

## Bomba Adrian Pelaez:

```
0804860d <encryptar_password>:
804860d: 55          push    %ebp
804860e: 89 e5      mov     %esp,%ebp
8048610: 83 ec 28   sub     $0x28,%esp
8048613: c7 45 f0 00 00 00 00 movl    $0x0,-0x10(%ebp)
804861a: 8b 45 08   mov     0x8(%ebp),%eax
804861d: 89 04 24   mov     %eax,(%esp)
8048620: e8 ab fe ff ff call    80484d0 <strlen@plt>
8048625: 83 e8 01   sub     $0x1,%eax
8048628: 89 45 f4   mov     %eax,-0xc(%ebp)
804862b: c7 45 f0 00 00 00 00 movl    $0x0,-0x10(%ebp)
8048632: eb 40      jmp     8048674 <encryptar_password+0x67>
8048634: 8b 45 f0   mov     -0x10(%ebp),%eax
8048637: 83 e0 01   and     $0x1,%eax
804863a: 85 c0      test    %eax,%eax
804863c: 75 1a      jne     8048658 <encryptar_password+0x4b>
804863e: 8b 55 f0   mov     -0x10(%ebp),%edx
8048641: 8b 45 08   mov     0x8(%ebp),%eax
8048644: 01 c2      add     %eax,%edx
8048646: 8b 4d f0   mov     -0x10(%ebp),%ecx
8048649: 8b 45 08   mov     0x8(%ebp),%eax
804864c: 01 c8      add     %ecx,%eax
804864e: 0f b6 00   movzbl (%eax),%eax
8048651: 83 c0 02   add     $0x2,%eax
8048654: 88 02      mov     %al,(%edx)
8048656: eb 18      jmp     8048670 <encryptar_password+0x63>
8048658: 8b 55 f0   mov     -0x10(%ebp),%edx
804865b: 8b 45 08   mov     0x8(%ebp),%eax
804865e: 01 c2      add     %eax,%edx
8048660: 8b 4d f0   mov     -0x10(%ebp),%ecx
8048663: 8b 45 08   mov     0x8(%ebp),%eax
8048666: 01 c8      add     %ecx,%eax
8048668: 0f b6 00   movzbl (%eax),%eax
804866b: 83 c0 01   add     $0x1,%eax
804866e: 88 02      mov     %al,(%edx)
8048670: 83 45 f0 01 addl    $0x1,-0x10(%ebp)
8048674: 8b 45 f0   mov     -0x10(%ebp),%eax
8048677: 3b 45 f4   cmp     -0xc(%ebp),%eax
804867a: 7c b8      jl      8048634 <encryptar_password+0x27>
804867c: c9        leave
804867d: c3        ret
```

```
0804867e <encryptar_code>:
804867e: 55          push    %ebp
804867f: 89 e5      mov     %esp,%ebp
8048681: 83 ec 10   sub     $0x10,%esp
8048684: 8b 45 08   mov     0x8(%ebp),%eax
8048687: 89 45 fc   mov     %eax,-0x4(%ebp)
804868a: 83 6d fc 7b subl    $0x7b,-0x4(%ebp)
804868e: 8b 45 fc   mov     -0x4(%ebp),%eax
8048691: c9        leave
8048692: c3        ret
```

## Metodo encriptar password:

```
804860d: 55          push    %ebp
804860e: 89 e5       mov     %esp,%ebp
8048610: 83 ec 28    sub     $0x28,%esp
8048613: c7 45 f0 00 00 00 00 movl    $0x0,-0x10(%ebp)
804861a: 8b 45 08     mov     0x8(%ebp),%eax
804861d: 89 04 24     mov     %eax,(%esp)
8048620: e8 ab fe ff ff call    80484d0 <strlen@plt>
8048625: 83 e8 01     sub     $0x1,%eax
8048628: 89 45 f4     mov     %eax,-0xc(%ebp)
804862b: c7 45 f0 00 00 00 00 movl    $0x0,-0x10(%ebp)
8048632: eb 40       jmp     8048674 <encriptar_password+0x67>
```

Lo primero que podemos ver en la carga del método es que crea una variable local inicializada a 0 y se almacena en `-0x10(%ebp)` y también calcula el tamaño del array con `strlen` y almacena ese valor restándole 1 en otra variable local almacenada en `-0xc(%ebp)`. Una vez realizado esto, salta a 8048674.

```
8048674: 8b 45 f0     mov     -0x10(%ebp),%eax
8048677: 3b 45 f4     cmp     -0xc(%ebp),%eax
804867a: 7c b8       jle     8048634 <encriptar_password+0x27>
```

Mueve el valor de la variable local inicializada a 0 al registro `%eax` para compararlo con el valor que guardamos (`strlen-1`) y si es `<=` saltamos a 8048634. En caso contrario el método terminaría como vemos en el código completo.

```
8048634: 8b 45 f0     mov     -0x10(%ebp),%eax
8048637: 83 e0 01     and     $0x1,%eax
804863a: 85 c0       test    %eax,%eax
804863c: 75 1a       jne     8048658 <encriptar_password+0x4b>
```

Mueve el valor de la variable tenemos almacenada en `-0x10(%ebp)` al registro `%eax`. Realiza un AND de 1 con este registro y hace un test. Estos pasos se resumen en saber si el numero es par o impar. Si en numero es par no salta y sigue con las instrucciones y si es impar salta a 8048658.

