Desactivando bomba de Guillermo Gomez Trenado

Lo primero que haremos será ejecutar el **gdb** para poder depurar el codigo ensamblador de la bomba de nuestro compañero. Para eso utilizaremos el siguiente comando:

```
juanka1995@juanka-desktop ~/Downloads/guillermo $ gdb bomba
```

Una vez aquí hacemos un **break** sobre el **main**, que es la parte que nos interesa analizar. Lanzamos el ejecutable.

Untitled 1

```
(gdb) break main
Breakpoint 1 at 0x80486f1
(gdb) run
```

Una vez aquí, lo primero que deberiamos hacer es buscar la funcion que se encarga de cifrar nuestra password introducida, para analizar paso a paso el algoritmo usado por nuestro compañero y asi poder desactivar su bomba.

```
0x0804874a <+92>:    mov    %eax,(%esp)
0x0804874d <+95>:    call   0x804860d <encryptPassword>
0x08048752 <+100>:   movl   $0x804a040,(%esp)
```

Una vez localizada la dirección de memoria en la que se aloja dicha funcion, realizamos un **break** a esta y **continuamos** con la ejecución del programa. Esto supondrá la solicitud de introducir la contraseña, en mi caso introducire **tupadre** como contraseña.

```
(gdb) break *0x804860d
Breakpoint 2 at 0x804860d
(gdb) continue
Continuing.
Introduce la contraseña: tupadre

Breakpoint 2, 0x0804860d in encryptPassword ()
```

Una vez el gdb se detenga el breakpoint tendremos a nuestra vista todo el codigo ensamblador de la función que contiene el algoritmo de cifrado.

```
=> 0x0804860d <+0>:     push   %ebp
   0x0804860e <+1>:     mov    %esp,%ebp
   0x08048610 <+3>:     sub    $0x28,%esp
   0x08048613 <+6>:     mov    0x8(%ebp),%eax
   0x08048616 <+9>:     mov    %eax,(%esp)
   0x08048619 <+12>:    call   0x80484d0 <strlen@plt>
   0x0804861e <+17>:    sub    $0x1,%eax
   0x08048621 <+20>:    mov    %eax,-0xc(%ebp)
   0x08048624 <+23>:    movl   $0x0,-0x10(%ebp)
   0x0804862b <+30>:    jmp    0x804865b <encryptPassword+78>
   0x0804862d <+32>:    mov    -0x10(%ebp),%edx
   0x08048630 <+35>:    mov    0x8(%ebp),%eax
   0x08048633 <+38>:    add    %edx,%eax
   0x08048635 <+40>:    movzbl (%eax),%eax
   0x08048638 <+43>:    mov    %al,-0x11(%ebp)
   0x0804863b <+46>:    mov    -0x10(%ebp),%edx
   0x0804863e <+49>:    mov    0x8(%ebp),%eax
   0x08048641 <+52>:    add    %eax,%edx
   0x08048643 <+54>:    movzbl -0x11(%ebp),%eax
   0x08048647 <+58>:    shl    $0x4,%eax
   0x0804864a <+61>:    mov    %eax,%ecx
   0x0804864c <+63>:    movzbl -0x11(%ebp),%eax
   0x08048650 <+67>:    shr    $0x4,%al
   0x08048653 <+70>:    or     %ecx,%eax
   0x08048655 <+72>:    mov    %al,(%edx)
   0x08048657 <+74>:    addl   $0x1,-0x10(%ebp)
   0x0804865b <+78>:    mov    -0x10(%ebp),%eax
   0x0804865e <+81>:    cmp    -0xc(%ebp),%eax
   0x08048661 <+84>:    jb     0x804862d <encryptPassword+32>
   0x08048663 <+86>:    leave
   0x08048664 <+87>:    ret
```

Si nos fijamos bien en la siguiente linea, se hace una llamada a la funcion **strlen** la cual será la encargada de calcular la longitud de la cadena introducida (**tupadre**). Salvará este valor que es **7** en la pila y tambien salvará el valor **0** en pila. Despues de esto comenzara un bucle que iterara **7** veces para recorrer toda nuestra cadena.

```
   0x08048619 <+12>:    call   0x80484d0 <strlen@plt>
=> 0x0804861e <+17>:    sub    $0x1,%eax
   0x08048621 <+20>:    mov    %eax,-0xc(%ebp)
   0x08048624 <+23>:    movl   $0x0,-0x10(%ebp)
   0x0804862b <+30>:    jmp    0x804865b <encryptPassword+78>
```

```
(gdb) print $eax
$2 = 7
```

