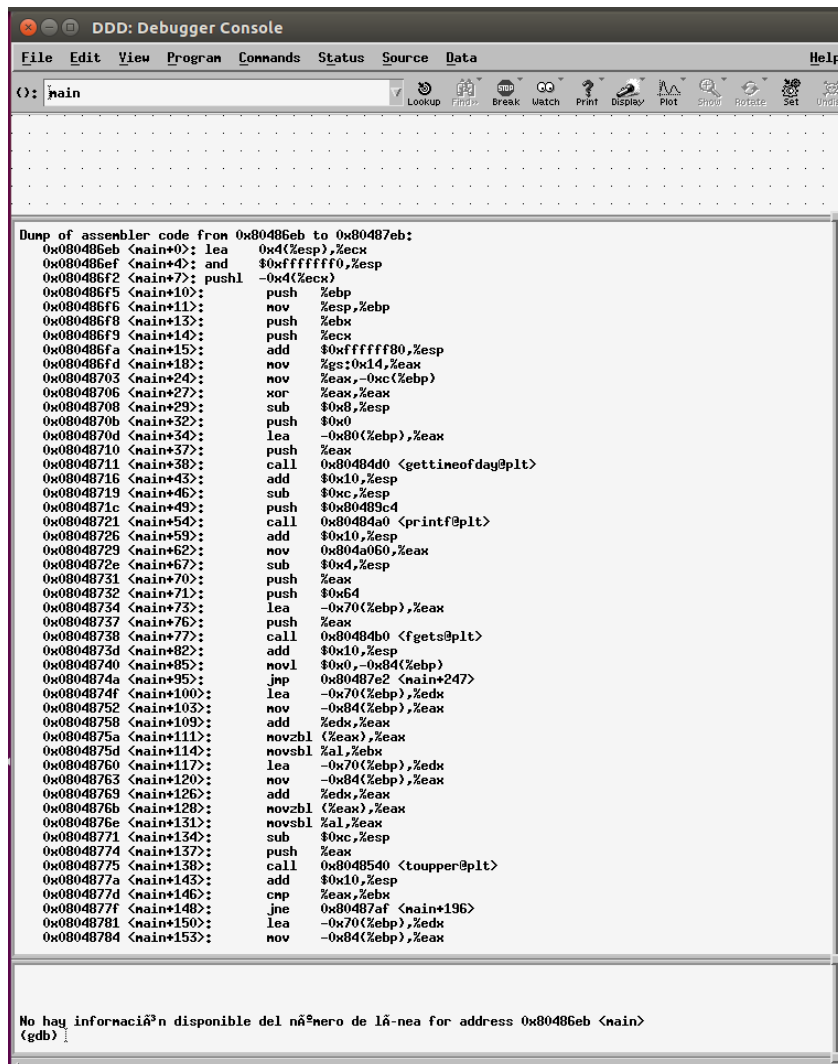


Descripción de MI BOMBA

Hacemos la primera visualización del código en ensamblador y obtenemos:



```
DDD: Debugger Console
File Edit View Program Commands Status Source Data Help
(): main
LookUp Find Break Watch Print Display Plot Step Rotate Undo

Dump of assembler code from 0x80486eb to 0x80487eb:
0x080486eb <main+0>: lea    0x4(%esp),%ecx
0x080486ef <main+4>: and    $0xffffffff,%esp
0x080486f2 <main+7>: pushl  -0x4(%ecx)
0x080486f5 <main+10>: push   %ebp
0x080486f6 <main+11>: mov    %esp,%ebp
0x080486f8 <main+13>: push   %ebx
0x080486f9 <main+14>: push   %ecx
0x080486fa <main+15>: add    $0xffffffff,%esp
0x080486fd <main+18>: mov    %gs:0x14,%eax
0x08048703 <main+24>: mov    %eax,-0xc(%ebp)
0x08048706 <main+27>: xor    %eax,%eax
0x08048708 <main+29>: sub    $0x8,%esp
0x0804870b <main+32>: push   $0x0
0x0804870d <main+34>: lea    -0x80(%ebp),%eax
0x08048710 <main+37>: push   %eax
0x08048711 <main+38>: call   0x8048d40 <gettimeofday@plt>
0x08048716 <main+43>: add    $0x10,%esp
0x08048719 <main+46>: sub    $0xc,%esp
0x0804871c <main+49>: push   $0x80483c4
0x08048721 <main+54>: call   0x8048d40 <printf@plt>
0x08048726 <main+59>: add    $0x10,%esp
0x08048729 <main+62>: mov    0x804a060,%eax
0x0804872e <main+67>: sub    $0x4,%esp
0x08048731 <main+70>: push   %eax
0x08048732 <main+71>: push   $0x64
