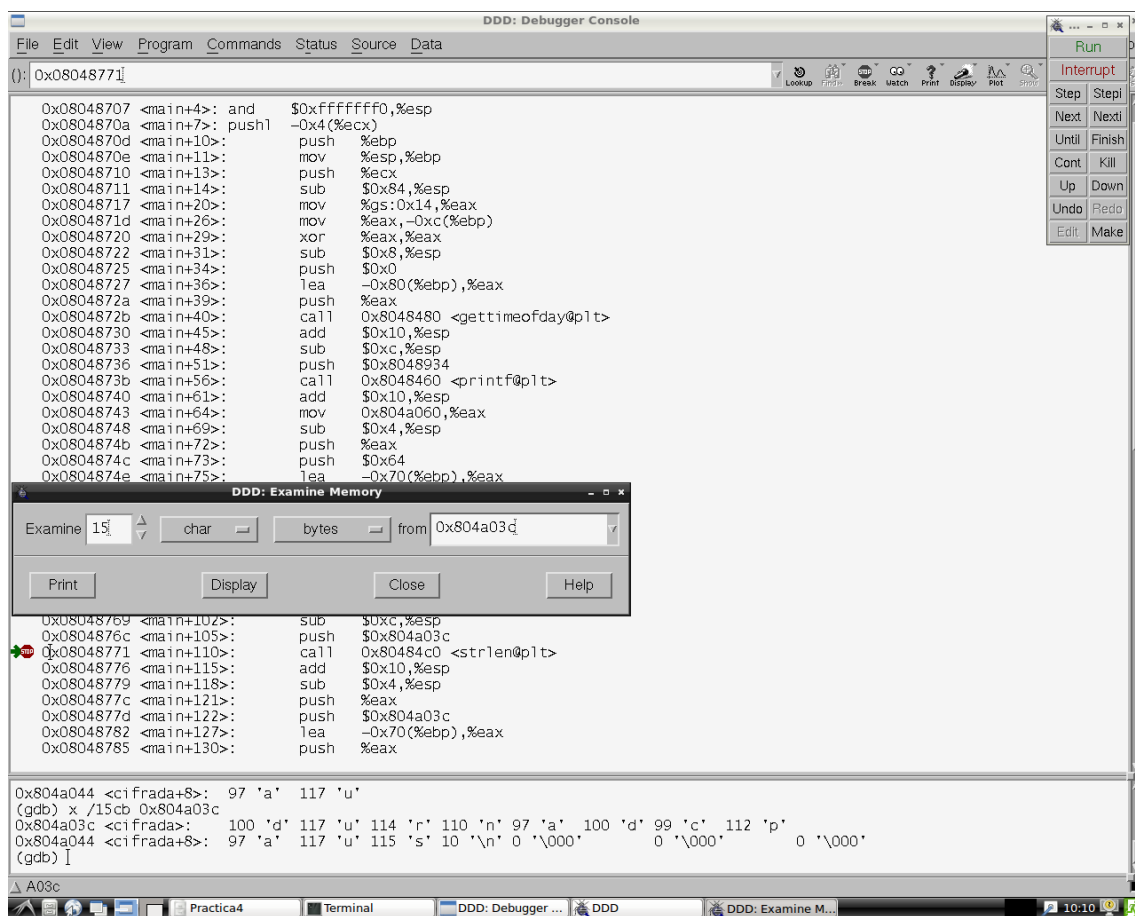


## Practica 4 Alejandro Poyatos López 2ºB(2)

Explicación de cómo resolver la bomba:

Para resolver mi propia bomba he seguido varios pasos:

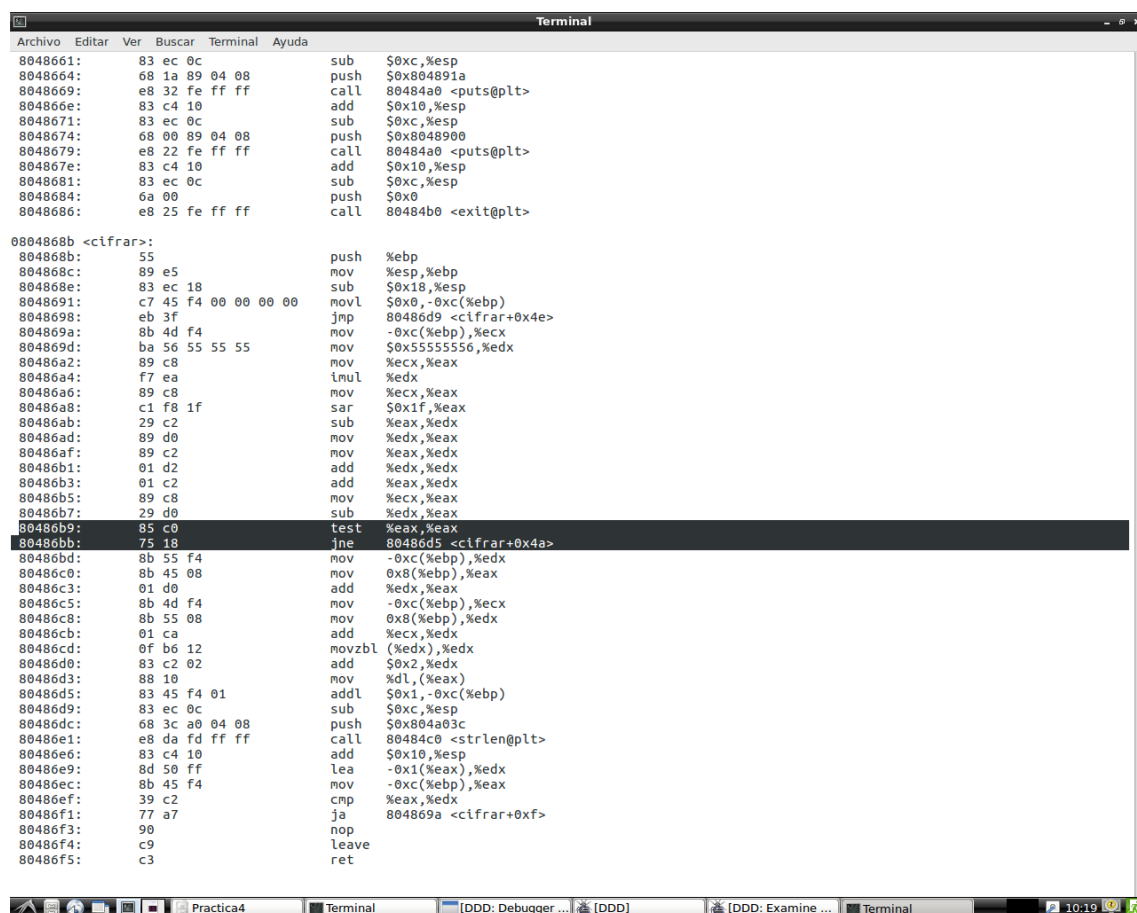
Primero he localizado y paralizado la ejecución con un breakpoint de DDD para obtener la contraseña cifrada, mediante la observación del argumento que le pasa el call a la llamada strlen:



Una vez llegados ahí, examinamos los datos de la memoria obteniéndose:

