



UNIVERSIDAD DE GRANADA

GRADO INGENIERÍA INFORMÁTICA (2017 – 2018)

ESTRUCTURA DE LOS COMPUTADORES

Practica 4: Bombas

Trabajo realizado por Antonio Miguel Morillo Chica

1. Mi bomba.

Para mostrar como desencriptar mi bomba lo haré como lo haría cualquier alumno.

Al principio unicamente hay que abrir el ejecutable con el ddd, tras esto lo más representativo del código es que existen dos funciones claras donde seguramente estén los passcode y password, que son: `contrasenia_numerica` y `contrasenia_caracteres`.

1.1. Contrasenias_caracteres

Una vez hecho esto he puesto breakpoints antes de estas funciones para poder entrar a ellas usando stepi y ver su código. El código asm de `contrasenia_caracteres` es el siguiente:

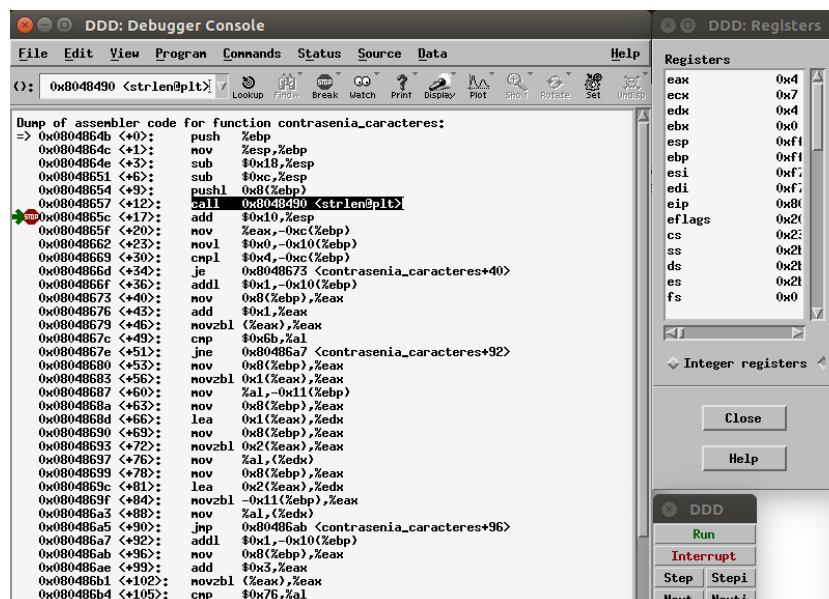
```
0x0804864b <+0>:    push    %ebp
0x0804864c <+1>:    mov     %esp,%ebp
0x0804864e <+3>:    sub     $0x18,%esp
0x08048651 <+6>:    sub     $0xc,%esp
0x08048654 <+9>:    pushl   0x8(%ebp)
0x08048657 <+12>:   call    0x8048490 <strlen@plt>
0x0804865c <+17>:   add     $0x10,%esp
0x0804865f <+20>:   mov     %eax,-0xc(%ebp)
0x08048662 <+23>:   movl    $0x0,-0x10(%ebp)
0x08048669 <+30>:   cmpl    $0x4,-0xc(%ebp)
0x0804866d <+34>:   je      0x8048673 <contrasenia_caracteres+40>
0x0804866f <+36>:   addl    $0x1,-0x10(%ebp)
0x08048673 <+40>:   mov     0x8(%ebp),%eax
0x08048676 <+43>:   add     $0x1,%eax
0x08048679 <+46>:   movzbl  (%eax),%eax
0x0804867c <+49>:   cmp     $0x6b,%al
0x0804867e <+51>:   jne     0x80486a7 <contrasenia_caracteres+92>
0x08048680 <+53>:   mov     0x8(%ebp),%eax
0x08048683 <+56>:   movzbl  0x1(%eax),%eax
0x08048687 <+60>:   mov     %al,-0x11(%ebp)
0x0804868a <+63>:   mov     0x8(%ebp),%eax
0x0804868d <+66>:   lea     0x1(%eax),%edx
0x08048690 <+69>:   mov     0x8(%ebp),%eax
0x08048693 <+72>:   movzbl  0x2(%eax),%eax
0x08048697 <+76>:   mov     %al,(%edx)
0x08048699 <+78>:   mov     0x8(%ebp),%eax
0x0804869c <+81>:   lea     0x2(%eax),%edx
0x0804869f <+84>:   movzbl  -0x11(%ebp),%eax
0x080486a3 <+88>:   mov     %al,(%edx)
0x080486a5 <+90>:   jmp     0x80486ab <contrasenia_caracteres+96>
0x080486a7 <+92>:   addl    $0x1,-0x10(%ebp)
0x080486ab <+96>:   mov     0x8(%ebp),%eax
0x080486ae <+99>:   add     $0x3,%eax
0x080486b1 <+102>:  movzbl  (%eax),%eax
0x080486b4 <+105>:  cmp     $0x76,%al
0x080486b6 <+107>:  jne     0x80486cc <contrasenia_caracteres+129>
0x080486b8 <+109>:  mov     0x8(%ebp),%eax
0x080486bb <+112>:  add     $0x1,%eax
0x080486be <+115>:  movb    $0x69,(%eax)
```

```

0x080486c1 <+118>: mov 0x8(%ebp),%eax
0x080486c4 <+121>: add $0x3,%eax
0x080486c7 <+124>: movb $0x65,(%eax)
0x080486ca <+127>: jmp 0x80486d0 <contrasenia_caracteres+133>
0x080486cc <+129>: addl $0x1,-0x10(%ebp)
0x080486d0 <+133>: cmpl $0x39,-0xc(%ebp)
0x080486d4 <+137>: je 0x80486dc <contrasenia_caracteres+145>
0x080486d6 <+139>: cmpl $0x4c,-0xc(%ebp)
0x080486da <+143>: jne 0x80486e3 <contrasenia_caracteres+152>
0x080486dc <+145>: mov $0xffffffff,%eax
0x080486e1 <+150>: jmp 0x804871b <contrasenia_caracteres+208>
0x080486e3 <+152>: mov 0x8(%ebp),%eax
0x080486e6 <+155>: movzbl (%eax),%eax
0x080486e9 <+158>: cmp $0x6d,%al
0x080486eb <+160>: jne 0x8048714 <contrasenia_caracteres+201>
0x080486ed <+162>: mov 0x8(%ebp),%eax
0x080486f0 <+165>: add $0x1,%eax
0x080486f3 <+168>: movzbl (%eax),%eax
0x080486f6 <+171>: cmp $0x69,%al
0x080486f8 <+173>: jne 0x8048714 <contrasenia_caracteres+201>
0x080486fa <+175>: mov 0x8(%ebp),%eax
0x080486fd <+178>: add $0x2,%eax
0x08048700 <+181>: movzbl (%eax),%eax
0x08048703 <+184>: cmp $0x6b,%al
0x08048705 <+186>: jne 0x8048714 <contrasenia_caracteres+201>
0x08048707 <+188>: mov 0x8(%ebp),%eax
0x0804870a <+191>: add $0x3,%eax
0x0804870d <+194>: movzbl (%eax),%eax
0x08048710 <+197>: cmp $0x65,%al
0x08048712 <+199>: je 0x8048718 <contrasenia_caracteres+205>
0x08048714 <+201>: addl $0x1,-0x10(%ebp)
0x08048718 <+205>: mov -0x10(%ebp),%eax
0x0804871b <+208>: leave
0x0804871c <+209>: ret

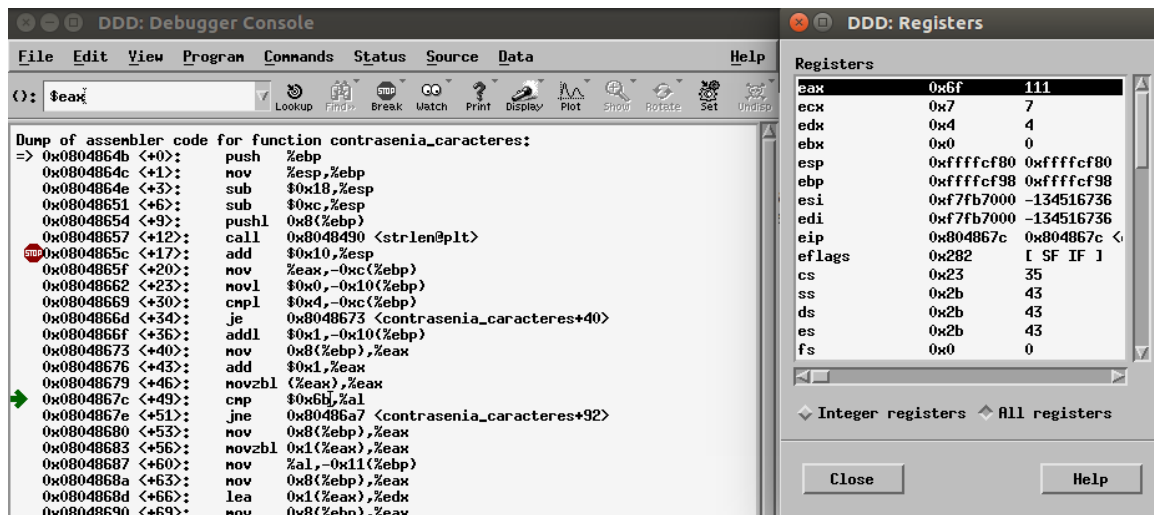
```

