

Password	Código	Compilación
M30vc12+	6977	\$ gcc -m32 -g bomba.c -o bomba

Desactivar

Inicio

Vamos a comprobar cómo se puede desactivar la bomba digital que he programado sin ser necesario para ello conocer ninguna de las claves ni hacerlo en un tiempo determinado.

Lo primero será abrir la bomba con el depurador gráfico **ddd**, que se basa en el depurador de línea de comandos gdb y además incorpora un desensamblador. Para que nos aparezca el desensamblado del método main de nuestra bomba, activaremos la vista del código máquina en **View** → **Machine Code Window**, comenzamos la ejecución con “**run**” y vamos ejecutando el código paso a paso con “**nexti**” para no meternos en subrutinas por ahora.

Saltarse la comprobación de la primera clave

Lo primero que deberemos conseguir es saltarnos la primera protección, la comprobación de que la contraseña que introduzcamos es correcta.

Para ello continuamos con la ejecución con **nexti** hasta cuando nos pida que introduzcamos la contraseña.

Introducimos una cadena de caracteres cualquiera.

The screenshot shows the DDD interface with the assembly code of the main function. The code is as follows:

```

0x080486b5 <main+1>: mov    %esp,%ebp
0x080486b7 <main+3>: and    $0xffffffff,%esp
0x080486ba <main+6>: sub    $0x90,%esp
0x080486c0 <main+12>: mov    %gs:0x14,%eax
0x080486c6 <main+18>: mov    %eax,0x8c(%esp)
0x080486cd <main+25>: xor    %eax,%eax
0x080486cf <main+27>: movl   $0x0,0x4(%esp)
0x080486d7 <main+35>: lea    0x14(%esp),%eax
0x080486db <main+39>: mov    %eax,(%esp)
0x080486de <main+42>: call   0x8048470 <gettimeofday@plt>
0x080486e3 <main+47>: mov    $0x8048914,%eax
0x080486e8 <main+52>: mov    %eax,(%esp)
0x080486eb <main+55>: call   0x8048450 <printf@plt>
0x080486f0 <main+60>: mov    0x804a030,%eax
0x080486f5 <main+65>: mov    %eax,0x8(%esp)
0x080486f9 <main+69>: movl   $0x64,0x4(%esp)
0x08048701 <main+77>: lea    0x28(%esp),%eax
0x08048705 <main+81>: mov    %eax,(%esp)
0x08048708 <main+84>: call   0x8048460 <fgets@plt>
0x0804870d <main+89>: lea    0x28(%esp),%eax
0x08048711 <main+93>: mov    %eax,(%esp)
0x08048714 <main+96>: call   0x80485a4 <decodepas>
0x08048719 <main+101>: test   %eax,%eax
0x0804871b <main+103>: jne    0x8048722 <main+110>
0x0804871d <main+105>: call   0x8048648 <boom>

```

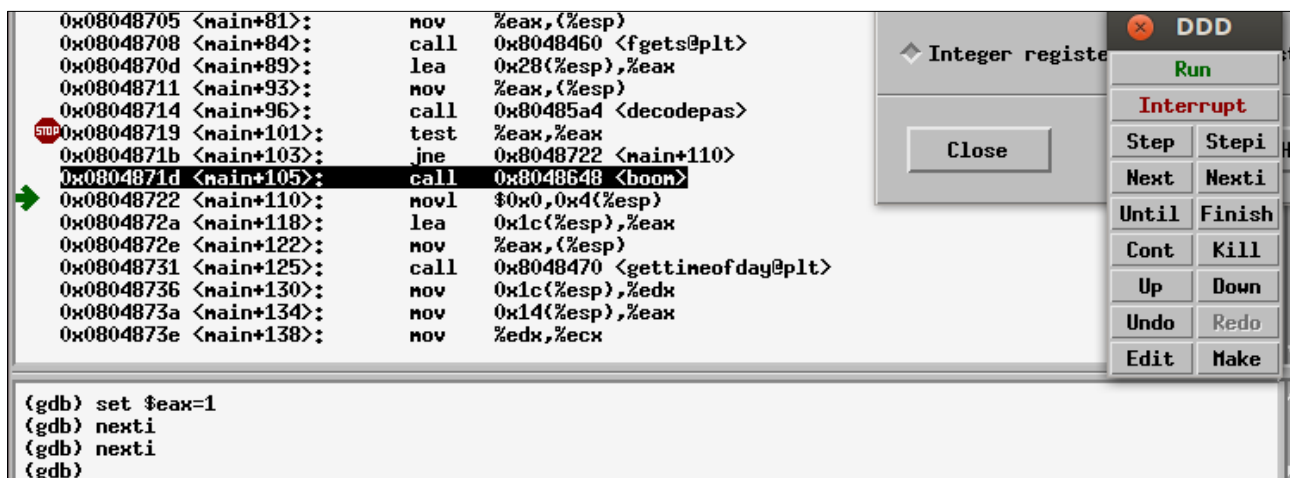
An execution window titled "DDD: Execution Window" is open in the foreground, displaying the prompt "Introduce la contraseña: jkls" with the input "jkls" entered.