0x08048734 <main+73>: lea    -0x70(%ebp),%eax
0x08048737 <main+76>: push   %eax
0x08048738 <main+77>: call   0x8048d40 <fgets@plt>
0x0804873d <main+82>: add    $0x10,%esp
0x08048740 <main+85>: movl   $0x0,-0x84(%ebp)
0x0804874a <main+95>: jmp     0x80487e2 <main+247>
0x0804874f <main+100>: lea    -0x70(%ebp),%edx
0x08048752 <main+103>: mov    -0x84(%ebp),%eax
0x08048758 <main+109>: add    %edx,%eax
0x0804875a <main+111>: movzbl (%eax),%eax
0x0804875d <main+114>: movsbl %al,%ebx
0x08048760 <main+117>: lea    -0x70(%ebp),%edx
0x08048763 <main+120>: mov    -0x84(%ebp),%eax
0x08048769 <main+126>: add    %edx,%eax
0x0804876b <main+128>: movzbl (%eax),%eax
0x0804876e <main+131>: movsbl %al,%eax
0x08048771 <main+134>: sub    $0xc,%esp
0x08048774 <main+137>: push   %eax
0x08048775 <main+138>: call   0x8048540 <toupper@plt>
0x0804877a <main+143>: add    $0x10,%esp
0x0804877d <main+146>: cmp    %eax,%ebx
0x0804877f <main+148>: jne     0x80487af <main+196>
0x08048781 <main+150>: lea    -0x70(%ebp),%edx
0x08048784 <main+153>: mov    -0x84(%ebp),%eax

No hay informaci³n disponible del n³mero de l³nea for address 0x80486eb <main>
(gdb)
```

Menos mal que es mi propia bomba y sé como funciona, vamos a empezar a poner un punto de ruptura en la linea main+77 que es donde obtiene la cadena que introducimos por teclado, y desde hay vamos ir viendo los pasos que realiza.