Como podemos ver en la línea +12 de la función se llama a strlen que devolverá el tamaño de la cadena introducida y será guardado en eax por convención, como podemos ver en la imagen:



En las siguientes líneas del ensamblador lo que se hace es mover el contenido de `eax` a una posición en concreto para después comparar lo con el valor 4 por lo que sabemos que la contraseña tiene 4 caracteres. En caso de que `eax - 4` sea 0 se saltará a la línea 40 y no se incrementará el valor de una variable.

Tras este punto lo más significativo es en la línea 49 donde hay un `compare` entre el registro `%al` y el valor `$0x6b` (107 en decimal) como podemos ver:



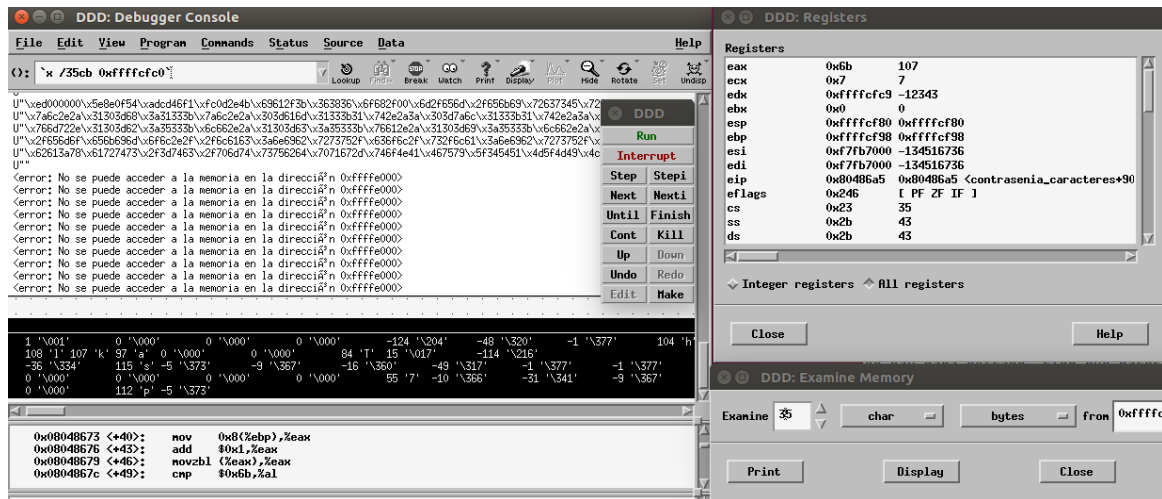
Si se produce el salto `jne` después del `compare` en este punto perderemos el código que existe entre las líneas 51 y 92 ya que no podremos verlo. En este punto me he dado cuenta de que el valor `$0x6b` es un valor muy importante, además el registro `%al` en 32 bits tiene el tamaño suficiente para que dentro coja un char por lo que seguramente la comparación es de un carácter. Por lo que, como en esta ejecución yo he usado la palabra “hola” tenemos:

- Valor inmediato (`$0x6b`): Que corresponde a 107 y en ascii a la letra `k`.
- Registro (`%eax`, `%al`): Como he escrito `hola`, el carácter `o` es 111, como se puede ver en la imagen.

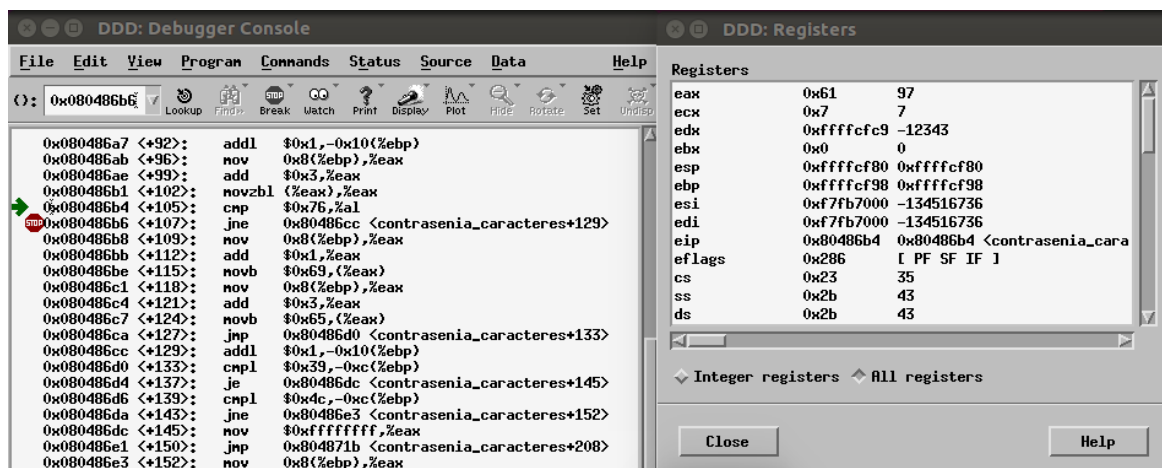
Lo que quiero decir esto es que la “o” debería de ser una “k” ya que sino nos saltaremos las líneas 51 – 92, por lo que haremos una ejecución con “hkla” para poder ejecutar el intervalo de las líneas 51 – 92.

Entre las líneas 53 – 92 lo que ocurre es una especie de swap entre los valores, si no fijamos en la línea 88, finalmente algo se guarda en (`%edx`), si

miramos lo que contiene es una posición de memoria que la podemos examinar con Data → Memory donde encontraremos lo siguiente, la palabra ha sido modificada, dejándola como “hlka”, fíjese en la caja subrayada de negro:



Posteriormente en la línea 105 se vuelve a comparar %al con un valor, en este caso el es el 76 en hex que es 166 en decimal y en ASCII la “v”. Como en este instante lo que tenemos en %al es 97 que corresponde a la letra “a” quiere decir que nuestra a debería de ser una “v” es decir, ahora en una nueva ejecución probaré con “hklv”. Esto lo sabemos con certeza ya que el código entre las líneas 129 y 150 se ejecutaría por el salto de la línea 107 para salir de la función ya que tras esto habría un salto incondicional al leave. En la siguiente imagen podemos ver lo dicho arriba:



Otro punto interesante es en la línea 158 donde ahora vuelve a compararse el valor \$6d con %al, en nuestro %al vale 0x68 que es la “h” por lo que ahora compara nuestra primera letra con lo que debería haber sido, una “m”. Así que usaremos para la siguiente ejecución “mklv”.

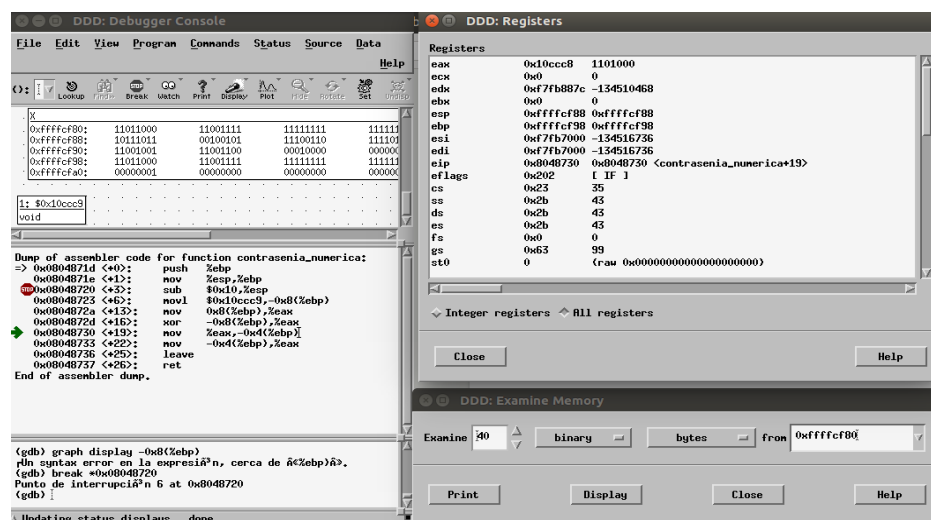
Si se realiza otra ejecución la contraseña mklv es válida ya que al final de la función lo que hace es ir comparando lo que hay en memoria (donde se guardaba la contraseña introducida) con los valores inmediatos: \$m \$i \$k \$e.