Y continuamos con **nexti**.

Como se puede observar en la imagen colocamos un break después de la llamada a la función **decodepas** esta función es la encargada de comprobar si la clave es correcta (1=true ó 0=false).

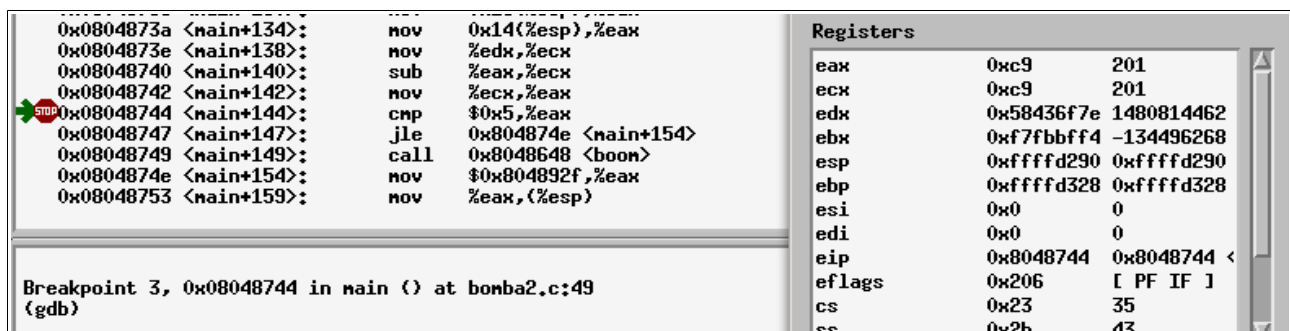
En este caso hemos ingresado una clave errónea por lo que el valor devuelto sera 0.

Para que se produzca el salto cambiamos el valor de eax (set \$eax=1). Con esto hemos conseguido que nos de por válida la clave introducida y así evitamos que continúe con la llamada a la función boom().



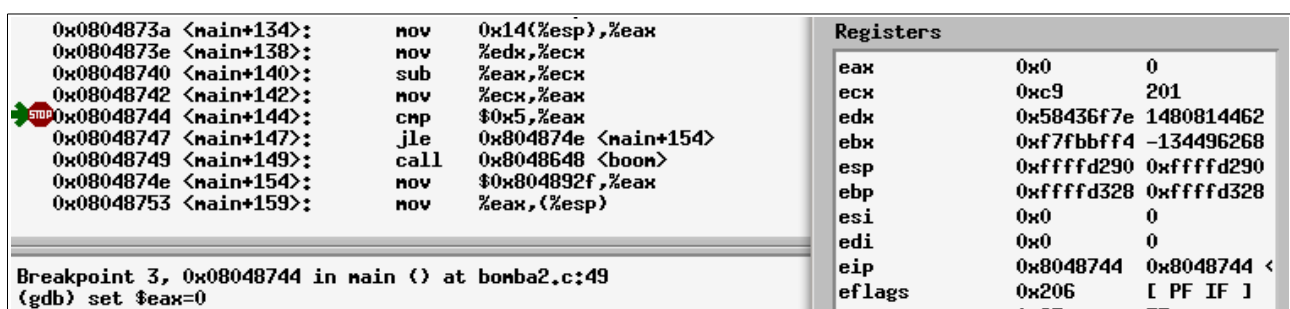
saltar la comprobación del primer temporizador

Continuamos ejecutando **nexti** en el depurador hasta que nos encontremos con otro salto condicional, este salto esta después de la llamada a la función **gettimeofday** pero antes del salto ponemos un breakpoint, en la instrucción **cmp \$0x5, %eax** esta comprobando si el valor de **eax** es menor que 5 .