En este paso se inicializará el registro **$eax** a 0. Será el que actue como índice. En **-0xc(%ebp)** es donde estará guardado el numero **7** y se realizará una iteracíon en el bucle por cada vez que **$eax** sea distinto de **7**.

```
=> 0x0804865b <+78>:    mov    -0x10(%ebp),%eax
   0x0804865e <+81>:    cmp    -0xc(%ebp),%eax
   0x08048661 <+84>:    jb     0x804862d <encryptPassword+32>
```

Lo siguiente que se hará es situarnos en la primera letra de nuestra cadena, en nuestro caso en la letra '**t**', cuyo valor guardaremos en el registro $**eax**.

Esta letra en decimal está representada por el **116**, lo que en hexadecimal es el **74**. Esta información nos será de mucha utilidad mas adelante.

```
   0x0804862d <+32>:    mov    -0x10(%ebp),%edx
   0x08048630 <+35>:    mov    0x8(%ebp),%eax
   0x08048633 <+38>:    add    %edx,%eax
   0x08048635 <+40>:    movzbl (%eax),%eax
=> 0x08048638 <+43>:    mov    %al,-0x11(%ebp)
```

```
(gdb) print (char) $eax
$12 = 116 't'
```

En la ultima instrucción de la linea anterior (**%al,-0x11(%ebp)**) lo que hace es salvar el valor **116** (t) en pila. A continuacíon veremos por que hace esto.

Ahora carga el valor **116** almacenado en la pila en el registro **$eax** y le realiza un desplazamiento de 4 bits a la izquierda mediante la instrucción **shl $0x4,%eax** con lo que obtenemos el numero **1856** que equivale al **740** en hexadecimal. Seguidamente salva dicho valor en el registro **$ecx**.

```
   0x08048643 <+54>:    movzbl -0x11(%ebp),%eax
   0x08048647 <+58>:    shl    $0x4,%eax
=> 0x0804864a <+61>:    mov    %eax,%ecx
```

```
(gdb) print $eax
$17 = 1856
```

Ahora vuelve a realizar la misma operación pero con un desplazamiento a la derecha de 4 bits mediante la instrucción **shr $0x4,%al**. Con esto se obtiene el valor **7** que en hexadecimal es el **7** tambien curiosamente.

```
   0x0804864c <+63>:    movzbl -0x11(%ebp),%eax
   0x08048650 <+67>:    shr    $0x4,%al
```

```
(gdb) print $al
$18 = 7
```

En el siguiente paso lo que realiza es la operación **or** entre el valor obtenido con el desplazamiento a la derecha y el valor obtenido con el desplazamiento a la izquierda. Con esto obtiene el **71** que en hexadecimal equivale al **47**.

```
   0x08048653 <+70>:    or     %ecx,%eax
```

```
(gdb) print $al
$20 = 71
```

Por ultimo guarda el valor obtenido en la direccion de memoria donde estaba almacenada nuestra **'t'**. Quedando de la siguiente forma nuestra clave introducida.

```
=> 0x08048655 <+72>:    mov    %al,(%edx)
```

```
0xffffd0a8:     "tupadre\n"
(gdb) nexti
0x08048657 in encryptPassword ()
(gdb) x /sb $edx
0xffffd0a8:     "Gupadre\n"
```

De aquí obtenimos que lo que esta realizando nuestro compañero es darle la vuelta al numero hexadecimal que representa la letra **'t'**, el cual es el **74** y tras el algoritmo quedaría como **47** que equivale a la **'G'**.

Por tanto realizaría este cifrado para todas las letras de nuestra clave.

Ahora obtendremos la clave que nuestro compañero tiene cifrada (la buena para desactivar la bomba) para ello volvemos al codigo ensamblador del **main**, en la siguiente linea.

```
0x08048752 <+100>:   movl   $0x804a040,(%esp)
```

De la direccion de memoria **$0x804a040** es de donde se carga la contraseña cifrada. Si mostramos el contenido como un string obtendremos lo siguiente:

```
(gdb) x /sb 0x804a040
0x804a040 <password>:   "\026&'\026\066\026F\026&'\026\n"
```

Como podemos observar la contraseña parece "ilegible", pero esto se debe a que despues del algoritmo de cifrado utilizado por nuestro compañero algunos de los caracteres obtenidos pertenecen a **caracteres ASCII no imprimibles**. Para averiguarla tuve que recorrerla letra por letra sabiendo que su longitud es de **11** y consultando el valor ASCII de cada letra con el cifrado aplicado, realizar el descifrado (proceso de cifrado inverso) y anotando los resultado obtenidos.

