Desensamble de la bomba de Ismael Tengo

Alberto Plaza Montes.

2ºB1 ingeniería informática

Introducción

Durante el desarrollo del siguiente trabajo documentaré el desensamble de la bomba de mi compañero Ismael Tengo Rodríguez. 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.

```
0x40139b <main+110>     mov    %rax,%rdi
0x40139e <main+113>     mov    $0xa,%edx
0x4013a3 <main+118>     lea    0x2cd6(%rip),%rsi        # 0x404080 <password>
0x4013aa <main+125>     call   0x4010d0 <strncmp@plt>
0x4013af <main+130>     test   %eax,%eax
0x4013b1 <main+132>     je     0x4013b8 <main+139>
0x4013b3 <main+134>     call   0x401256 <boom>
```

En la imagen podemos observar como 0x404080 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 = 0x404080 <password> **"wdwpvw34\n"** De modo que ya sabemos cual es la contraseña real encriptada, además de su longitud, que es de un total de **8 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, “HolaHola”, que cuenta exactamente con los 8 carácteres que necesitábamos.

Lo siguiente es poner un breakpoint al comienzo de la función de encriptado para poder empezar a debuggear, de buscamos la función, la cual podemos ver a simple vista que es “encripta”.

```
B+>0x401296 <encripta>          endbr64
   0x40129a <encripta+4>        mov    %rdi,%rax
   0x40129d <encripta+7>        mov    $0x0,%edx
   0x4012a2 <encripta+12>       cmp    $0x7,%edx
   0x4012a5 <encripta+15>       jle    0x4012a8 <encripta+18>
   0x4012a7 <encripta+17>       ret
   0x4012a8 <encripta+18>       movslq %edx,%rcx
   0x4012ab <encripta+21>       add    %sil,(%rax,%rcx,1)
   0x4012af <encripta+25>       add    $0x1,%edx
   0x4012b2 <encripta+28>       jmp    0x4012a2 <encripta+12>
```

En caso de que el código fuese más complejo, quizás convendría más realizar un seguimiento íntegro mediante el gdb tal y como hicimos en el encriptamiento de la contraseña que llevó a cabo nuestro compañero Jaime, pero como no es el caso, vamos a limitarnos a analizar el código y los registros, de modo que empezaremos por estos últimos.

Este es el estado actual de los registros:

```
rax            0x7fffffffdef0      140737488346864
rdx            0x0                 0
rbp            0x0                 0x0
r9             0x4056b0            4216496
r12            0x401170            4198768
r15            0x0                 0
cs             0x33                51
es             0x0                 0
```

```
rbx            0x401470            4199536
rsi            0x2                 2
rsp            0x7fffffffdeb8      0x7fffffffdeb8
r10            0x77                119
r13            0x0                 0
rip            0x401296            0x401296 <encripta>
ss             0x2b                43
fs             0x0                 0
```

```
rcx            0x4056b9             4216505
rdi            0x7fffffffdef0       140737488346864
r8             0x7fffffffdef0       140737488346864
r11            0x246                582
r14            0x0                  0
eflags         0x206                [ PF IF ]
ds             0x0                  0
gs             0x0                  0
```

A simple vista lo que más destaca es el registro $rsi, este tiene simplemente un 2, lo que probablemente quiera decir que utiliza un segundo parámetro en su función encripta. Confirmaremos nuestras dudas si después resulta que utiliza el registro en el código.

Por otro lado, comprobaremos que contiene el registro rdi para ver si le pasa la cadena introducida como parámetro.

Efectivamente lo hace.

Pasaremos ahora a analizar el código:

En el encripta +4 se mueve el contenido de %rsi a %rax, por lo que sabemos que las operaciones que ejecute sobre %rax repercutirán directamente en la cadena a encriptar.

Durante las lineas 7,12 y 15 vemos una estructura de bucle, mueve un 0 %edx, lo que probablemente sea la inicialización de la variable que recorrerá con el bucle, la llamaremos i, y después hace una comparación entre i y 0, que posteriormente es un jle (less or equal), por lo que esto se traducirá en un bucle del tipo: for(int i=0; i<=7;i++).

Posteriormente mueve con extensión de signo %edx a %rcx, lo que se traduce en que el uso del registro %rcx a modo de i, para después realizar el movimiento en la cadena introducida. En la orden encripta+21 suma %sil (el registro %rsi que antes sospechábamos que usaba como segundo parámetro y que contiene un 2) a 1*i+%rax, es decir, a la posición de la cadena que designa i, cadena[i] += 2. Podemos ver cómo hace esto de forma iterativa, ya que después simplemente incrementa la i y vuelve a comenzar el bucle.