Durnadcpaus

Que no parece decir nada pero si observamos el código de cifrar con objdump veremos que hay un bucle y dentro una comprobación:



```

8048661: 83 ec 0c          sub     $0xc,%esp
8048664: 68 1a 89 04 08    push   $0x804891a
8048669: e8 32 fe ff ff    call   80484a0 <puts@plt>
804866e: 83 c4 10          add     $0x10,%esp
8048671: 83 ec 0c          sub     $0xc,%esp
8048674: 68 00 89 04 08    push   $0x8048900
8048679: e8 22 fe ff ff    call   80484a0 <puts@plt>
804867e: 83 c4 10          add     $0x10,%esp
8048681: 83 ec 0c          sub     $0xc,%esp
8048684: 6a 00            push   $0x0
8048686: e8 25 fe ff ff    call   80484b0 <exit@plt>

0804868b <cifrar>:
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 3f           jmp     80486d9 <cifrar+0x4e>
804869a: 8b 4d f4        mov     -0xc(%ebp),%ecx
804869d: ba 56 55 55 55  mov     $0x55555556,%edx
80486a2: 89 c8           mov     %ecx,%eax
80486a4: f7 ea          imul    %edx
80486a6: 89 c8           mov     %ecx,%eax
80486a8: c1 f8 1f       sar     $0x1f,%eax
80486ab: 29 c2          sub     %eax,%edx
80486ad: 89 d0           mov     %edx,%eax
80486af: 89 c2           mov     %eax,%edx
80486b1: 01 d2          add     %edx,%edx
80486b3: 01 c2          add     %eax,%edx
80486b5: 89 c8           mov     %ecx,%eax
80486b7: 29 d0          sub     %edx,%eax
80486b9: 85 c0          test    %eax,%eax
80486bb: 75 18          jne     80486d5 <cifrar+0x4a>
80486bd: 8b 55 f4        mov     -0xc(%ebp),%edx
80486c0: 8b 45 08        mov     0x8(%ebp),%eax
80486c3: 01 d0          add     %edx,%eax
80486c5: 8b 4d f4        mov     -0xc(%ebp),%ecx
80486c8: 8b 55 08        mov     0x8(%ebp),%edx
80486cb: 01 ca          add     %ecx,%edx
80486cd: 0f b6 12       movzbl (%edx),%edx
80486d0: 83 c2 02        add     $0x2,%edx
80486d3: 8b 10           mov     %dl,(%eax)
80486d5: 83 45 f4 01     addl    $0x1,-0xc(%ebp)
80486d9: 83 ec 0c        sub     $0xc,%esp
80486dc: 68 3c a0 04 08    push   $0x804a03c
80486e1: e8 da fd ff ff    call   80484c0 <strlen@plt>
80486e6: 83 c4 10        add     $0x10,%esp
80486e9: 8d 50 ff        lea     -0x1(%eax),%edx
80486ec: 8b 45 f4        mov     -0xc(%ebp),%eax
80486ef: 39 c2          cmp     %eax,%edx
80486f1: 77 a7          ja      804869a <cifrar+0xf>
80486f3: 90             nop
80486f4: c9             leave
80486f5: c3             ret

```

Que comprobará si el elemento que se esta leyendo divide a 3, en cuyo caso hará lo siguiente:

```

Terminal
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
8048661: 83 ec 0c      sub    $0xc,%esp
8048664: 68 1a 89 04 08 push  $0x804891a
8048669: e8 32 fe ff ff call   80484a0 <puts@plt>
804866e: 83 c4 10      add    $0x10,%esp
8048671: 83 ec 0c      sub    $0xc,%esp
8048674: 68 00 89 04 08 push  $0x8048900
8048679: e8 22 fe ff ff call   80484a0 <puts@plt>
804867e: 83 c4 10      add    $0x10,%esp
8048681: 83 ec 0c      sub    $0xc,%esp
8048684: 6a 00        push  $0x0
8048686: e8 25 fe ff ff call   80484b0 <exit@plt>

004868b <cfrrar>:
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 3f       jmp   80486d9 <cfrrar+0x4e>
804869a: 8b 4d f4    mov   -0xc(%ebp),%ecx
804869d: ba 56 55 55 55 mov   $0x55555555,%edx
80486a2: 89 c8       mov   %ecx,%eax
80486a4: f7 ea       imul  %edx
80486a6: 89 c8       mov   %ecx,%eax
80486a8: c1 f8 1f    sar   $0x1f,%eax
80486ab: 29 c2       sub   %eax,%edx
80486ad: 89 d0       mov   %edx,%eax
80486af: 89 c2       mov   %eax,%edx
80486b1: 01 d2       add   %edx,%edx
80486b3: 01 c2       add   %eax,%edx
80486b5: 89 c8       mov   %ecx,%eax
80486b7: 29 d0       sub   %edx,%eax
80486b9: 85 c0       test  %eax,%eax
80486bb: 75 18       jne   80486d5 <cfrrar+0x4a>
80486bd: 8b 55 f4    mov   -0xc(%ebp),%edx
80486c0: 8b 45 08    mov   0x8(%ebp),%eax
80486c3: 01 d0       add   %edx,%eax
80486c5: 8b 4d f4    mov   -0xc(%ebp),%ecx
80486c8: 8b 55 08    mov   0x8(%ebp),%edx
80486cb: 01 ca       add   %ecx,%edx
80486cd: 0f b6 12    movzbl (%edx),%edx
80486d0: 83 c2 02    add   $0x2,%edx
80486d3: 88 10       mov   %dl,(%eax)
80486d5: 83 45 f4 01 addl  $0x1,-0xc(%ebp)
80486d9: 83 ec 0c    sub   $0xc,%esp
80486dc: 68 3c a0 04 08 push  $0x804a03c
80486e1: e8 da fd ff ff call   80484c0 <strlen@plt>
80486e6: 83 c4 10    add   $0x10,%esp
80486e9: 8d 50 ff    lea   -0x1(%eax),%edx
80486ec: 8b 45 f4    mov   -0xc(%ebp),%eax
80486ef: 39 c2       cmp   %eax,%edx
80486f1: 77 a7       ja    804869a <cfrrar+0xf>
80486f3: 90          nop
80486f4: c9          leave
80486f5: c3          ret