Tras esto pude comprobar que la contraseña cifrada correspondia con **"abracadabra"**. Para ver como averigue cada letra pondre el ejemplo de la primera letra, que correspondería a la **"a"**.

```
(gdb) x /cb 0x804a040
0x804a040 <password>:   22 '\026'
```

En la imagen anterior podemos ver que el primer carácter de la **password** que como no es un carácter ASCII imprimible me aparece como **'\026'** cuyo valor en la tabla ASCII es el **22**. Si el valor **22** que esta en decimal lo pasamos a hexadecimal obtenemos el **16**. Bien si ahora a este valor le invertimos los numeros, es decir, ahora tendriamos el **61** en valor hexadecimal y lo pasamos a decimal obtenemos el **97** cuyo valor en la tabla ASCII equivale a la letra **'a'**.

DESCIFRANDO CODIGO DE SEGURIDAD

Ahora pasaremos a ver que cifrado utiliza nuestro compañero para el codigo x cifras. Lo primero que haremos será un **break** en la función que se encargará de cifrar el codigo que nosotros introduzcamos, para poder ver paso a paso cual es el algoritmo utilizado por nuestro compañero.

```
=> 0x08048665 <+0>:     push   %ebp
   0x08048666 <+1>:     mov    %esp,%ebp
   0x08048668 <+3>:     sub    $0x10,%esp
   0x0804866b <+6>:     mov    0x8(%ebp),%eax
   0x0804866e <+9>:     mov    (%eax),%eax
   0x08048670 <+11>:    mov    %eax,-0x4(%ebp)
   0x08048673 <+14>:    mov    -0x4(%ebp),%eax
   0x08048676 <+17>:    ror    $0x10,%eax
   0x08048679 <+20>:    mov    %eax,%edx
   0x0804867b <+22>:    mov    0x8(%ebp),%eax
   0x0804867e <+25>:    mov    %edx,(%eax)
   0x08048680 <+27>:    leave
   0x08048681 <+28>:    ret
```

Si nos fijamos bien podremos ver como lo que hace es usar la instrucción **'ror $0x10,%eax'** sobre nuestra clave introducida, con esto lo que consigue es rotar **16 bits** a la derecha nuestro numero introducido. Como un **integer** se almacena en **32 bits** lo que realmente está haciendo, es intercambiar los 16 bits de la derecha por los 16 bits de la izquierda, generando otro numero totalmente diferente.

```
   0x08048676 <+17>:    ror    $0x10,%eax
```

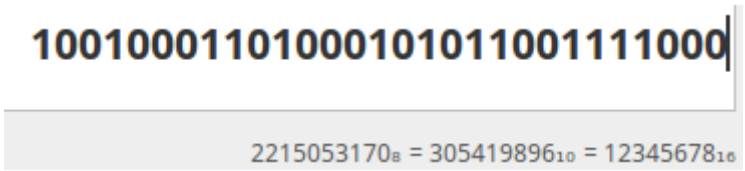
Ahora lo que deberemos hacer es buscar el momento en que carga su numero cifrado de una dirección de memoria y aplicarle el algoritmo a la inversa.

Si nos fijamos en la siguiente instrucción veremos que ahí es donde realiza la comparación de su código cifrado y del nuestro. En **$eax** tendremos su código cifrado y en **$edx** el nuestro.

```
   0x080487d0 <+226>:   call   0x8048665 <encryptCode>
   0x080487d5 <+231>:   mov    0x14(%esp),%edx
   0x080487d9 <+235>:   mov    0x804a050,%eax
=> 0x080487de <+240>:   cmp    %eax,%edx
```

```
(gdb) print $eax
$1 = 1450709556
(gdb) print $edx
$2 = 269615104
```

Si el numero **1450709556** lo pasamos a binario y rotamos 16 bits a la derecha obtendremos lo siguiente:



Y si ese numero lo pasamos a hexadecimal veremos que es el **12345678** (lo pasamos a hexadecimal debido a que en la llamada a **scanf** nuestro compañero almacena el **passcode** como un dato **hexadecimal**). Este sería el codigo de nuestro compañero.

Una vez sacado la contraseña (**abracadabra**) y el codigo (**12345678**) podriamos desactivar la bomba sin ningun problema.