Bomba Julio Fresneda

```
0804868b <cifrar_password>:
 804868b:       55                      push   %ebp
 804868c:       89 e5                   mov    %esp,%ebp
 804868e:       83 ec 18                sub    $0x18,%esp
 8048691:       c7 45 f4 00 00 00 00    movl   $0x0,-0xc(%ebp)
 8048698:       eb 26                   jmp    80486c0 <cifrar_password+0x35>
 804869a:       8b 45 f4                mov    -0xc(%ebp),%eax
 804869d:       83 e0 01                and    $0x1,%eax
 80486a0:       85 c0                   test   %eax,%eax
 80486a2:       75 18                   jne    80486bc <cifrar_password+0x31>
 80486a4:       8b 55 f4                mov    -0xc(%ebp),%edx
 80486a7:       8b 45 08                mov    0x8(%ebp),%eax
 80486aa:       01 d0                   add    %edx,%eax
 80486ac:       8b 4d f4                mov    -0xc(%ebp),%ecx
 80486af:       8b 55 08                mov    0x8(%ebp),%edx
 80486b2:       01 ca                   add    %ecx,%edx
 80486b4:       0f b6 12                movzbl (%edx),%edx
 80486b7:       83 c2 05                add    $0x5,%edx
 80486ba:       88 10                   mov    %dl,(%eax)
 80486bc:       83 45 f4 01             addl   $0x1,-0xc(%ebp)
 80486c0:       83 ec 0c                sub    $0xc,%esp
 80486c3:       68 3c a0 04 08          push   $0x804a03c
 80486c8:       e8 f3 fd ff ff          call   80484c0 <strlen@plt>
 80486cd:       83 c4 10                add    $0x10,%esp
 80486d0:       8d 50 ff                lea    -0x1(%eax),%edx
 80486d3:       8b 45 f4                mov    -0xc(%ebp),%eax
 80486d6:       39 c2                   cmp    %eax,%edx
 80486d8:       77 c0                   ja     804869a <cifrar_password+0xf>
 80486da:       90                      nop
 80486db:       c9                      leave
 80486dc:       c3                      ret
```

```
080486dd <cifrar_passcode>:
 80486dd:       55                      push   %ebp
 80486de:       89 e5                   mov    %esp,%ebp
 80486e0:       8b 45 08                mov    0x8(%ebp),%eax
 80486e3:       05 39 30 00 00          add    $0x3039,%eax
 80486e8:       5d                      pop    %ebp
 80486e9:       c3                      ret
```

Método Cifrar Password:

```
804868b:       55                          push   %ebp
804868c:       89 e5                       mov    %esp,%ebp
804868e:       83 ec 18                    sub    $0x18,%esp
8048691:       c7 45 f4 00 00 00 00        movl   $0x0,-0xc(%ebp)
8048698:       eb 26                       jmp    80486c0 <cifrar_password+0x35>
```

Podemos comrpobar que realiza la carga del método y crea una variable local que la guarda en -0xc(%ebp) y que su valor será 0. Cuando realiza esto, salta directamente a 80486c0.

```
80486c0:       83 ec 0c                    sub    $0xc,%esp
80486c3:       68 3c a0 04 08              push   $0x804a03c
80486c8:       e8 f3 fd ff ff              call   80484c0 <strlen@plt>
80486cd:       83 c4 10                    add    $0x10,%esp
80486d0:       8d 50 ff                    lea    -0x1(%eax),%edx
80486d3:       8b 45 f4                    mov    -0xc(%ebp),%eax
80486d6:       39 c2                       cmp    %eax,%edx
80486d8:       77 c0                       ja     804869a <cifrar_password+0xf>
```

Vemos que hace una llamada a strlen para calcular el tamaño que finalmente guarda ese valor en %edx, luego guarda en %eax el valor de la variable local creada anteriormente. Finalmente compara esos dos valores, en el caso de que la variable local(%eax) sea mayor que el valor de srtlen(%edx) saldríamos del método como se puede ver en el código completo, y en caso contrario, saltamos a 804869a.

```
804869a:       8b 45 f4                    mov    -0xc(%ebp),%eax
804869d:       83 e0 01                    and    $0x1,%eax
80486a0:       85 c0                       test   %eax,%eax
80486a2:       75 18                       jne    80486bc <cifrar_password+0x31>
```