Como se muestra en la imagen el valor de **eax** es mayor que 5 exactamente **eax = 201**.

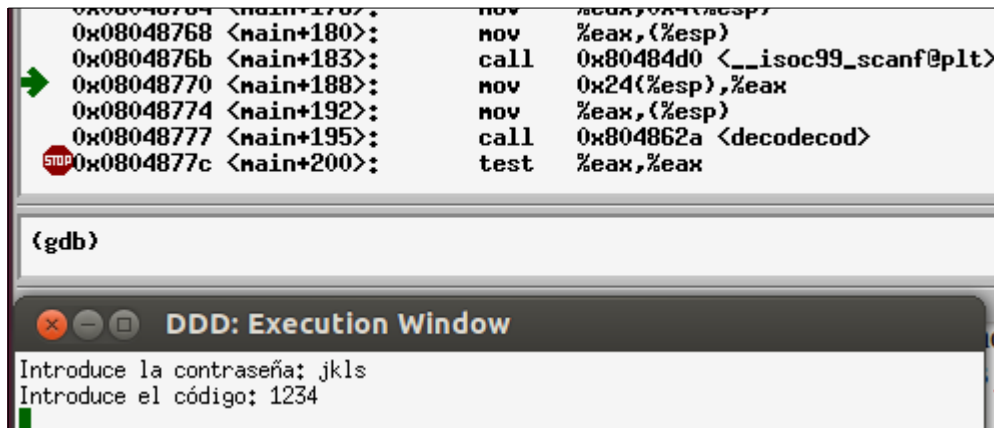
Para provocar el salto solo tendremos que cambiar el valor de **eax**, este valor tendrá que ser menor que 5.



Saltar la comprobación del código

Llegados a este punto solo nos queda por saltarnos la comprobación del código y la comprobación del segundo temporizador.

Continuamos con la ejecución con **nexti** hasta cuando nos pida que introduzcamos el código. Introducimos un valor cualquiera.



The screenshot shows the GDB interface. The assembly window displays the following instructions:

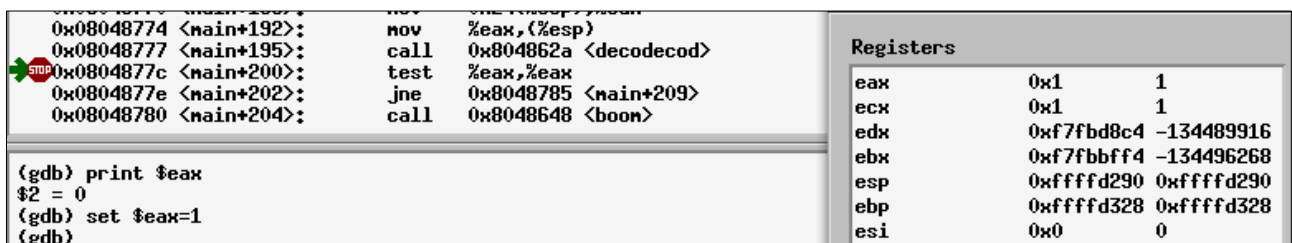
```
0x08048768 <nain+180>:  mov    %eax,%esp
0x0804876b <nain+183>:  call   0x80484d0 <__isoc99_scanf@plt>
0x08048770 <nain+188>:  mov    0x24(%esp),%eax
0x08048774 <nain+192>:  mov    %eax,%esp
0x08048777 <nain+195>:  call   0x804862a <decodecod>
0x0804877c <nain+200>:  test   %eax,%eax
```

The execution window (DDD: Execution Window) shows the following input:

```
Introduce la contraseña: jkls
Introduce el código: 1234
```

Continuamos con **nexti** hasta llegar a la instrucción que ejecuta la llamada a la función **decodecod**, esta función es la encargada de comprobar si el código es el correcto, el valor que devuelve la función es true o false (1 ó 0 respectivamente).

Como hemos introducido un valor erróneo la función devolverá false (0), por lo que colocamos un breakpoint en la ejecución de la instrucción test por si tenemos que volver a ejecutar y cambiamos el valor de eax.