Por lo tanto tenemos que vamos a ir recorriendo el vector comparando si el elemento actual esta en una posición par o impar. Vamos a ver que pasaría si la posición es impar.

```
8048658: 8b 55 f0     mov     -0x10(%ebp),%edx
804865b: 8b 45 08     mov     0x8(%ebp),%eax
804865e: 01 c2       add     %eax,%edx
8048660: 8b 4d f0     mov     -0x10(%ebp),%ecx
8048663: 8b 45 08     mov     0x8(%ebp),%eax
8048666: 01 c8       add     %ecx,%eax
8048668: 0f b6 00     movzbl  (%eax),%eax
804866b: 83 c0 01     add     $0x1,%eax
804866e: 88 02       mov     %al,(%edx)
8048670: 83 45 f0 01 addl    $0x1,-0x10(%ebp)
8048674: 8b 45 f0     mov     -0x10(%ebp),%eax
8048677: 3b 45 f4     cmp     -0xc(%ebp),%eax
804867a: 7c b8       jle     8048634 <encriptar_password+0x27>
```

Vemos que guarda el valor de la variable local inicializada a 0 que lleva el conteo del elemento por el que vamos, en `%ecx` y el array en `%eax` para sumar estos 2 valores y obtener el valor actual del elemento en el que nos encontramos.

Vemos que después suma 1 al elemento actual y después suma 1 a la variable que controla el numero de elemento por el que vamos. Después pasa a la posición 8048674 que ya la tenemos explicada mas arriba. **Tenemos finalmente que si es impar suma 1 al elemento.**

Vamos ahora a comprobar que pasa cuando el elemento es par:

```
804863e: 8b 55 f0      mov     -0x10(%ebp),%edx
8048641: 8b 45 08      mov     0x8(%ebp),%eax
8048644: 01 c2        add     %eax,%edx
8048646: 8b 4d f0      mov     -0x10(%ebp),%ecx
8048649: 8b 45 08      mov     0x8(%ebp),%eax
804864c: 01 c8        add     %ecx,%eax
804864e: 0f b6 00     movzbl  (%eax),%eax
8048651: 83 c0 02     add     $0x2,%eax
8048654: 88 02        mov     %al,(%edx)
8048656: eb 18        jmp     8048670 <encriptar_password+0x63>
```

Si es par realiza la misma carga que anteriormente para obtener el elemento actual en el que nos encontramos. A este elemento le suma 2 y salta a la posición 8048670 que podemos ver en el código completo que lo que hace es sumar 1 a la variable que controla el numero de elemento por el que vamos y después pasa a la posición 8048674 que ya la tenemos explicada mas arriba.

**Tenemos finalmente que si el numero es par, le suma 2 al elemento actual.**

Metodo encriptar code:

```
0804867e <encriptar_code>:
804867e: 55          push    %ebp
804867f: 89 e5       mov     %esp,%ebp
8048681: 83 ec 10    sub     $0x10,%esp
8048684: 8b 45 08    mov     0x8(%ebp),%eax
8048687: 89 45 fc    mov     %eax,-0x4(%ebp)
804868a: 83 6d fc 7b subl     $0x7b,-0x4(%ebp)
804868e: 8b 45 fc    mov     -0x4(%ebp),%eax
8048691: c9         leave  %eax
8048692: c3         ret
```

Podemos ver que lo realiza es restarle  $0x7b = 123$  al numero que representa el código.



Buscando contraseña con ddd:

The screenshot shows a debugger window with assembly code on the left and a 'DDD: Examine Memory' window on the right. The assembly code is for a function named 'main' and includes instructions like 'call', 'movl', 'leal', and 'je'. The 'DDD: Examine Memory' window shows 'Examine 1' with 'string' and 'bytes' tabs. The 'from' field is set to '0x804a040'. Below the assembly code, a breakpoint is set at '0x08048782 in main ()' with the command '(gdb) x /1sb 0x804a040'. The output shows the password 'KMqwgFE' stored in memory.

```
0x08048729 <main+42>: call 0x8048480 <gettimeofday@plt>
0x0804872e <main+47>: movl $0x8048924, (%esp)
0x08048735 <main+54>: call 0x8048460 <printf@plt>
0x0804873a <main+59>: mov 0x804a050, %eax
0x0804873f <main+64>: mov %eax, 0x8(%esp)
0x08048743 <main+68>: movl $0x64, 0x4(%esp)
0x0804874b <main+76>: lea 0x28(%esp), %eax
0x0804874f <main+80>: mov %eax, (%esp)
0x08048752 <main+83>: call 0x8048470 <fgets@plt>
0x08048757 <main+88>: lea 0x28(%esp), %eax
0x0804875b <main+92>: mov %eax, (%esp)
0x0804875e <main+95>: call 0x804860d <encrypt@plt>
0x08048763 <main+100>: movl $0x804a040, (%esp)
0x0804876a <main+107>: call 0x80484d0 <strlen@plt>
0x0804876f <main+112>: mov %eax, 0x8(%esp)
0x08048773 <main+116>: movl $0x804a040, 0x4(%esp)
0x0804877b <main+124>: lea 0x28(%esp), %eax
0x0804877f <main+128>: mov %eax, (%esp)
0x08048782 <main+131>: call 0x8048500 <strncmp@plt>
0x08048787 <main+136>: test %eax, %eax
0x08048789 <main+138>: je 0x8048790 <main+145>
0x0804878b <main+140>: call 0x8048693 <boom>
```

breakpoint 1, 0x08048782 in main ()  
(gdb) x /1sb 0x804a040  
0x804a040 <password>: "KMqwgFE"

Podemos ver como la contraseña **cifrada es KMqwgFE**. Utilizando la inversa del método de cifrar que hemos averiguado, **obtenemos la contraseña ILoveEC**.

```
jose@JOSELETE-PC ~/Escritorio/bomba_adri $ ./bomba_adrian_pelaez
Introduce la contraseña: ILoveEC
Introduce el código: 
```

Podemos ver que al introducir la contraseña pasamos correctamente sin que explote.



## Buscando code con ddd:

The screenshot shows the ddd debugger interface. On the left, the assembly code is displayed with addresses from 0x080487ae to 0x080487fa. The instruction at 0x080487ea is `mov 0x804a04c,%eax`, where the address `0x804a04c` is highlighted with a red box. On the right, the 'Registers' window shows the state of various registers. The `eax` register contains the value `0x73b1851`, with `1851` highlighted by a red box. Below the registers, there are buttons for 'Integer registers' and 'All registers', and at the bottom, 'Close' and 'Help' buttons.

```
0x080487ae <main+175>:  mov    %edx,%eax
0x080487b0 <main+177>:  cmp     $0x5,%eax
0x080487b3 <main+180>:  jle     0x80487ba <main+187>
0x080487b5 <main+182>:  call    0x8048693 <boom>
0x080487ba <main+187>:  movl    $0x804893f,(%esp)
0x080487c1 <main+194>:  call    0x8048460 <printf@plt>
0x080487c6 <main+199>:  lea     0x10(%esp),%eax
0x080487ca <main+203>:  mov     %eax,0x4(%esp)
0x080487ce <main+207>:  movl    $0x8048956,(%esp)
0x080487d5 <main+214>:  call    0x80484f0 <__isoc99_scanf@plt>
0x080487da <main+219>:  mov     0x10(%esp),%eax
0x080487de <main+223>:  mov     %eax,0x4(%esp)
0x080487e1 <main+226>:  call    0x804867e <encryptar_code>
0x080487e6 <main+231>:  mov     %eax,0x14(%esp)
0x080487ea <main+235>:  mov     0x804a04c,%eax
0x080487ef <main+240>:  cmp     %eax,0x14(%esp)
0x080487f3 <main+244>:  je      0x80487fa <main+251>
0x080487f5 <main+246>:  call    0x8048693 <boom>
0x080487fa <main+251>:  movl    $0x0,0x4(%esp)
End of assembler dump.
```

eax	0x73b1851
ecx	0x1
edx	0xf7fab87c -134563716
ebx	0x0
esp	0xffffd130 0xffffd130
ebp	0xffffd1c8 0xffffd1c8
esi	0xf7faa000 -134569984
edi	0xf7faa000 -134569984
eip	0x80487ef 0x80487ef <main+240>
eflags	0x212 [ AF IF ]
cs	0x23 35
ss	0x2b 43
ds	0x2b 43

Integer registers All registers

Close Help

Vemos que guarda el código cifrado en el registro `%eax` y vemos que ese registro almacena el valor **1851**. Si aplicamos la inversa del algoritmo descubierto **obtenemos que el código es 1974**.

```
jose@JOSELETE-PC ~/Escritorio/bomba adri $ ./bomba_adrian_pelaez
Introduce la contraseña: ILoveEC
Introduce el código: 1974
*****
*** bomba desactivada ***
*****
```

Como vemos, introducimos los datos obtenidos y la bomba se desactiva correctamente.