

# TheReverser-Find\_password

<https://crackmes.one/crackme/5e9f4e8033c5d476117463a9>

Crackme writeup by @H0I3BI4ck

<https://twitter.com/H0I3BI4ck>

crackmes.one user [b1h0](https://crackmes.one/user/b1h0) <https://crackmes.one/user/b1h0>

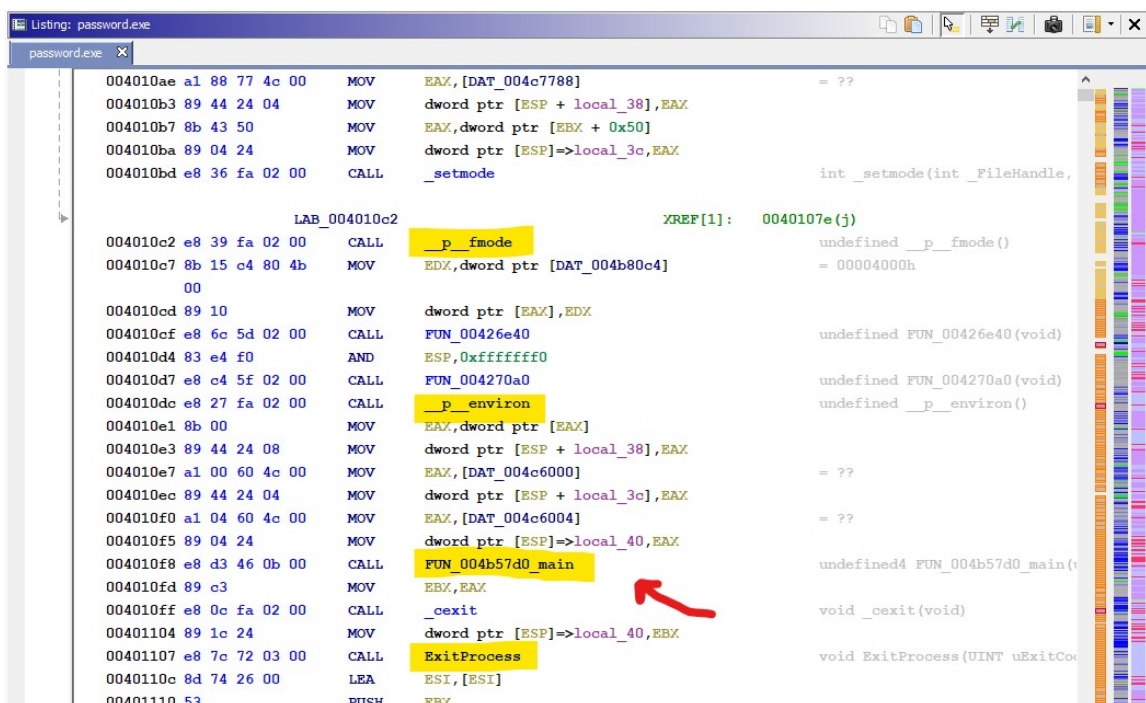
Date: 05/may/2020

You can download TheReverser-Find\_password from this [link](#).

## Ghidra's static analysis

We have a **Windows executable**, which has no reference to the **main()** function. We look for the entry point and we follow the different calls where we find a part of the code where it makes calls to the CRT mode and then to the environment variables, and then below there is a call to the exit to the system.

