos encontramos un salto que nos devuelve a la línea +20, por lo que confirmamos que es un bucle. De modo que primero analizaremos las condiciones del bucle y después lo que este realiza.
Inicializa %eax a 1, y después vemos como en la +35, justo antes de volver a empezar incrementa lo incrementa, por lo que es probable que se trate de un bucle for que comienza en 1, veamos la condición para saber cuántas iteraciones se llevan a cabo. Al hacer la comprobación en la línea +25, la realiza en entre %eax y %edx, de modo que la línea anterior que realiza un cálculo y lo introduce en %edx, es la condición. -0x2(%rsi), resta 2 a %rsi, que ya habíamos descubierto anteriormente que era el número de elementos que tenía la cadena, por lo que será un 12 - 2 = 0.

Más adelante tenemos una instrucción de salto jg. Esto nos hace poder traducir el bucle de esta manera: **for (int i=1; i<tamañocadena-2; i++).** Así sabemos que el bucle operará con todos los carácteres intermedios de la cadena. Tiene sentido, pues anteriormente se habían calculado los extremos.

Finalmente podemos observar en la +28 y la +32 como lo único que hace es poner %rdx al valor de la i y suma un 2 en la posición del puntero a char que designa la i. Así pues, el resultado final será la suma de 2 a cada elemento intermedio de la cadena.

De igual forma que antes, podemos calcular de la posición 1 a la 9, de nuevo, aplicando la lógica inversa, como al encriptar, le resta 2, al desencriptar se lo sumamos. De modo que de la contraseña encriptada **?Gt6qxCkGI?** (teniendo en cuenta que los signos de interrogación ya los hemos calculado antes)conseguimos obtener los elementos desencriptados **?Er4ovAiEj?**

Encriptamiento del pin

En el caso del pin, podríamos realizar el mismo procedimiento que utilizamos en la contraseña y crear un pin de prueba para analizar qué cambios realiza en el mismo, sin embargo como el código en ensamblador es muy simple no lo necesitaremos.

```
0x40123e <encriptarPin>          endbr64
0x401242 <encriptarPin+4>        lea    -0x4(%rdi),%eax
0x401245 <encriptarPin+7>        ret
```

Podemos comprobar como el único registro (que no sea %eax) utilizado es %rdi, por lo que asumimos que este es el que se utiliza para pasar el pin a encriptar como parámetro. Realiza un lea para hacer cálculos, concretamente restar 4 al contenido de %rdi, después lo introduce en %eax y lo devuelve como resultado de la función, por lo que ya sabemos que lo único que hace para encriptar el pin es restarle 4. buscaremos cuál era el pin encriptado aprovechando de nuevo el momento en el que compara el pin introducido con el encriptado:

```
0x40138a <main+260>     call   0x40123e <encriptarPin>
0x40138f <main+265>     mov    %eax,0xc(%rsp)
0x401393 <main+269>     cmp    0x2ccf(%rip),%eax        # 0x404068 <passcode>
0x401399 <main+275>     je     0x4013a0 <main+282>
0x40139b <main+277>     call   0x401246 <boom>
```

y utilizaremos la orden **p*(int)0x404068** del que obtenemos que el pin encriptado es 1230.

Por lo que aplicaremos de nuevo el método inverso, y sumaremos 4 al pin encriptado para obtener el real. 1230 + 4 = **1234.**

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: pEr4ovAiEjo
Pin: 1234

```
albertoplaza@AlbertoUbuntu:~/Universidad/ED/bomba$ ./bomba

Introduce la contraseÃ±a: pEr4ovAiEjo

Introduce el pin: 1234

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·

albertoplaza@AlbertoUbuntu:~/Universidad/ED/bomba$
```