De este modo sabemos que el encriptamiento consiste en sumar dos a cada posición, por lo que usando lógica inversa, podemos descifrar la contraseña y pasar de :

wdwpvw34 → ubuntu12

Encriptamiento del pin

Respecto al encriptamiento del pin, nos encontramos con un código mucho más imponente.

```
0x4012b4 <encripta_pin>         endbr64
0x4012b8 <encripta_pin+4>       push   %rbx
0x4012b9 <encripta_pin+5>       sub    $0x10,%rsp
0x4012bd <encripta_pin+9>       mov    %edi,%r8d
0x4012c0 <encripta_pin+12>      mov    %esi,%ebx
0x4012c2 <encripta_pin+14>      mov    %fs:0x28,%rax
0x4012cb <encripta_pin+23>      mov    %rax,0x8(%rsp)
0x4012d0 <encripta_pin+28>      xor    %eax,%eax
0x4012d2 <encripta_pin+30>      mov    %rsp,%rdi
0x4012d5 <encripta_pin+33>      lea    0xdec(%rip),%rcx        # 0x4020c8
0x4012dc <encripta_pin+40>      mov    $0x8,%edx
0x4012e1 <encripta_pin+45>      mov    $0x1,%esi
0x4012e6 <encripta_pin+50>      call   0x401160 <__sprintf_chk@plt>
0x4012eb <encripta_pin+55>      mov    $0x0,%eax
0x4012f0 <encripta_pin+60>      cmp    $0x7,%eax
0x4012f3 <encripta_pin+63>      jle    0x40131c <encripta_pin+104>
0x4012f5 <encripta_pin+65>      mov    %rsp,%rdi
0x4012f8 <encripta_pin+68>      mov    $0xa,%edx
0x4012fd <encripta_pin+73>      mov    $0x0,%esi
0x401302 <encripta_pin+78>      call   0x401120 <strtol@plt>
0x401307 <encripta_pin+83>      mov    0x8(%rsp),%rcx
0x40130c <encripta_pin+88>      xor    %fs:0x28,%rcx
0x401315 <encripta_pin+97>      je     0x401327 <encripta_pin+115>
0x401317 <encripta_pin+99>      call   0x4010f0 <__stack_chk_fail@plt>
0x40131c <encripta_pin+104>     movslq %eax,%rdx
0x40131f <encripta_pin+107>     add    %bl,(%rsp,%rdx,1)
0x401322 <encripta_pin+110>     add    $0x1,%eax
0x401325 <encripta_pin+113>     jmp    0x4012f0 <encripta_pin+60>
0x401327 <encripta_pin+115>     add    $0x10,%rsp
0x40132b <encripta_pin+119>     pop    %rbx
0x40132c <encripta_pin+120>     ret
```

Sin embargo, si lo analizamos más detenidamente, la gran mayoría de estos movimientos, que utilizan incluso elementos de la pila, no son más que una conversión de los números a elementos ascii, algo que está ahí simplemente para despistar, así pues, esta vez nos limitaremos a ver cómo repercute el encriptamiento en un prin de prueba y a realizar esos cambios de manera inversa sobre el pin encriptado.

```
0x40142d <main+256>             call   0x4012b4 <encripta_pin>
0x401432 <main+261>             cmp    %eax,0x2c40(%rip)        # 0x404078 <passcode>
0x401438 <main+267>             je     0x40143f <main+274>
```

De nuevo, aprovecharemos la comparación entre el pin encriptado y el introducido para obtener cual es el pin encriptado:

```
(gdb) p*(int)0x404078
$1 = 4859
```

Y a continuación, pondremos un breakpoint antes de encriptar el pin. Introduciremos como pin de prueba, “**5555”**, que podemos observar que se encuentra en %rdi

```
rcx            0x0                 0
rdi            0x15b3              5555
r8             0x1999999999999999  1844674407370955161
r11            0x7ffff7f583c0      140737353450432
r14            0x0                 0
eflags         0x246               [ PF ZF IF ]
ds             0x0                 0
gs             0x0                 0
```

A continuación, tal y como indica el código de la última imagen, llamará a la función encripta_pin, por lo que nos dará el resultado de nuestro pin introducido (5555) en el registro %rax.

```
rax            0x1e61              7777
rdx            0x0                 0
rbp            0x0                 0x0
r9             0x0                 0
r12            0x401170            4198768
r15            0x0                 0
cs             0x33                51
es             0x0                 0
```

Podemos observar que efectivamente la conversión a ascii estaba ahí solo para hacer más complejo el encriptamiento (además de así poder operar con cada dígito por separado). De este modo hemos descubierto que lo único que realmente hace es sumar 2 a cada dígito del pin. Finalmente, aplicaremos la lógica inversa y restaremos 2 a cada dígito del pin encriptado, de este modo:

4859 → 2637.

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: ubuntu12
Pin: 2637

```
albertoplaza@AlbertoUbuntu:~/Universidad/EC/bomba kitano$ ./bomba_ismael

Introduce la contraseña: ubuntu12

Introduce el pin: 2637

.........................
... bomba desactivada ...
.........................
```