1.2. Contraseña_caracteres

El código de esta función es muy simple como podemos ver:

```
0x0804871d <+0>: push  %ebp
0x0804871e <+1>: mov   %esp,%ebp
0x08048720 <+3>: sub   $0x10,%esp
0x08048723 <+6>: movl  $0x10ccc9,-0x8(%ebp)
0x0804872a <+13>: mov   0x8(%ebp),%eax
0x0804872d <+16>: xor   -0x8(%ebp),%eax
0x08048730 <+19>: mov   %eax,-0x4(%ebp)
0x08048733 <+22>: mov   -0x4(%ebp),%eax
0x08048736 <+25>: leave
0x08048737 <+26>: ret
```

En esta función únicamente lo que se hace es hacer unos movimientos con los datos y registro y hacer un xor. La instrucción de la línea 16 lo hace con la posición de memoria de 8(%ebp) pero como podemos ver antes ha introducido en esta posición lo siguiente: \$10ccc9 que en decimal es 1101001, por lo que en %eax que estará la contraseña introducida por el usuario debería de ser este número.



Finalmente pruebo ambas cosas y contraseñas y el resultado es el siguiente:

```
mike@l00per: ~/Escritorio/Bomba/Entrega exe
mike@l00per:~/Escritorio/Bomba/Entrega exe$ ./bomba_MorilloChica_AntonioMiguel
Introduce la contraseña: mklv
Introduce el código BINARIO: 1101001
.....
..... Bomba desactivada .....
.....
mike@l00per:~/Escritorio/Bomba/Entrega exe$ █
```

2. Bomba de Natalia Fernandez Martinez

El código de la alumna no presenta funciones como en mi bomba, todo se desarrola sobre el main y es el siguiente:

```
0x08048679 <+0>:    push    %ebp
0x0804867a <+1>:    mov     %esp,%ebp
0x0804867c <+3>:    and     $0xffffffff,%esp
0x0804867f <+6>:    sub     $0x90,%esp
0x08048685 <+12>:   mov     %gs:0x14,%eax
0x0804868b <+18>:   mov     %eax,0x8c(%esp)
0x08048692 <+25>:   xor     %eax,%eax
0x08048694 <+27>:   movl    $0x0,0x4(%esp)
0x0804869c <+35>:   lea     0x18(%esp),%eax
0x080486a0 <+39>:   mov     %eax,(%esp)
0x080486a3 <+42>:   call    0x8048480 <gettimeofday@plt>
0x080486a8 <+47>:   movl    $0x8048884,(%esp)
0x080486af <+54>:   call    0x8048460 <printf@plt>
0x080486b4 <+59>:   mov     0x804a050,%eax
0x080486b9 <+64>:   mov     %eax,0x8(%esp)
0x080486bd <+68>:   movl    $0x64,0x4(%esp)
0x080486c5 <+76>:   lea     0x28(%esp),%eax
0x080486c9 <+80>:   mov     %eax,(%esp)
0x080486cc <+83>:   call    0x8048470 <fgets@plt>
0x080486d1 <+88>:   movl    $0x804a040,(%esp)
0x080486d8 <+95>:   call    0x80484d0 <strlen@plt>
0x080486dd <+100>:  mov     %eax,0x8(%esp)
0x080486e1 <+104>:  movl    $0x804a040,0x4(%esp)
0x080486e9 <+112>:  lea     0x28(%esp),%eax
```