Una vez estamos en esta posición es porque el valor de la variable local que controla el número de veces que realizaremos este paso que es el igual al valor de strlen. Vemos que guarda el valor actual de la variable local en el registro %eax y reliza un AND sobre este valor. Una vez realizado pruebas sobre este paso comprobé que lo que consigues con esto es que si el numero es par, el resultado es 0 y si el numero es impar, no da 0. Seguidamente realiza la instrucción test sobre el resultado y si no son iguales *(el numero es impar)* salta a 80486bc que podemos ver en el código completo que suma 1 a la variable local y ya sigue con 80486c0 que podemos ver su explicación justo encima de esta. En el caso de que el numero sea par, pasaría lo siguiente:

```
80486a4:       8b 55 f4                    mov    -0xc(%ebp),%edx
80486a7:       8b 45 08                    mov    0x8(%ebp),%eax
80486aa:       01 d0                       add    %edx,%eax
80486ac:       8b 4d f4                    mov    -0xc(%ebp),%ecx
80486af:       8b 55 08                    mov    0x8(%ebp),%edx
80486b2:       01 ca                       add    %ecx,%edx
80486b4:       0f b6 12                    movzbl (%edx),%edx
80486b7:       83 c2 05                    add    $0x5,%edx
80486ba:       88 10                       mov    %dl,(%eax)
80486bc:       83 45 f4 01                 addl   $0x1,-0xc(%ebp)
80486c0:       83 ec 0c                    sub    $0xc,%esp
```

Vemos que guarda en %ecx el valor de la variable local, y en %ecx el array y los suma para colocarse en el elemento actual. Copia el valor actual a %edx **y le suma 5,** suma 1 a la variable local y vuelve a 80486c0 que la tenemos explicada justo arriba.

Tenemos finalmente que este algoritmo suma 5 a los elementos pares del array.

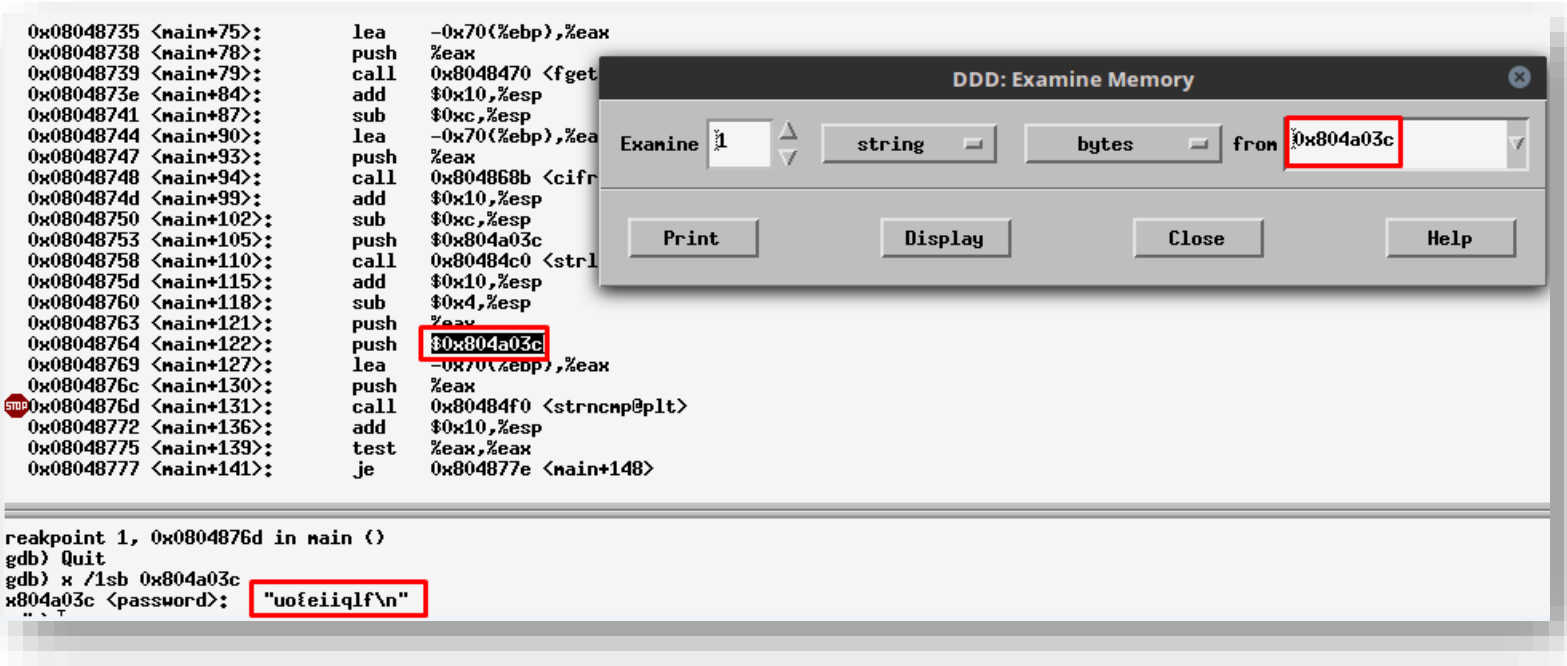
Metodo cifrar code:

```
080486dd <cifrar_passcode>:
 80486dd:       55                      push   %ebp
 80486de:       89 e5                   mov    %esp,%ebp
 80486e0:       8b 45 08                mov    0x8(%ebp),%eax
 80486e3:       05 39 30 00 00          add    $0x3039,%eax
 80486e8:       5d                      pop    %ebp
 80486e9:       c3                      ret
```

Podemos ver que lo que realiza únicamente es sumarle 0x3039 = 12345 al numero.