```

Es decir que le suma 2 y va dando saltos, por lo que con el algoritmo que se basa en aumentar el valor del char estándar Unicode deducimos:

$$d - 2 = b$$

$$u = u$$

$$r = r$$

$$n - 2 = l$$

$$a = a$$

$$d = d$$

$$c - 2 = a$$

$$p = p$$

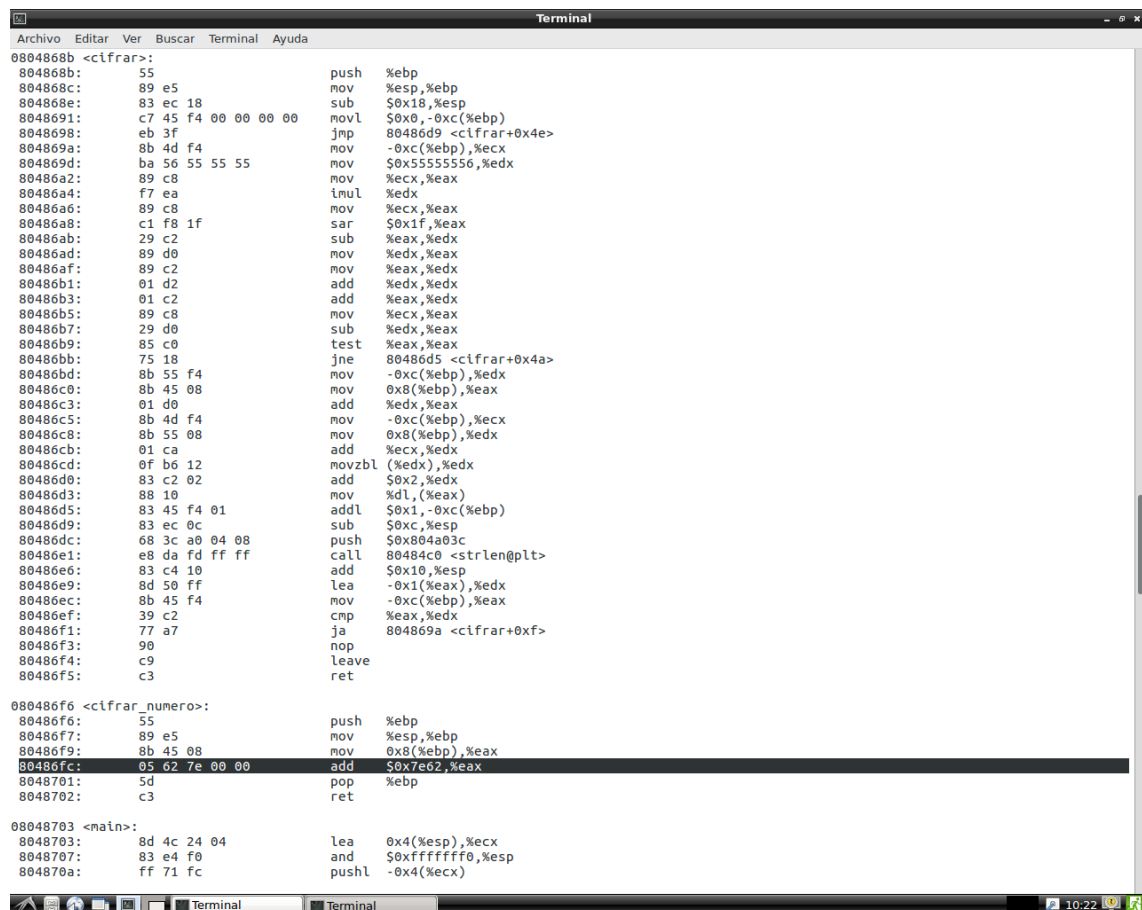
a = a

u -2= s

s = s

Entonces: durnadcpaus=burladapass

Para el código miraremos con objdump lo que hace el método cifrar\_numero, encargado de cifrar el código numérico:



```
004868b: <cifrar>:
004868b: 55                push    %ebp
004868c: 89 e5            mov     %esp,%ebp
004868e: 83 ec 18         sub     $0x18,%esp
0048691: c7 45 f4 00 00 00 00 movl    $0x0,-0xc(%ebp)
0048698: eb 3f           jmp     00486d9 <cifrar+0x4e>
004869a: 8b 4d f4         mov     -0xc(%ebp),%ecx
004869d: ba 56 55 55 55   mov     $0x55555556,%edx
00486a2: 89 c8            mov     %ecx,%eax
00486a4: f7 ea           imul    %edx
00486a6: 89 c8            mov     %ecx,%eax
00486a8: c1 f8 1f        sar     $0x1f,%eax
00486ab: 29 c2           sub     %eax,%edx
00486ad: 89 d0           mov     %edx,%eax
00486af: 89 c2           mov     %eax,%edx
00486b1: 01 d2           add     %edx,%edx
00486b3: 01 c2           add     %eax,%edx
00486b5: 89 c8           mov     %ecx,%eax
00486b7: 29 d0           sub     %edx,%eax
00486b9: 85 c0           test    %eax,%eax
00486bb: 75 18           jne     00486d5 <cifrar+0x4a>
00486bd: 8b 55 f4         mov     -0xc(%ebp),%edx