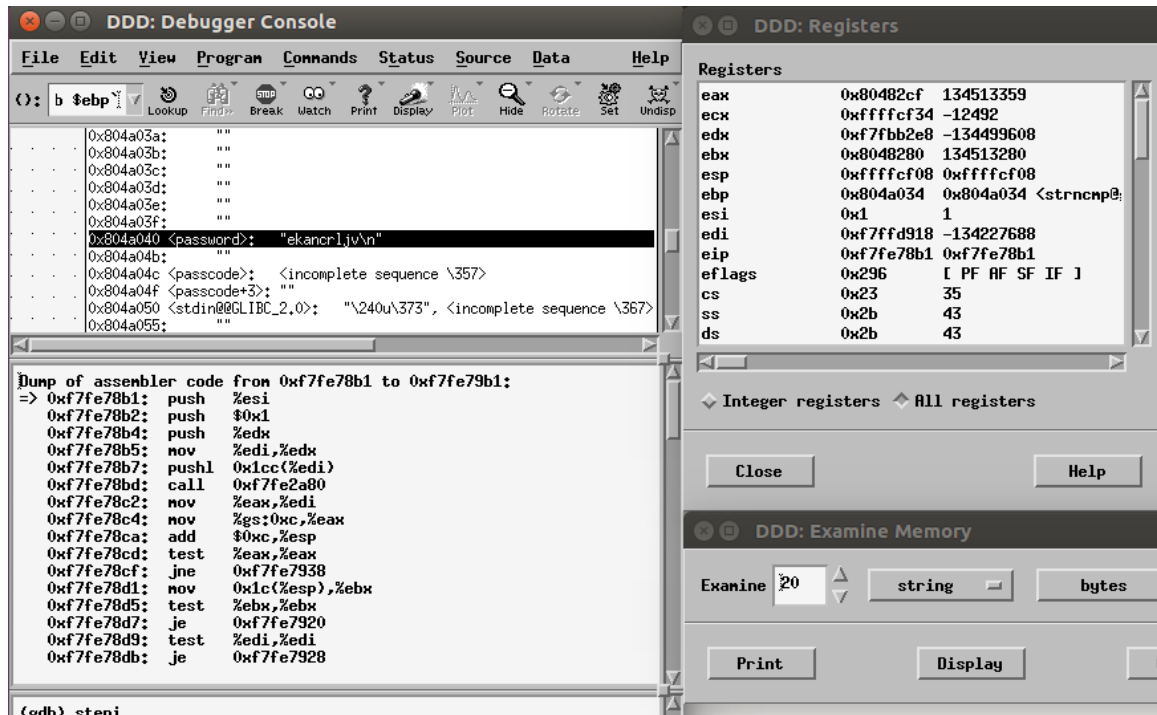
```

0x080486ed <+116>:  mov    %eax,(%esp)
0x080486f0 <+119>:  call   0x8048500 <strcmp@plt>
0x080486f5 <+124>:  test   %eax,%eax
0x080486f7 <+126>:  je     0x80486fe <main+133>
0x080486f9 <+128>:  call   0x804860d <boom>
0x080486fe <+133>:  movl   $0x0,0x4(%esp)
0x08048706 <+141>:  lea     0x20(%esp),%eax
0x0804870a <+145>:  mov     %eax,(%esp)
0x0804870d <+148>:  call   0x8048480 <gettimeofday@plt>
0x08048712 <+153>:  mov     0x20(%esp),%edx
0x08048716 <+157>:  mov     0x18(%esp),%eax
0x0804871a <+161>:  sub     %eax,%edx
0x0804871c <+163>:  mov     %edx,%eax
0x0804871e <+165>:  cmp     $0x3c,%eax
0x08048721 <+168>:  jle     0x8048728 <main+175>
0x08048723 <+170>:  call   0x804860d <boom>
0x08048728 <+175>:  movl   $0x80488a1,(%esp)
0x0804872f <+182>:  call   0x8048460 <printf@plt>
0x08048734 <+187>:  lea     0x14(%esp),%eax
0x08048738 <+191>:  mov     %eax,0x4(%esp)
0x0804873c <+195>:  movl   $0x80488ba,(%esp)
0x08048743 <+202>:  call   0x80484f0 <__isoc99_scanf@plt>
0x08048748 <+207>:  mov     0x14(%esp),%edx
0x0804874c <+211>:  mov     0x804a04c,%eax
0x08048751 <+216>:  cmp     %eax,%edx
0x08048753 <+218>:  je     0x804875a <main+225>
0x08048755 <+220>:  call   0x804860d <boom>
0x0804875a <+225>:  movl   $0x0,0x4(%esp)
0x08048762 <+233>:  lea     0x18(%esp),%eax
0x08048766 <+237>:  mov     %eax,(%esp)
0x08048769 <+240>:  call   0x8048480 <gettimeofday@plt>
0x0804876e <+245>:  mov     0x18(%esp),%edx
0x08048772 <+249>:  mov     0x20(%esp),%eax
0x08048776 <+253>:  sub     %eax,%edx
0x08048778 <+255>:  mov     %edx,%eax
0x0804877a <+257>:  cmp     $0x3c,%eax
0x0804877d <+260>:  jle     0x8048784 <main+267>
0x0804877f <+262>:  call   0x804860d <boom>
0x08048784 <+267>:  call   0x8048643 <defused>
0x08048789 <+272>:  mov     0x8c(%esp),%ecx
0x08048790 <+279>:  xor     %gs:0x14,%ecx
0x08048797 <+286>:  je     0x804879e <main+293>
0x08048799 <+288>:  call   0x8048490 <__stack_chk_fail@plt>
0x0804879e <+293>:  leave
0x0804879f <+294>:  ret

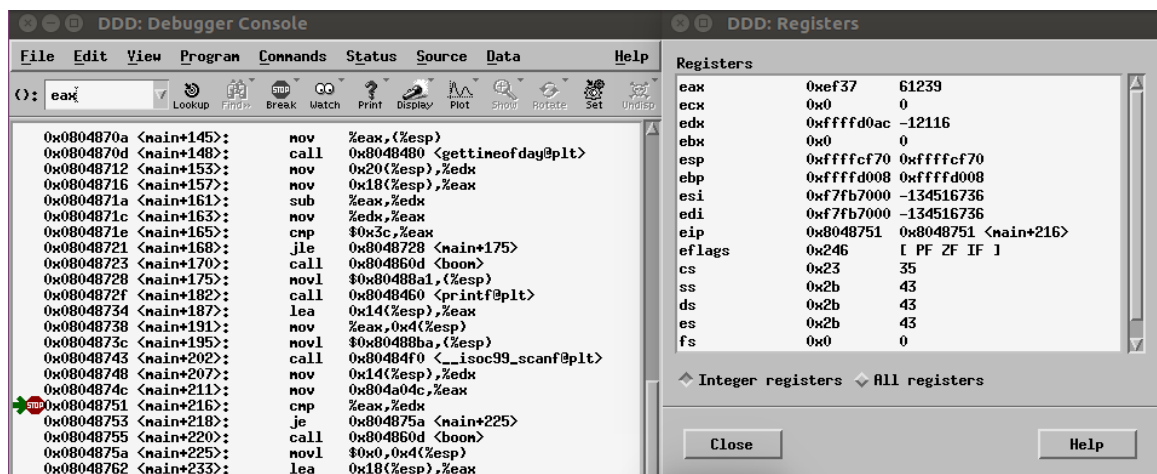
```

La primera cosa importante sucede en la línea 95 del main, donde se hace un strlen, en mi caso como prueba de contraseña he introducido hola pero la función ha devuelto en eax un 10 por lo que la contraseña ya está en memoria ya que sino nos hubiese devuelto 4.

