

## Explicación bomba Jose Antonio Ruiz Millán.

```
0804860b <encripta>:
804860b: 55          push    %ebp
804860c: 89 e5      mov     %esp,%ebp
804860e: 83 ec 18   sub     $0x18,%esp
8048611: c7 45 f0 00 00 00 00 movl    $0x0,-0x10(%ebp)
8048618: eb 4b      jmp     8048665 <encripta+0x5a>
804861a: c7 45 f4 00 00 00 00 movl    $0x0,-0xc(%ebp)
8048621: eb 38      jmp     804865b <encripta+0x50>
8048623: 8b 55 f0   mov     -0x10(%ebp),%edx
8048626: 8b 45 08   mov     0x8(%ebp),%eax
8048629: 01 d0      add     %edx,%eax
804862b: 0f b6 00   movzbl (%eax),%eax
804862e: 3c 7a      cmp     $0x7a,%al
8048630: 75 0d      jne     804863f <encripta+0x34>
8048632: 8b 55 f0   mov     -0x10(%ebp),%edx
8048635: 8b 45 08   mov     0x8(%ebp),%eax
8048638: 01 d0      add     %edx,%eax
804863a: c6 00 61   movb    $0x61,(%eax)
804863d: eb 18      jmp     8048657 <encripta+0x4c>
804863f: 8b 55 f0   mov     -0x10(%ebp),%edx
8048642: 8b 45 08   mov     0x8(%ebp),%eax
8048645: 01 d0      add     %edx,%eax
8048647: 8b 4d f0   mov     -0x10(%ebp),%ecx
804864a: 8b 55 08   mov     0x8(%ebp),%edx
804864d: 01 ca      add     %ecx,%edx
804864f: 0f b6 12   movzbl (%edx),%edx
8048652: 83 c2 01   add     $0x1,%edx
8048655: 88 10      mov     %dl,(%eax)
8048657: 83 45 f4 01 addl    $0x1,-0xc(%ebp)
804865b: 83 7d f4 02 cmpl    $0x2,-0xc(%ebp)
804865f: 7e c2      jle     8048623 <encripta+0x18>
8048661: 83 45 f0 01 addl    $0x1,-0x10(%ebp)
8048665: 83 ec 0c   sub     $0xc,%esp
8048668: ff 75 08   pushl   0x8(%ebp)
804866b: e8 50 fe ff ff call    80484c0 <strlen@plt>
8048670: 83 c4 10   add     $0x10,%esp
8048673: 89 c2      mov     %eax,%edx
8048675: 8b 45 f0   mov     -0x10(%ebp),%eax
8048678: 39 c2      cmp     %eax,%edx
804867a: 77 9e      ja      804861a <encripta+0xf>
804867c: 90        nop
804867d: c9        leave
804867e: c3        ret
```

Pongo el código completo para una mejor búsqueda ahora al ir por partes.

### Carga Inicio del método encripta.

```
804860b: 55          push    %ebp
804860c: 89 e5       mov     %esp,%ebp
804860e: 83 ec 18    sub     $0x18,%esp
8048611: c7 45 f0 00 00 00 00 movl    $0x0,-0x10(%ebp)
8048618: eb 4b       jmp     8048665 <encripta+0x5a>
```

La primera carga del método se posiciona en la pila y añade un 0 en la posición -0x10 de ebp que sería una variable local.

Una vez cargados los primeros parámetros salta a la posición 8048665.

```
8048665: 83 ec 0c    sub     $0xc,%esp
8048668: ff 75 08    pushl   0x8(%ebp)
804866b: e8 50 fe ff ff call    80484c0 <strlen@plt>
8048670: 83 c4 10    add     $0x10,%esp
8048673: 89 c2       mov     %eax,%edx
8048675: 8b 45 f0    mov     -0x10(%ebp),%eax
8048678: 39 c2       cmp     %eax,%edx
804867a: 77 9e       ja      804861a <encripta+0xf>
```

Una vez estamos en la posición del salto vemos que llama a strlen para saber el tamaño y se almacena en %eax. Seguidamente mueve el contenido de %eax (tamaño) a %edx para después meter en %eax el valor de la variable local que guardamos en el primer cargado del método.