Por lo tanto tenemos que suma 12345 al valor del código.

Buscando la contraseña con ddd:

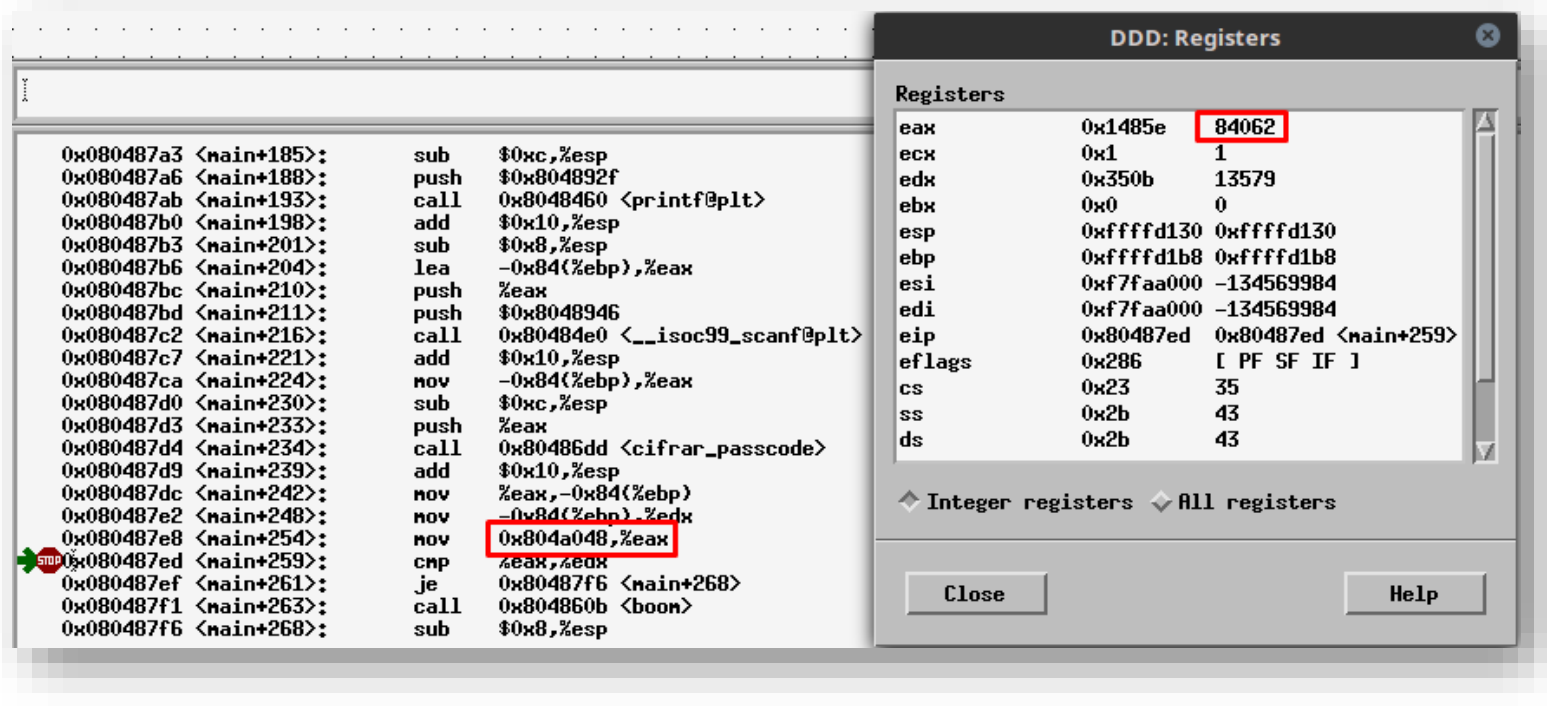


Podemos ver que el valor de la contraseña **cifrada** es uo{eiiqlf. Usandola inversa del algoritmo sacado anteriormente obtenemos **que la contraseña es povedilla.**

```
jose@JOSELETE-PC ~/Escritorio/bomba julio $ ./bomba_Julio_Fresneda
Introduce la contraseña: povedilla
Introduce el código:
```

Podemos ver que efectivamente al escribir el código, pasa y no explota la bomba.

Buscando code con ddd:



Podemos ver como guarda el valor del code **cifrado** en %eax y mirando el valor de ese registro, obtenemos que el code es **84062.** Utilizando la inversa de su cifrado obtenemos **que el código es 71717.**

```
jose@JOSELETE-PC ~/Escritorio/bomba julio $ ./bomba_Julio_Fresneda
Introduce la contraseña: povedilla
Introduce el código: 71717
**************************
*** bomba desactivada ***
**************************
```

Finalmente vemos como poniendo tanto la contraseña como el código que hemos sacado, conseguimos desactivar la bomba.