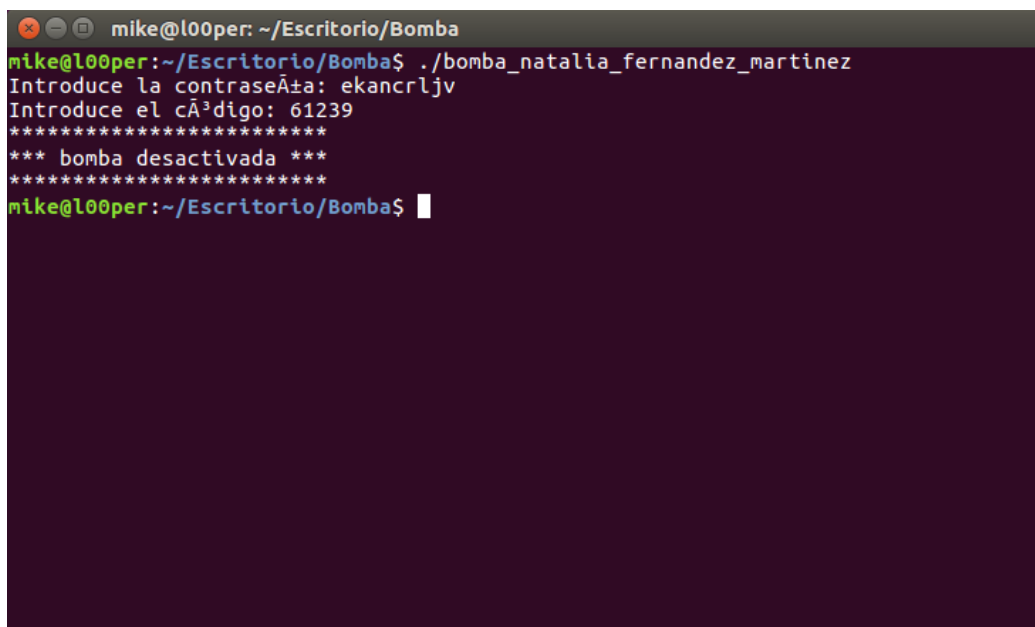
Tras esto se compara con strcmp que un compare entre strings por lo que vamos a buscarla sobre la dirección que marco en la imagen y que queda contenida en %eax debido al lea. Para ver lo que hay en la dirección Data -> Memory y en la dirección 0x804a040 la encontramos como se puede ver



Como podemos ver la contraseña estaba en memoria y es: ekancrljv. Para la contraseña numerica también lo tenemos fácil ya que tras pedirla (scanf línea 202) lo único que hace es aver la comparación y ver que hay la línea 216, en este caso contiene: 61239 que es el passcode:



Por último solo queda probarlo en una ejecución normal que se puede ver en la siguiente imagen:



```
mike@l00per: ~/Escritorio/Bomba
mike@l00per:~/Escritorio/Bomba$ ./bomba_natalia_fernandez_martinez
Introduce la contraseña: ekancrljv
Introduce el código: 61239
*****
*** bomba desactivada ***
*****
mike@l00per:~/Escritorio/Bomba$
```