Comparamos ahora la variable local que tenía valor 0 al inicio con el tamaño, si es menor saltamos a 804861a y si no lo es vemos en la imagen del código completo que termina el método, por lo tanto podemos asimilar que tenemos un método que va a realizarse tantas veces como sea el tamaño.

```
804861a: c7 45 f4 00 00 00 00 movl    $0x0,-0xc(%ebp)
8048621: eb 38       jmp     804865b <encripta+0x50>
```

Si la variable local creada al principio es menor que el tamaño, saltamos a 804861a y vemos que lo que hace es crear una nueva variable local que también valdrá 0 y saltamos a 804865b.

```
804865b: 83 7d f4 02    cmpl    $0x2,-0xc(%ebp)
804865f: 7e c2       jle     8048623 <encripta+0x18>
```

Una vez saltamos, vemos que compara la variable local creada actualmente con un 2 y si es menor o igual saltamos a 8048623, en caso de no ser menor o igual vemos en el código general de arriba que suma 1 a la primera variable local y vuelve a comprobar el tamaño y vuelve de nuevo a realizar lo que estamos comentando.

```

8048623:    8b 55 f0      mov     -0x10(%ebp),%edx
8048626:    8b 45 08      mov     0x8(%ebp),%eax
8048629:    01 d0        add     %edx,%eax
804862b:    0f b6 00      movzbl (%eax),%eax
804862e:    3c 7a        cmp     $0x7a,%al
8048630:    75 0d        jne     804863f <encripta+0x34>

```

Una vez saltamos movemos el valor de la primera variable local que lleva el conteo del tamaño, en %edx, luego movemos el vector que pasamos como parámetro a %eax, y finalmente guardamos en %eax la suma del elemento con la posición en la que nos encontramos. Guardamos en %eax el contenido al que apunta esa variable y comparamos \$0x7a que equivale en la tabla ASCII al carácter 'z' con el elemento que tenemos del vector. Si no es igual saltamos a 804863f y si es igual hace lo siguiente:

```

8048632:    8b 55 f0      mov     -0x10(%ebp),%edx
8048635:    8b 45 08      mov     0x8(%ebp),%eax
8048638:    01 d0        add     %edx,%eax
804863a:    c6 00 61      movb    $0x61,(%eax)
804863d:    eb 18        jmp     8048657 <encripta+0x4c>

```

De nuevo cargamos el valor actual y ponemos el valor \$0x61 al elemento actual que este elemento en la tabla ASCII equivale al carácter 'a', por lo tanto tenemos que compara si el elemento actual es igual a 'z' y en el caso de que lo sea, pone el elemento igual a 'a'.

Vemos en el código completo que después de esto suma 1 a la segunda variable local que debe ser menor que 3 y vuelve a realizar lo mismo.

En el caso de que el elemento actual no fuese una 'z' vamos a ver que realiza.

```

804863f:    8b 55 f0      mov     -0x10(%ebp),%edx
8048642:    8b 45 08      mov     0x8(%ebp),%eax
8048645:    01 d0        add     %edx,%eax
8048647:    8b 4d f0      mov     -0x10(%ebp),%ecx
804864a:    8b 55 08      mov     0x8(%ebp),%edx
804864d:    01 ca        add     %ecx,%edx
804864f:    0f b6 12      movzbl (%edx),%edx
8048652:    83 c2 01      add     $0x1,%edx
8048655:    88 10        mov     %dl,(%eax)
8048657:    83 45 f4 01   addl    $0x1,-0xc(%ebp)
804865b:    83 7d f4 02   cmpl    $0x2,-0xc(%ebp)
804865f:    7e c2        jle     8048623 <encripta+0x18>

```

Podemos ver que lo primero que realiza es cargar tanto en %eax como %edx el elemento actual en el que nos encontramos. Carga el contenido del elemento en %edx y le suma 1 por lo tanto estamos pasando a la siguiente letra el elemento actual. Luego mueve el resultado a %eax y suma 1 a la segunda variable local que creamos. Finalmente compara si es menor o igual que 2 y si lo es vuelve a saltar a 8042623 que ya esta explicado mas arriba lo que realiza.

## Conclusión:

Tenemos que crea una variable local a 0 y que se va a realizar tantas veces como elementos tenga el array.

Luego vemos que para cada elemento del array creamos otra variable local que va desde 0 hasta 2 y por lo tanto deducimos que por cada elemento del array va a sumarle 1 al elemento en cada iteración a no ser que el elemento actual sea el 'z' y entonces lo pasa a ser el carácter 'a'.

En resumen tenemos que para cada carácter va ha sumarle 3 en la tabla ASCII teniendo en cuenta que cuando llegamos a la 'z' vuelve a la 'a' en vez de seguir por los números de la tabla ASCII.

