Desactivando bomba de Guillermo Sandoval

Lo primero que deberemos hacer es ejecutar el **gdb** con la bomba de nuestro compañero. Creamos un **breakpoint** en el main y buscamos la función que se encarga de cifrar la password.

```
juanka1995@juanka-laptop ~/Downloads/guillermo_sandoval $ gdb bomba
```

```
(gdb) break main
Breakpoint 1 at 0x8048759
(gdb) run
Starting program: /home/juanka1995/Downloads/guillermo_sandoval/bomba
```

```
0x080487bc <+104>:   movl   $0x804a040,(%esp)
0x080487c3 <+111>:   call   0x80486f1 <encrypt>
0x080487c8 <+116>:   mov    %eax,%ebx
0x080487ca <+118>:   lea    0x28(%esp),%eax
0x080487ce <+122>:   mov    %eax,(%esp)
0x080487d1 <+125>:   call   0x80486f1 <encrypt>
0x080487d6 <+130>:   mov    %esi,0x8(%esp)
```

Si nos fijamos bien, veremos que nuestro compañero realiza **dos** llamadas a la función **encrypt**. Gracias a esto he podido descubrir que el compañero no está realizando bien el ejercicio, ya que en la primera llamada a la función **encrypt** cifra **su** password y en la segunda llamada cifra la que nosotros introducimos.

De esta forma podríamos mirar la dirección de memoria en la que esta almacenada la **password** sin cifrar y pasariamos la primera parte de la bomba.

```
0x080487bc <+104>:   movl   $0x804a040,(%esp)
0x080487c3 <+111>:   call   0x80486f1 <encrypt>
```

```
(gdb) print (char*) 0x804a040
$1 = 0x804a040 <password> "mandalorian\n"
```

De todas formas, voy a entrar dentro de la funcion **encrypt** para averiguar que algoritmo usa nuestro compañero, aunque para la desactivación de la bomba no sea necesario debido al fallo de nuestro compañero. Para ello haremos un nuevo **break** e introducire como password **tupadre** para ver que sucede con nuestra clave.

```
(gdb) break *0x80486f1
Breakpoint 2 at 0x80486f1
```

```
(gdb) conti
Continuing.
Introduce la contraseña: tupadre
```

A continuación mostraré el codigo ensamblador de la funcion **encrypt** de nuestro compañero.

```
Dump of assembler code for function encrypt:
=> 0x080486f1 <+0>:     push   %ebp
   0x080486f2 <+1>:     mov    %esp,%ebp
   0x080486f4 <+3>:     sub    $0x10,%esp
   0x080486f7 <+6>:     movl   $0x0,-0x4(%ebp)
   0x080486fe <+13>:    jmp    0x8048740 <encrypt+79>
   0x08048700 <+15>:    mov    -0x4(%ebp),%eax
   0x08048703 <+18>:    and    $0x1,%eax
   0x08048706 <+21>:    test   %eax,%eax
   0x08048708 <+23>:    jne    0x8048724 <encrypt+51>
   0x0804870a <+25>:    mov    -0x4(%ebp),%edx
   0x0804870d <+28>:    mov    0x8(%ebp),%eax
   0x08048710 <+31>:    add    %eax,%edx
   0x08048712 <+33>:    mov    -0x4(%ebp),%ecx
   0x08048715 <+36>:    mov    0x8(%ebp),%eax
   0x08048718 <+39>:    add    %ecx,%eax
   0x0804871a <+41>:    movzbl (%eax),%eax
   0x0804871d <+44>:    add    $0x1,%eax
   0x08048720 <+47>:    mov    %al,(%edx)
   0x08048722 <+49>:    jmp    0x804873c <encrypt+75>
   0x08048724 <+51>:    mov    -0x4(%ebp),%edx
   0x08048727 <+54>:    mov    0x8(%ebp),%eax
   0x0804872a <+57>:    add    %eax,%edx
   0x0804872c <+59>:    mov    -0x4(%ebp),%ecx
   0x0804872f <+62>:    mov    0x8(%ebp),%eax
   0x08048732 <+65>:    add    %ecx,%eax
   0x08048734 <+67>:    movzbl (%eax),%eax
   0x08048737 <+70>:    sub    $0x1,%eax
   0x0804873a <+73>:    mov    %al,(%edx)
   0x0804873c <+75>:    addl   $0x1,-0x4(%ebp)
   0x08048740 <+79>:    mov    -0x4(%ebp),%edx
   0x08048743 <+82>:    mov    0x8(%ebp),%eax
   0x08048746 <+85>:    add    %edx,%eax
   0x08048748 <+87>:    movzbl (%eax),%eax
   0x0804874b <+90>:    test   %al,%al
   0x0804874d <+92>:    jne    0x8048700 <encrypt+15>
   0x0804874f <+94>:    mov    0x8(%ebp),%eax
   0x08048752 <+97>:    leave
   0x08048753 <+98>:    ret
```