The screenshot shows the GDB interface. The assembly window displays the following instructions:

```
0x08048774 <nain+192>:  mov    %eax,%esp
0x08048777 <nain+195>:  call   0x804862a <decodecod>
0x0804877c <nain+200>:  test   %eax,%eax
0x0804877e <nain+202>:  jne    0x8048785 <nain+209>
0x08048780 <nain+204>:  call   0x8048648 <boom>
```

The registers window shows the following values:

Register	Value
eax	0x1 1
ecx	0x1 1
edx	0xf7bd8c4 -134489916
ebx	0xf7bbff4 -134496268
esp	0xffffd290 0xffffd290
ebp	0xffffd328 0xffffd328
esi	0x0 0

The command window shows the following commands:

```
(gdb) print $eax
$2 = 0
(gdb) set $eax=1
(gdb)
```

Con ello hemos conseguido el salto evitando que continúe con la llamada a la función boom().

Saltar la comprobación del segundo temporizador

Lo siguiente que nos queda por hacer es evitar que la bomba explote por la temporización.

Para ello procedemos como el caso del salto de la comprobación del temporizador anterior.

Continuamos con ejecutando con **nexti**, como podemos observar en la siguiente imagen el valor que contiene eax es 487.

Este valor es mayor que 5, que es lo que se comprueba en la ejecución de la línea en la que se ha colocado el breakpoint.

Procedemos a cambiar el valor de eax por un valor menor que 5.

0x000407a3 <main+241>:	mov	%ecx,%eax		
0x080487a7 <main+243>:	cmp	\$0x5,%eax		
0x080487aa <main+246>:	jle	0x80487b1 <main+253>		


```

(gdb) print $eax
$3 = 487
(gdb) set $eax=0
(gdb) print $eax
$4 = 0

```

Registers		
eax	0x0	0
ecx	0x1e7	487
edx	0x58437165	1480814949
ebx	0xf7fbbff4	-134496268
esp	0xffffd290	0xffffd290
ebp	0xffffd328	0xffffd328

Con estos pasos hemos conseguido desactivar la bomba sin necesidad de conocer los códigos de desactivación.

0x000407a3 <main+241>:	mov	%ecx,%eax		
0x080487a5 <main+241>:	sub	%eax,%ecx		
0x080487a7 <main+243>:	mov	%ecx,%eax		
0x080487a7 <main+243>:	cmp	\$0x5,%eax		
0x080487aa <main+246>:	jle	0x80487b1 <main+253>		
0x080487ac <main+248>:	call	0x8048648 <boon>		
0x080487b1 <main+253>:	call	0x804867e <defused>		

End of assembler dump.

```

(gdb) nexti
[Inferior 1 (process 5652) exited normally]

```

DDD: Execution Window

Introduce la contraseña: jkls
 Introduce el código: 1234

 ... bomba desactivada ...

Obtención de la clave

Para obtener la clave procedemos como en el salto de comprobación de la clave, la diferencia esta en que esta ocasión necesitamos entrar en la subrutina de la función **decodepas**.

Procedemos con la ejecución y vamos avanzando con **nexti** hasta llegar a la instrucción de llamada a la función **decodepas** y continuamos con **stepi** esto nos permite entrar en la subrutina.

0x000407a3 <main+241>:	mov	%eax,%esp		
0x08048708 <main+84>:	call	0x8048460 <fgets@plt>		
0x0804870d <main+89>:	lea	0x28(%esp),%eax		
0x08048711 <main+93>:	mov	%eax,(%esp)		
0x08048714 <main+96>:	call	0x80485a4 <decodepas>		
0x08048719 <main+101>:	test	%eax,%eax		
0x0804871b <main+103>:	jne	0x8048722 <main+110>		
0x0804871d <main+105>:	call	0x8048648 <boon>		
0x08048722 <main+110>:	movl	\$0x0,0x4(%esp)		
0x0804872a <main+118>:	lea	0x1c(%esp),%eax		
0x0804872e <main+122>:	mov	%eax,(%esp)		
0x08048731 <main+125>:	call	0x8048470 <gettimeofday@plt>		


```

(gdb) print $eax
$1 = -11592
(gdb) x /1sb $eax
0xffffd2b8: "ghhjj\n"
(gdb)