- Realiza el primer salto si encuentra paridad o la paridad es nula, y salta a la linea main+247, en este caso realiza el salto.
- Ahora en las siguientes lineas, inicializa las variables necesarias para un bucle, que se registra en el registro eax y edx.
- También realiza un nuevo salto en la linea main+272, (ja) que es el encargado de salta si la bandera de acarreo o de cero esta desactivada. También realiza el salto

```

Dump of assembler code from 0x80487ee to 0x80488ee:
=> 0x080487ee <main+259>:    add    $0x10,%esp
0x080487f1 <main+262>:    mov    %eax,%edx
0x080487f3 <main+264>:    mov    -0x84(%ebp),%eax
0x080487f9 <main+270>:    cmp    %eax,%edx
0x080487fb <main+272>:    ja     0x804874f <main+100>
0x08048801 <main+278>:    sub    $0xc,%esp
0x08048804 <main+281>:    push   $0x804a048
0x08048809 <main+286>:    call   0x8048520 <strlen@plt>
0x0804880e <main+291>:    add    $0x10,%esp
0x08048811 <main+294>:    sub    $0x4,%esp
0x08048814 <main+297>:    push   %eax
0x08048815 <main+298>:    push   $0x804a048
0x0804881a <main+303>:    lea     -0x70(%ebp),%eax
0x0804881d <main+306>:    push   %eax
0x0804881e <main+307>:    call   0x8048560 <strncmp@plt>
0x08048823 <main+312>:    add    $0x10,%esp
0x08048826 <main+315>:    test   %eax,%eax
0x08048828 <main+317>:    je     0x804882f <main+324>
0x0804882a <main+319>:    call   0x804866b <boom>
0x0804882f <main+324>:    sub    $0x8,%esp
0x08048832 <main+327>:    push   $0x0
0x08048834 <main+329>:    lea     -0x78(%ebp),%eax
0x08048837 <main+332>:    push   %eax
0x08048838 <main+333>:    call   0x80484d0 <gettimeofday@plt>
0x0804883d <main+338>:    add    $0x10,%esp
0x08048840 <main+341>:    mov     -0x78(%ebp),%edx
0x08048843 <main+344>:    mov     -0x80(%ebp),%eax
0x08048846 <main+347>:    sub     %eax,%edx
0x08048848 <main+349>:    mov     %edx,%eax
0x0804884a <main+351>:    cmp     $0x14,%eax
0x0804884d <main+354>:    jle     0x8048854 <main+361>
0x0804884f <main+356>:    call    0x804866b <boom>
0x08048854 <main+361>:    sub     $0xc,%esp
0x08048857 <main+364>:    push    $0x80489df
0x0804885c <main+369>:    call    0x80484a0 <printf@plt>
0x08048861 <main+374>:    add     $0x10,%esp
0x08048864 <main+377>:    sub     $0x8,%esp
0x08048867 <main+380>:    lea     -0x88(%ebp),%eax
0x0804886d <main+386>:    push    %eax
0x0804886e <main+387>:    push    $0x80489f6
0x08048873 <main+392>:    call    0x8048550 <__isoc99_scanf@plt>
0x08048878 <main+397>:    add     $0x10,%esp
0x0804887b <main+400>:    mov     -0x88(%ebp),%eax
0x08048881 <main+406>:    add     $0x1681,%eax
0x08048886 <main+411>:    mov     %eax,-0x88(%ebp)
0x0804888c <main+417>:    mov     -0x88(%ebp),%edx
0x08048892 <main+423>:    mov     0x804a054,%eax
0x08048897 <main+428>:    cmp     %eax,%edx
0x08048899 <main+430>:    je      0x80488a0 <main+437>

```

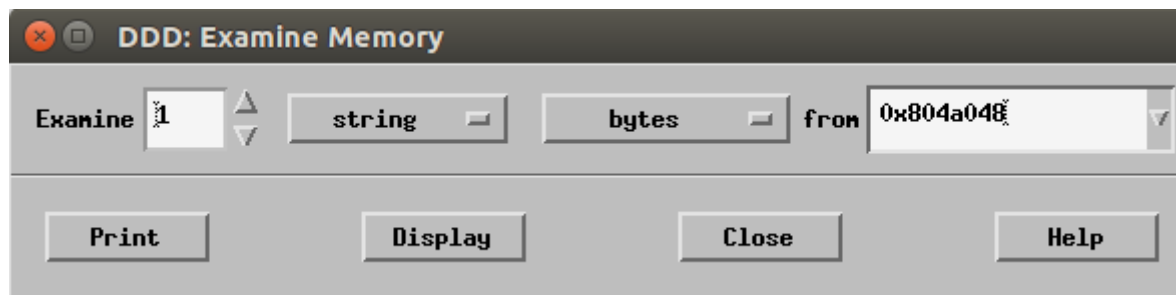
- iv. Resumiendo un poco obtenemos la longitud de la cadena introducida por teclado, con la instrucción de llamada strlen de la linea main+286. Que nos servirá para poder recorrer la cadena introducida.
- v. Con la instrucción lea obtenemos el primer elemento de la cadena, es decir, los dos primeros bytes del array, que vamos leyendo del la posicion de memoria de la pila y lo almacenamos en eax. En la que realizamos la llamada a la función toupper (función que devuelve el carácter de la posición indicada en mayúscula).
- vi. Aquí podemos ver un engorro, ya que realiza la llamada a toupper varias veces y luego la llamada a la funcion tolower, (función contraria a toupper, es decir, convierte un carácter de mayúscula a minúscula). Esto se resume fácilmente en el siguiente proceso:
 1. Primero comprueba el primer carácter de la cadena introducida por teclado es mayúscula, en caso de que sea cierto, realiza un salto y convierte ese carácter en mayúscula, si fuera falso, realiza el proceso contrario, convierte el carácter de mayúscula a minúscula.
- vii. Una vez que ha recorrido la cadena introducida, incluyendo la interacción del carácter “\n”, realiza una instrucción máquina interesante:

```

0x080487f1 <main+262>:    mov    %eax,%edx
0x080487f3 <main+264>:    mov    -0x84(%ebp),%eax
0x080487f9 <main+270>:    cmp    %eax,%edx
0x080487fb <main+272>:    ja     0x804874f <main+100>
0x08048801 <main+278>:    sub    $0xc,%esp
0x08048804 <main+281>:    push   $0x804a048
0x08048809 <main+286>:    call   0x8048520 <strlen@plt>
0x0804880e <main+291>:    add    $0x10,%esp