00486c0: 8b 45 08         mov     0x8(%ebp),%eax
00486c3: 01 d0           add     %edx,%eax
00486c5: 8b 4d f4         mov     -0xc(%ebp),%ecx
00486c8: 8b 55 08         mov     0x8(%ebp),%edx
00486cb: 01 ca           add     %ecx,%edx
00486cd: 0f b6 12        movzbl  (%edx),%edx
00486d0: 83 c2 02        add     $0x2,%edx
00486d3: 88 10           mov     %dl,(%eax)
00486d5: 83 45 f4 01     addl    $0x1,-0xc(%ebp)
00486d9: 83 ec 0c        sub     $0xc,%esp
00486dc: 68 3c a0 04 08   push    $0x804a03c
00486e1: e8 da fd ff ff   call    00484c0 <strlen@plt>
00486e6: 83 c4 10        add     $0x10,%esp
00486e9: 9d 50 ff        lea     -0x1(%eax),%edx
00486ec: 8b 45 f4         mov     -0xc(%ebp),%eax
00486ef: 39 c2           cmp     %eax,%edx
00486f1: 77 a7           ja      004869a <cifrar+0xf>
00486f3: 90              nop
00486f4: c9              leave
00486f5: c3              ret

00486f6: <cifrar_numero>:
00486f6: 55                push    %ebp
00486f7: 89 e5            mov     %esp,%ebp
00486f9: 8b 45 08         mov     0x8(%ebp),%eax
00486fc: 05 62 7e 00 00   add     $0x7e62,%eax
0048701: 5d              pop     %ebp
0048702: c3              ret

0048703: <main>:
0048703: 8d 4c 24 04      lea     0x4(%esp),%ecx
0048707: 83 e4 f0         and     $0xffffffff0,%esp
004870a: ff 71 fc        pushl   -0x4(%ecx)
```

Vemos que hace un suma del valor 7e62 (hex) que será igual a sumar 32354, por lo que ya solo queda que valor inicial tiene ese eax.