## Desactivar la bomba:

Ejecutamos ddd con el ejecutable y vamos a la comprobación de las cadenas para comprobar la contraseña.

```
0x0804875c <main+93>: push %eax
0x0804875d <main+94>: call 0x804860b <encr
0x08048762 <main+99>: add $0x10,%esp
0x08048765 <main+102>: sub $0xc,%esp
0x08048768 <main+105>: push $0x804a03c
0x0804876d <main+110>: call 0x80484c0 <strl
0x08048772 <main+115>: add $0x10,%esp
0x08048775 <main+118>: sub $0x4,%esp
0x08048778 <main+121>: push %eax
0x08048779 <main+122>: push $0x804a03c
0x0804877e <main+127>: lea -0x70(%ebp),%eax
0x08048781 <main+130>: push %eax
0x08048782 <main+131>: call 0x80484f0 <strncmp@plt>
0x08048787 <main+136>: add $0x10,%esp
0x0804878a <main+139>: test %eax,%eax
0x0804878c <main+141>: je 0x8048793 <main+148>
0x0804878e <main+143>: call 0x804867f <boom>
0x08048793 <main+148>: sub $0x8,%esp
0x08048796 <main+151>: push $0x0
0x08048798 <main+153>: lea -0x78(%ebp),%eax
0x0804879b <main+156>: push %eax

(gdb) x /1sb $0x804a03c
Value can't be converted to integer.
(gdb) x /1sb 0x804a03c
0x804a03c <password>: "krodvrbnrvh"
```

Ya tenemos la contraseña y vemos que esta encriptada. Si realizamos la inversa al método que uso para cifrar y restamos 3 a cada carácter tenemos que la contraseña sería **holasoyjose**

Sabiendo esto avanzamos por el programa y paramos de nuevo en la comparación del código.

```
0x080487c0 <main+193>: call 0x8048460 <printf@plt>
0x080487c5 <main+198>: add $0x10,%esp
0x080487c8 <main+201>: sub $0x8,%esp
0x080487cb <main+204>: lea -0x84(%ebp),%eax
0x080487d1 <main+210>: push %eax
0x080487d2 <main+211>: push $0x8048954
0x080487d7 <main+216>: call 0x80484e0 <__isoc99_scanf@plt>
0x080487dc <main+221>: add $0x10,%esp
0x080487df <main+224>: nov -0x84(%ebp),%edx
0x080487e5 <main+230>: nov 0x804a048,%eax
0x080487ea <main+235>: cmp %eax,%edx
0x080487ec <main+237>: je 0x80487f3 <main+244>
0x080487ee <main+239>: call 0x804867f <boom>
0x080487f3 <main+244>: sub $0x8,%esp
0x080487f6 <main+247>: push $0x0
0x080487f8 <main+249>: lea -0x80(%ebp),%eax
0x080487fb <main+252>: push %eax
0x080487fc <main+253>: call 0x8048480 <gettimeofday@plt>
End of assembler dump.
```

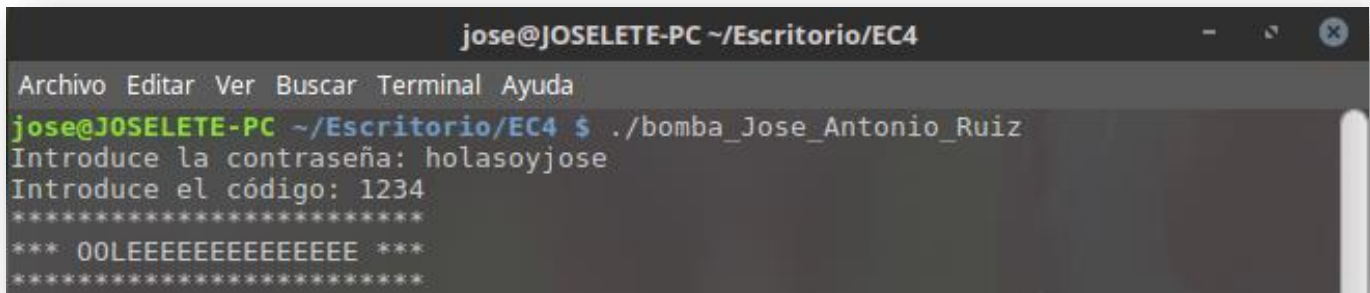
Registers		
eax	0x4d2	1234
ecx	0x1	1
edx	0xff88	65416
ebx	0x0	0
esp	0xffffd140	0xffffd140
ebp	0xffffd1c8	0xffffd1c8
esi	0xf7faa000	-134569984
edi	0xf7faa000	-134569984
eip	0x80487ea	0x80487ea <main+235>
eflags	0x282	[ SF IF ]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43

Integer registers All registers

Vemos en la imagen que tenemos en el registro %eax tenemos el código para desactivar la bomba y en %edx tenemos lo que hemos insertado nosotros al azar.

Asique finalmente tenemos que la contraseña es **holasoyjose** y el código es **1234**.

### Comprobación:



```
jose@JOSELETE-PC ~/Escritorio/EC4
Archivo Editar Ver Buscar Terminal Ayuda
jose@JOSELETE-PC ~/Escritorio/EC4 $ ./bomba_Jose_Antonio_Ruiz
Introduce la contraseña: holasoyjose
Introduce el código: 1234
*****
*** 00LEEEEEEEEEEEEEEE ***
*****
```

Vemos que la bomba se desactiva correctamente