```

DDD: Execution Window

Introduce la contraseña: ghhjj

Registers		
eax	0xffffd2b8	-11592
ecx	0xf7fd7006	-134385658
edx	0xf7fd8c4	-134489916
ebx	0xf7fbbff4	-134496268
esp	0xffffd290	0xffffd290
ebp	0xffffd328	0xffffd328
esi	0x0	0
edi	0x0	0
eip	0x8048714	0x8048714 <main+96>
eflags	0x282	[SF IF]
cs	0x23	35

Integer registers All registers

Close Help

Una vez dentro tenemos que tener en cuenta que a la función se le pasa un parámetro por lo que procedo a buscar dentro de la subrutina alguna variable que contenga el valor de la cadena que se utiliza para realizar la comparación.

```

Dump of assembler code for function decodepas:
=> 0x080485a4 <+0>:      push    %ebp
0x080485a5 <+1>:      mov     %esp,%ebp
0x080485a7 <+3>:      push    %edi
0x080485a8 <+4>:      sub     $0x44,%esp
0x080485ab <+7>:      mov     0x8(%ebp),%eax
0x080485ae <+10>:     mov     %eax,-0x2c(%ebp)
0x080485b1 <+13>:     mov     %gs:0x14,%eax
0x080485b7 <+19>:     mov     %eax,-0xc(%ebp)
0x080485ba <+22>:     xor     %eax,%eax
0x080485bc <+24>:     movl    $0x7630334d,-0x16(%ebp)
0x080485c3 <+31>:     movl    $0x2b323163,-0x12(%ebp)
0x080485ca <+38>:     movw    $0xa,-0xe(%ebp)
0x080485d0 <+44>:     lea     -0x16(%ebp),%eax
0x080485d3 <+47>:     movl    $0xffffffff,-0x30(%ebp)
0x080485da <+54>:     mov     %eax,%edx
0x080485dc <+56>:     mov     $0x0,%eax
0x080485e1 <+61>:     mov     -0x30(%ebp),%ecx

```

(gdb) x /1sb \$eax
0xffffd272: "M30vc12+\n"

Obtención del código

Es Similar que el paso anterior continuamos con nexti hasta que nos pida el código y una vez que estemos en la instrucción que llama a la función **decodecod** ejecutamos **stepi** para entrar en la subrutina

```

Dump of assembler code for function decodecod:
=> 0x0804862a <+0>:      push    %ebp
0x0804862b <+1>:      mov     %esp,%ebp
0x0804862d <+3>:      sub     $0x10,%esp
0x08048630 <+6>:      movl    $0x1b41,-0x8(%ebp)
0x08048637 <+13>:     mov     -0x8(%ebp),%eax
0x0804863a <+16>:     cmp     0x8(%ebp),%eax
0x0804863d <+19>:     sete    %al
0x08048640 <+22>:     movzbl  %al,%eax
0x08048643 <+25>:     mov     %eax,-0x4(%ebp)
0x08048646 <+28>:     leave   %eax
0x08048647 <+29>:     ret
End of assembler dump.

```

Registers		
eax	0x115c	4444
ecx	0x1	1
edx	0xf7fbd8c4	-134489916

Al entrar en la subrutina comprobamos que el valor que pasamos a la función es 4444, ahora empezaremos a buscar donde se encuentra almacenado el valor del código

Como podemos observar en la imagen siguiente encontramos una instrucción que compara dos valores y los almacena en eax.

Dump of assembler code for function decodecod:
=> 0x0804862a <+0>: push %ebp
0x0804862b <+1>: mov %esp,%ebp
0x0804862d <+3>: sub \$0x10,%esp
0x08048630 <+6>: movl \$0x1b41,-0x8(%ebp)
0x08048637 <+13>: mov -0x8(%ebp),%eax
➔ 0x0804863a <+16>: **cmp 0x8(%ebp),%eax**
0x0804863d <+19>: sete %al
0x08048640 <+22>: movzbl %al,%eax
0x08048643 <+25>: mov %eax,-0x4(%ebp)
0x08048646 <+28>: leave
0x08048647 <+29>: ret
End of assembler dump.

DDD: Registers

Registers

eax	0x1b41	6977
ecx	0x1	1
edx	0xf7fbd8c4	-134489916
ebx	0xf7fbbff4	-134496268
esp	0xffffd278	0xffffd278
ebp	0xffffd288	0xffffd288
esi	0x0	0

Procedo a averiguar el valor que tiene `eax` antes de que se ejecute la instrucción, el valor de `eax` es 6977 que es el código que desactiva nuestra bomba.