```

En esta dirección de memoria encontramos:



```
(gdb) nexti
0x08048804 in main ()
(gdb) x /1sb 0x804a048
0x804a048 <password>: "sacalacabra"
(gdb) |
```

▲ B 0x804a048

Ya hemos encontrado la password almacenada en el programa.

En conclusión de este programa, sabemos que la palabra que introducimos, la convierte en mayúscula si es minúscula, y viceversa, si es minúscula la convierte en mayúscula, carácter a carácter de toda la cadena.

Como hemos encontrado la palabra clave almacenada en memoria “sacalacabra”, debemos introducir “SACALACABRA” para que la convierta en minúscula y desactive la bomba.

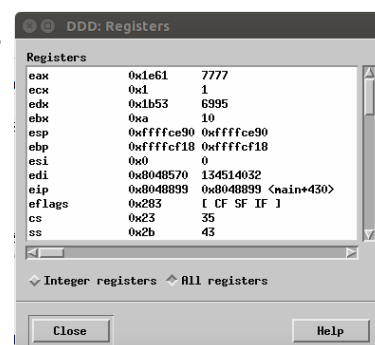
- Pasamos al código numérico.

Ahora nos vamos a la linea main+392, y ponemos hay el punto de ruptura, como anteriormente, nos vamos a la instrucción que obtiene el código numérico introducido por teclado.



Ahora es más facil, ya que:

1. Lo primero que hace es almacenar en eax lo que introducimos por teclado.
2. Le agrega o le suma a este registro 0x1681, en decimal, le suma 5671.
3. Para salvarlo lo vuelve almacenar el resultado anterior en el registro edx, para posteriormente...
4. Almacenar en eax el código numérico almacenado en el programa para desactivar la bomba.
5. Compara ambos registros, y si son iguales se desactiva la bomba y sino... BOOM!!



En este ejemplo anterior, a priori hemos introducido “1234”, por lo que a 1234 le suma 5671 = 6995. Como el código numérico almacenado es 7777, debemos introducir un numero, al que le sumemos 5671 sea igual a 7777.

Para llevar acaba la correcta desactivación debemos de introducir:

```
marcos@ubuntu: ~/Escritorio/1º Cuatrimestre 2ºC/EC/Practicas/practica4/2 Ficheros fuente
marcos@ubuntu:~/Escritorio/1º Cuatrimestre 2ºC/EC/Practicas/practica4/2 Ficheros
fuente$ ./bomba
Introduce la contraseña: SACALACABRA
Introduce el código: 2016
*****
*** bomba desactivada ***
*****
marcos@ubuntu:~/Escritorio/1º Cuatrimestre 2ºC/EC/Practicas/practica4/2 Ficheros
fuente$
```