It is in this area, before the exit to the system, where the **main()** function of the program is located, so we rename it as the first starting point.



```

Listing: password.exe
password.exe X
004010ae a1 88 77 4c 00 MOV     EAX,[DAT_004c7788]
004010b3 89 44 24 04 MOV     dword ptr [ESP + local_38],EAX
004010b7 8b 43 50 MOV     EAX,dword ptr [EBX + 0x50]
004010ba 89 04 24 MOV     dword ptr [ESP+>local_3c,EAX]
004010bd e8 36 fa 02 00 CALL    _setmode                                int _setmode(int _FileHandle,

LAB_004010c2
004010c2 e8 39 fa 02 00 CALL    __p_fmode                                XREF[1]: 0040107e(j) undefined __p_fmode()
004010c7 8b 15 c4 80 4b MOV     EDX,dword ptr [DAT_004b80c4] = 00004000h
00
004010cd 89 10 MOV     dword ptr [EAX],EDX
004010cf e8 6c 5d 02 00 CALL    FUN_00426e40                                undefined FUN_00426e40(void)
004010d4 83 e4 f0 AND     ESP,0xffffffff0
004010d7 e8 c4 5f 02 00 CALL    FUN_004270a0                                undefined FUN_004270a0(void)
004010dc e8 27 fa 02 00 CALL    __p_environ                                undefined __p_environ()
004010e1 8b 00 MOV     EAX,dword ptr [EAX]
004010e3 89 44 24 08 MOV     dword ptr [ESP + local_38],EAX
004010e7 a1 00 60 4c 00 MOV     EAX,[DAT_004c6000] = ??
004010ec 89 44 24 04 MOV     dword ptr [ESP + local_3c],EAX
004010f0 a1 04 60 4c 00 MOV     EAX,[DAT_004c6004] = ??
004010f5 89 04 24 MOV     dword ptr [ESP]>local_40,EAX
004010f8 e8 d3 46 0b 00 CALL    FUN_004b57d0_main                                undefined4 FUN_004b57d0_main(v
004010fd 89 c3 MOV     EBX,EAX
004010ff e8 0c fa 02 00 CALL    _cexit                                void _cexit(void)
00401104 89 1c 24 MOV     dword ptr [ESP]>local_40,EBX
00401107 e8 7c 72 03 00 CALL    ExitProcess                                void ExitProcess(UINT uExitCo
0040110c 8d 74 26 00 LEA     ESI,[ESI]
00401110 53 PUSH    EBX
  
```

Entering already inside the **main()** we can quickly identify, and helping ourselves with the decompiler, the area where it shows the messages and asks us for the password. We already have specific points to look at and some variables to rename.