A esta altura no se muy bien el porque ddd no me permitió romper el código en el punto donde se le pasa el valor a cifrar número, por lo que opte por usar gdb:

The image shows two windows from a Linux desktop environment. The left window is a terminal displaying assembly code for a program. The right window is a gdb debugger interface showing a breakpoint at address 0x0048808 in the main function. The register Seax is shown with the value 67645.

```

alex@ubuntu: ~/Escritorio/Practica4
004871d: 89 45 f4      mov     %eax,%ecx(%ebp)
0048720: 31 c9        xor     %eax,%eax
0048722: 83 ec 08      sub     $0x8,%esp
0048725: 6a 00        push   $0x0
0048727: 8d 45 80      lea     -0x80(%ebp),%eax
004872a: 50          push   %eax
004872b: e8 50 fd ff ff call    0048480 <gettimeofday@plt>
0048730: 83 c4 10      add     $0x10,%esp
0048733: 83 ec 0c      sub     $0xc,%esp
0048736: 68 34 09 04 08 push   $0x0040934
004873b: e8 20 fd ff ff call    0048460 <printf@plt>
0048740: 83 c4 10      add     $0x10,%esp
0048743: a1 60 a0 04 08 mov     0x804a060,%eax
0048748: 83 ec 04      sub     $0x4,%esp
004874b: 50          push   %eax
004874c: 6a 64        push   $0x64
004874e: 8d 45 90      lea     -0x70(%ebp),%eax
0048751: 50          push   %eax
0048752: e8 19 fd ff ff call    0048470 <fgets@plt>
0048757: 83 c4 10      add     $0x10,%esp
004875a: 83 ec 0c      sub     $0xc,%esp
004875d: 8d 45 90      lea     -0x70(%ebp),%eax
0048760: 50          push   %eax
0048761: e8 25 fd ff ff call    00486b0 <memcmp@plt>
0048766: 83 c4 10      add     $0x10,%esp
0048769: 83 ec 0c      sub     $0xc,%esp
004876c: 68 3c a0 04 08 push   $0x0040a3c
0048771: e8 4a fd ff ff call    00484c0 <strlen@plt>
0048776: 83 c4 10      add     $0x10,%esp
0048779: 83 ec 04      sub     $0x4,%esp
004877c: 50          push   %eax
004877d: 68 3c a0 04 08 push   $0x0040a3c
0048782: 8d 45 90      lea     -0x70(%ebp),%eax
0048785: 50          push   %eax
0048786: e8 65 fd ff ff call    00484f0 <strncmp@plt>
004878b: 83 c4 10      add     $0x10,%esp
004878e: 85 c0        test    %eax,%eax
0048790: 74 05        je      0048797 <main+0x94>
0048792: e8 74 fd ff ff call    00486b0 <memcmp@plt>
0048797: 83 ec 08      sub     $0x8,%esp
004879a: 6a 00        push   $0x0
004879c: 8d 45 88      lea     -0x78(%ebp),%eax
004879f: 50          push   %eax
00487a0: e8 db fd ff ff call    0048480 <gettimeofday@plt>
00487a5: 83 c4 10      add     $0x10,%esp
00487a8: 8b 58        mov     -0x78(%ebp),%edx
00487ab: 8b 45 08      mov     -0x80(%ebp),%eax
00487ae: 29 c2        sub     %eax,%edx
00487b0: 89 d0        mov     %edx,%eax

```

```

alex@ubuntu: ~/Escritorio/Practica4
0x004882d <+298>: jle     0x0048834 <main+305>
0x004882f <+300>: call   0x00486b0 <boom>
0x0048834 <+305>: call   0x004864b <defused>
0x0048839 <+310>: mov     $0x0,%eax
0x004883e <+315>: mov     -0xc(%ebp),%ecx
0x0048841 <+318>: xor     %gs:0x14,%ecx
0x0048848 <+325>: je      0x004884f <main+332>
0x004884a <+327>: call    0x0048490 <__stack_chk_fail@plt>
0x004884f <+332>: mov     -0x4(%ebp),%ecx
0x0048852 <+335>: leave   %ecx
0x0048853 <+336>: lea     -0x4(%ecx),%esp
0x0048856 <+339>: ret
End of assembler dump.
(gdb) break *0x0048808
Punto de interrupción 2 at 0x0048808
(gdb) cont
Continuando.
Introduce la contraseña: burladapass
Introduce el código: 56565
Breakpoint 2, 0x0048808 in main ()
(gdb) display $eax
1: $eax = 67645
(gdb)

```

Ahí se ven las ordenes que use, y como hago un breakpoint justo antes del salto para mirar que valor decimal contiene el registro `eax = 67645`

Luego teniendo en cuenta el anterior ya deducimos el código:

$$67645 - 32354 = 35291$$

Finalmente 35291 es el código correcto

Finalmente, ya tenemos las dos contraseñas descifradas:

Password= burladapass

Passcode = 35291