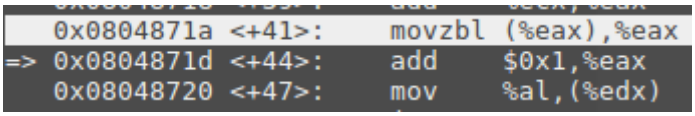
Veremos que ocurre con la primera letra que es la **‘t’** en mi caso, la cual carga en la instrucción siguiente:

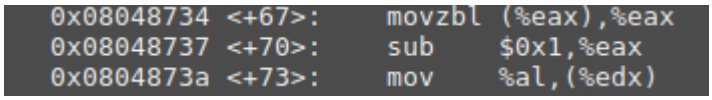
```
   0x08048746 <+85>:    add    %edx,%eax
   0x08048748 <+87>:    movzbl (%eax),%eax
```

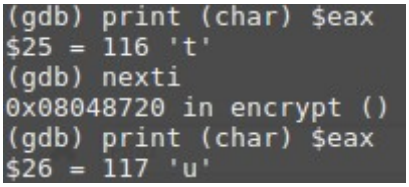
```
(gdb) print $eax
$7 = 116
(gdb) print (char) $eax
$8 = 116 't'
```

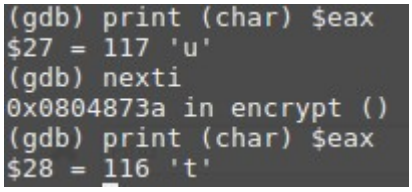
Si nos fijamos en la instrucción siguiente, veremos que en ese instante nuestro compañero tiene almacenado en **$eax** nuestra letra **‘t’**, y lo que hace seguidamente es **sumarle uno** con la instrucción **add $0x1,%eax**. Esto se debe a que previamente, en la instrucciónes que mostraré mas abajo realiza un filtro mirando si la **letra** se encuentra en una posición **par** o **impar** del **char***.

```
   0x080486fe <+13>:    jmp    0x8048740 <encrypt+79>
   0x08048700 <+15>:    mov    -0x4(%ebp),%eax
   0x08048703 <+18>:    and    $0x1,%eax
   0x08048706 <+21>:    test   %eax,%eax
```



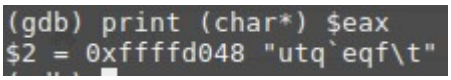






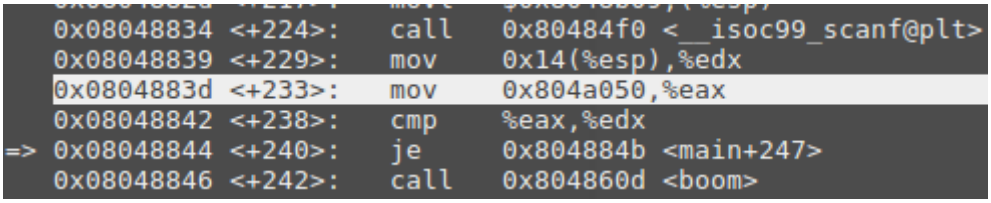
Si la letra se encuentra en una posición **par** lo que realiza nuestro compañero es **sumarle uno** a dicha letra obteniendo en la **'t'** una **'u'** (imágenes superior izquierda), y en caso de que la letra se encuentre en una posición **impar** lo que realiza nuestro compañero es **restarle uno** a dicha letra obteniendo en la letra **'u'** una **'t'** (imágenes superior derecha).

Por lo tanto la password que hemos introducido (**tupadre**) se encriptaría como **'utq`eqf\t'**.

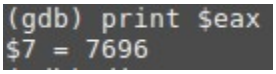


DESCIFRANDO CODIGO DE SEGURIDAD

Ahora pasaremos a ver que algoritmo de cifrado usa nuestro compañero para el **passcode**. Tras analizar un poco el codigo ensamblador, nos damos cuenta que nuestro compañero no realiza **ningún** tipo de cifrado para el **passcode** por lo que descifrarlo es tan facil como ver que hay en la posición de memoria donde este se almacena.



Si no fijamos está cargando de la dirección de memoria **0x804a050** en el registro **$eax**, por lo que después de esa instrucción si miramos el contenido de dicho registro tendríamos su **passcode**. Su **passcode** es el **7696**.



Teniendo toda esta informacion podríamos desactivar la bomba de nuestro compañero sin ningún problema.