3. Bomba de Raimundo Perez Rubio.

El código de este alumno posee dos funciones para cifrar el password y contraseña, el código del main es el siguiente:

```
0x080486de <+0>: lea 0x4(%esp),%ecx
0x080486e2 <+4>: and $0xffffffff0,%esp
0x080486e5 <+7>: pushl -0x4(%ecx)
0x080486e8 <+10>: push %ebp
0x080486e9 <+11>: mov %esp,%ebp
0x080486eb <+13>: push %ecx
0x080486ec <+14>: sub $0x84,%esp
0x080486f2 <+20>: mov %gs:0x14,%eax
0x080486f8 <+26>: mov %eax,-0xc(%ebp)
0x080486fb <+29>: xor %eax,%eax
0x080486fd <+31>: sub $0x8,%esp
0x08048700 <+34>: push $0x0
0x08048702 <+36>: lea -0x80(%ebp),%eax
0x08048705 <+39>: push %eax
0x08048706 <+40>: call 0x8048480 <gettimeofday@plt>
0x0804870b <+45>: add $0x10,%esp
0x0804870e <+48>: sub $0xc,%esp
0x08048711 <+51>: push $0x8048914
0x08048716 <+56>: call 0x8048460 <printf@plt>
0x0804871b <+61>: add $0x10,%esp
0x0804871e <+64>: mov 0x804a060,%eax
0x08048723 <+69>: sub $0x4,%esp
```

```

0x08048726 <+72>: push  %eax
0x08048727 <+73>: push  $0x64
0x08048729 <+75>: lea   -0x70(%ebp),%eax
0x0804872c <+78>: push  %eax
0x0804872d <+79>: call  0x8048470 <fgets@plt>
0x08048732 <+84>: add   $0x10,%esp
0x08048735 <+87>: sub   $0xc,%esp
0x08048738 <+90>: lea   -0x70(%ebp),%eax
0x0804873b <+93>: push  %eax
0x0804873c <+94>: call  0x804868b <cifrar_password>
0x08048741 <+99>: add   $0x10,%esp
0x08048744 <+102>: sub   $0xc,%esp
0x08048747 <+105>: push  $0x804a03c
0x0804874c <+110>: call  0x80484c0 <strlen@plt>
0x08048751 <+115>: add   $0x10,%esp
0x08048754 <+118>: sub   $0x4,%esp
0x08048757 <+121>: push  %eax
0x08048758 <+122>: push  $0x804a03c
0x0804875d <+127>: lea   -0x70(%ebp),%eax
0x08048760 <+130>: push  %eax
0x08048761 <+131>: call  0x80484f0 <strncmp@plt>
0x08048766 <+136>: add   $0x10,%esp
0x08048769 <+139>: test  %eax,%eax
0x0804876b <+141>: je     0x8048772 <main+148>
0x0804876d <+143>: call  0x804860b <boom>
0x08048772 <+148>: sub   $0x8,%esp
0x08048775 <+151>: push  $0x0
0x08048777 <+153>: lea   -0x78(%ebp),%eax
0x0804877a <+156>: push  %eax
0x0804877b <+157>: call  0x8048480 <gettimeofday@plt>
0x08048780 <+162>: add   $0x10,%esp
0x08048783 <+165>: mov   -0x78(%ebp),%edx
0x08048786 <+168>: mov   -0x80(%ebp),%eax
0x08048789 <+171>: sub   %eax,%edx
0x0804878b <+173>: mov   %edx,%eax
0x0804878d <+175>: cmp   $0x5,%eax
0x08048790 <+178>: jle    0x8048797 <main+185>
0x08048792 <+180>: call  0x804860b <boom>
0x08048797 <+185>: sub   $0xc,%esp
0x0804879a <+188>: push  $0x804892f
0x0804879f <+193>: call  0x8048460 <printf@plt>
0x080487a4 <+198>: add   $0x10,%esp
0x080487a7 <+201>: sub   $0x8,%esp
0x080487aa <+204>: lea   -0x84(%ebp),%eax
0x080487b0 <+210>: push  %eax
0x080487b1 <+211>: push  $0x8048946
0x080487b6 <+216>: call  0x80484e0 <__isoc99_scanf@plt>
0x080487bb <+221>: add   $0x10,%esp
0x080487be <+224>: mov   -0x84(%ebp),%eax
0x080487c4 <+230>: sub   $0xc,%esp
0x080487c7 <+233>: push  %eax
0x080487c8 <+234>: call  0x80486d1 <cifrar_passcode>
0x080487cd <+239>: add   $0x10,%esp

```

```

0x080487d0 <+242>: mov    %eax,-0x84(%ebp)
0x080487d6 <+248>: mov    -0x84(%ebp),%edx
0x080487dc <+254>: mov    0x804a04c,%eax
0x080487e1 <+259>: cmp    %eax,%edx
0x080487e3 <+261>: je     0x80487ea <main+268>
0x080487e5 <+263>: call   0x804860b <boom>
0x080487ea <+268>: sub    $0x8,%esp
0x080487ed <+271>: push   $0x0
0x080487ef <+273>: lea    -0x80(%ebp),%eax
0x080487f2 <+276>: push   %eax
0x080487f3 <+277>: call   0x8048480 <gettimeofday@plt>
0x080487f8 <+282>: add    $0x10,%esp
0x080487fb <+285>: mov    -0x80(%ebp),%edx
0x080487fe <+288>: mov    -0x78(%ebp),%eax
0x08048801 <+291>: sub    %eax,%edx
0x08048803 <+293>: mov    %edx,%eax
0x08048805 <+295>: cmp    $0x5,%eax
0x08048808 <+298>: jle    0x804880f <main+305>
0x0804880a <+300>: call   0x804860b <boom>
0x0804880f <+305>: call   0x804864b <defused>
0x08048814 <+310>: mov    $0x0,%eax
0x08048819 <+315>: mov    -0xc(%ebp),%ecx
0x0804881c <+318>: xor    %gs:0x14,%ecx
0x08048823 <+325>: je     0x804882a <main+332>
0x08048825 <+327>: call   0x8048490 <__stack_chk_fail@plt>
0x0804882a <+332>: mov    -0x4(%ebp),%ecx
0x0804882d <+335>: leave
0x0804882e <+336>: lea    -0x4(%ecx),%esp
0x08048831 <+339>: ret

```

Como podemos ver hay varios puntos donde existen saltos que evitan que se active la función boom por lo que vamos a hacer que estos saltos se vuelvan incondicionales ya que así “saltarán por encima de las instrucciones call <boom>”

Para relizar esto lo haremos con el gdb y para ello deberemos relizar los siguientes comandos, también se puede hacer esto mismo con un editor hexadecimal.

```

$ gdb
(gdb) set write on
(gdb) file bomba_RaimundoPerezRubio

```

Ahora para todos las direcciones de que pertenecen a las instrucciones de salto haremos un set para modificarlos por un jmp cuyo código es 0xeb, es decir:

```
(gdb) set *(char*)0x0804876b=0xeb
(gdb) set *(char*)0x08048790=0xeb
(gdb) set *(char*)0x080487e3=0xeb
(gdb) set *(char*)0x08048808=0xeb
```

Tras esto si hacemos un `disas main` mostrará el mismo código que antes pero todos los saltos serán `jmp`:

```
0x080486de <+0>:    lea    0x4(%esp),%ecx
0x080486e2 <+4>:    and    $0xffffffff0,%esp
0x080486e5 <+7>:    pushl  -0x4(%ecx)
0x080486e8 <+10>:   push   %ebp
0x080486e9 <+11>:   mov    %esp,%ebp
0x080486eb <+13>:   push   %ecx
0x080486ec <+14>:   sub    $0x84,%esp
0x080486f2 <+20>:   mov    %gs:0x14,%eax
0x080486f8 <+26>:   mov    %eax,-0xc(%ebp)
0x080486fb <+29>:   xor    %eax,%eax
0x080486fd <+31>:   sub    $0x8,%esp
0x08048700 <+34>:   push   $0x0
0x08048702 <+36>:   lea    -0x80(%ebp),%eax
0x08048705 <+39>:   push   %eax
0x08048706 <+40>:   call   0x8048480 <gettimeofday@plt>
0x0804870b <+45>:   add    $0x10,%esp
0x0804870e <+48>:   sub    $0xc,%esp
0x08048711 <+51>:   push   $0x8048914
0x08048716 <+56>:   call   0x8048460 <printf@plt>
0x0804871b <+61>:   add    $0x10,%esp
0x0804871e <+64>:   mov    0x804a060,%eax
0x08048723 <+69>:   sub    $0x4,%esp
0x08048726 <+72>:   push   %eax
0x08048727 <+73>:   push   $0x64
0x08048729 <+75>:   lea    -0x70(%ebp),%eax
0x0804872c <+78>:   push   %eax
0x0804872d <+79>:   call   0x8048470 <fgets@plt>
0x08048732 <+84>:   add    $0x10,%esp
0x08048735 <+87>:   sub    $0xc,%esp
0x08048738 <+90>:   lea    -0x70(%ebp),%eax
0x0804873b <+93>:   push   %eax
0x0804873c <+94>:   call   0x804868b <cifrar_password>
0x08048741 <+99>:   add    $0x10,%esp
0x08048744 <+102>:  sub    $0xc,%esp
0x08048747 <+105>:  push   $0x804a03c
0x0804874c <+110>:  call   0x80484c0 <strlen@plt>
0x08048751 <+115>:  add    $0x10,%esp
0x08048754 <+118>:  sub    $0x4,%esp
0x08048757 <+121>:  push   %eax
0x08048758 <+122>:  push   $0x804a03c
```

```

0x0804875d <+127>: lea -0x70(%ebp),%eax
0x08048760 <+130>: push %eax
0x08048761 <+131>: call 0x80484f0 <strncmp@plt>
0x08048766 <+136>: add $0x10,%esp
0x08048769 <+139>: test %eax,%eax
0x0804876b <+141>: jmp 0x8048772 <main+148>
0x0804876d <+143>: call 0x804860b <boom>
0x08048772 <+148>: sub $0x8,%esp
0x08048775 <+151>: push $0x0
0x08048777 <+153>: lea -0x78(%ebp),%eax
0x0804877a <+156>: push %eax
0x0804877b <+157>: call 0x8048480 <gettimeofday@plt>
0x08048780 <+162>: add $0x10,%esp
0x08048783 <+165>: mov -0x78(%ebp),%edx
0x08048786 <+168>: mov -0x80(%ebp),%eax
0x08048789 <+171>: sub %eax,%edx
0x0804878b <+173>: mov %edx,%eax
0x0804878d <+175>: cmp $0x5,%eax
0x08048790 <+178>: jmp 0x8048797 <main+185>
0x08048792 <+180>: call 0x804860b <boom>
0x08048797 <+185>: sub $0xc,%esp
0x0804879a <+188>: push $0x804892f
0x0804879f <+193>: call 0x8048460 <printf@plt>
0x080487a4 <+198>: add $0x10,%esp
0x080487a7 <+201>: sub $0x8,%esp
0x080487aa <+204>: lea -0x84(%ebp),%eax
0x080487b0 <+210>: push %eax
0x080487b1 <+211>: push $0x8048946
0x080487b6 <+216>: call 0x80484e0 <__isoc99_scanf@plt>
0x080487bb <+221>: add $0x10,%esp
0x080487be <+224>: mov -0x84(%ebp),%eax
0x080487c4 <+230>: sub $0xc,%esp
0x080487c7 <+233>: push %eax
0x080487c8 <+234>: call 0x80486d1 <cifrar_passcode>
0x080487cd <+239>: add $0x10,%esp
0x080487d0 <+242>: mov %eax,-0x84(%ebp)
0x080487d6 <+248>: mov -0x84(%ebp),%edx
0x080487dc <+254>: mov 0x804a04c,%eax
0x080487e1 <+259>: cmp %eax,%edx
0x080487e3 <+261>: jmp 0x80487ea <main+268>
0x080487e5 <+263>: call 0x804860b <boom>
0x080487ea <+268>: sub $0x8,%esp
0x080487ed <+271>: push $0x0
0x080487ef <+273>: lea -0x80(%ebp),%eax
0x080487f2 <+276>: push %eax
0x080487f3 <+277>: call 0x8048480 <gettimeofday@plt>
0x080487f8 <+282>: add $0x10,%esp
0x080487fb <+285>: mov -0x80(%ebp),%edx
0x080487fe <+288>: mov -0x78(%ebp),%eax
0x08048801 <+291>: sub %eax,%edx
0x08048803 <+293>: mov %edx,%eax
0x08048805 <+295>: cmp $0x5,%eax
0x08048808 <+298>: jmp 0x804880f <main+305>

```

```
0x0804880a <+300>: call 0x804860b <boom>
0x0804880f <+305>: call 0x804864b <defused>
0x08048814 <+310>: mov $0x0,%eax
0x08048819 <+315>: mov -0xc(%ebp),%ecx
0x0804881c <+318>: xor %gs:0x14,%ecx
0x08048823 <+325>: je 0x804882a <main+332>
0x08048825 <+327>: call 0x8048490 <__stack_chk_fail@plt>
0x0804882a <+332>: mov -0x4(%ebp),%ecx
0x0804882d <+335>: leave
0x0804882e <+336>: lea -0x4(%ecx),%esp
0x08048831 <+339>: ret
```

Dando igual que contraseña meter en las bombas, nunca se ejecutará la función estallido.