The screenshot shows the debugger interface with the assembly view on the left and the decompiled code on the right. A red arrow points to a loop in the decompiled code that handles password input and validation.

```

Listing: password.exe
FUN_004b57d0_main
004b57d0 8d 4c 24 04 LEA ECX, param_1, [ESP + 0x4]
004b57d4 83 e4 f0 AND ESP, 0xffffffff
004b57d7 ff 71 fe PUSH dword ptr [ECX + local_res0]
004b57da 55 PUSH EBP
004b57db 89 e5 MOV ESP, ESP
004b57dd 51 PUSH ECX
004b57de 8d 45 f8 LEA EAX, local_10, [ESP + -0x8]
004b57e1 81 ec 84 00 00 SUB ESP, 0x84
004b57e7 c7 45 ac d0 17 MOV dword ptr [ESP + local_5c], LAB_004017d0
004b57ee c7 45 b0 e0 5d MOV dword ptr [ESP + local_58], DAT_004b5de0
004b57f5 89 45 b4 MOV dword ptr [ESP + local_54], EAX
004b57f8 8d 45 94 LEA EAX, local_74, [ESP + -0x6c]
004b57fb 89 65 bc MOV dword ptr [ESP + local_4c], ESP
004b57fe c7 45 b8 65 59 MOV dword ptr [ESP + local_50], LAB_004b5965
004b5805 89 04 24 MOV dword ptr [ESP], local_90, EAX
004b5808 e8 03 21 f7 ff CALL FUN_00427910
004b580d e8 18 f7 ff CALL FUN_004270a0
004b5812 8d 45 d0 LEA EAX, local_38, [ESP + -0x30]
004b5815 8d 4d c8 LEA ECX, local_40, [ESP + -0x38]
004b5818 89 45 c8 MOV dword ptr [ESP + local_40], EAX
004b581b c7 44 24 04 54 MOV dword ptr [ESP + local_8c], DAT_004b9054
004b5823 c7 04 24 4e 90 MOV dword ptr [ESP], local_90, DAT_004b904e
004b582a c7 45 98 ff ff MOV dword ptr [ESP + local_70], 0xffffffff
004b5831 e8 2a bb f4 ff CALL FUN_00401360
004b5836 8d 45 e8 LEA EAX, local_20, [ESP + -0x18]
004b5839 83 ec 08 SUB ESP, 0x8

Decompile: FUN_004b57d0_main - (password.exe)
19 undefined local_10 [4];
20 undefined local_5c;
21
22 local_c = param_1;
23 local_54 = local_10;
24 local_4c = &local_0xffffffff;
25 local_5c = &LAB_004017d0;
26 local_58 = &DAT_004b5de0;
27 local_50 = &LAB_004b5965;
28 FUN_00427910(local_74);
29 FUN_004270a0();
30 local_4c = local_38;
31 local_70 = 0xffffffff;
32 FUN_00401360(&local_40, &DAT_004b904e, (int)&DAT_004b9054);
33 local_28 = local_20;
34 local_70 = 2;
35 FUN_00401360(&local_28, &DAT_004b9055, (int)&DAT_004b905f);
36 local_70 = 2;
37 FUN_004b3dc0((int *)&DAT_004c6860, "inserisci la password per accedere al programma ");
38 FUN_00403800((int *)&DAT_004c6900, (int *)&local_28);
39 do {
40     local_70 = 2;
41     FUN_004b0060((int *)&DAT_004c6860, "password errata.\n", &lab1);
42     FUN_004b0060((int *)&DAT_004c6860, "inserisci la password per accedere al programma ", &lab30);
43     FUN_00403800((int *)&DAT_004c6900, (int *)&local_28);
44     while (local_24 != local_30) {
45         iVar1 = memcmp(local_28, local_40, local_24);
46     } while (iVar1 != 0);
47     FUN_004b0060((int *)&DAT_004c6860, "benvenuto", &lab9);
48     FUN_004b0070((int *)&DAT_004c6860);
49     if (local_28 != local_30) {
50         free(local_28);
51     }
52     if (local_40 != local_30) {
53         free(local_40);
54     }
55     FUN_00427a70(&local_74);
56     return 0;
57 }
  
```

If we focus on the decompiled code we see that there is a function that repeats 2 times. One before starting a loop and one inside a loop. If we test the program we can verify that this is the data entry function where the password is requested.

The funny thing is that after asking for the password the first time it does not do any checking at all. It goes directly into the loop, so only the password we enter second is valid. The first time, even if it's a good one, it will always show us an error message and will ask us for the password again.

The screenshot shows the debugger interface with the assembly view on the left and the decompiled code on the right. A red arrow points to a loop in the decompiled code that handles password input and validation.

```

004b5885 89 44 24 04 MOV dword ptr [ESP + local_8c], EAX
004b5889 e8 72 df f4 ff CALL FUN_00403800_input_string
004b588e 66 90 NOP

LAB_004b5890
004b5890 c7 44 24 08 11 00 00 MOV dword ptr [ESP + local_88], 0x11
004b5898 c7 44 24 04 91 90 4b 00 MOV dword ptr [ESP], local_8c, &password_errata_004b5890
004b58a0 c7 04 24 60 68 4c 00 MOV dword ptr [ESP], local_90, DAT_004c6860
004b58a7 c7 45 98 02 00 00 00 MOV dword ptr [ESP + local_70], 0x2
004b58ae e8 ad a7 ff ff CALL FUN_004b0060
004b58b3 c7 44 24 08 30 00 00 00 MOV dword ptr [ESP + local_88], 0x30
004b58b8 c7 44 24 04 60 90 4b 00 MOV dword ptr [ESP], local_8c, &inserisci_la_password
004b58c3 c7 04 24 60 68 4c 00 MOV dword ptr [ESP], local_90, DAT_004c6860
004b58ca e8 91 a7 ff ff CALL FUN_004b0060
004b58cf 8d 45 a0 LEA EAX, user_input_pass, [ESP + -0x20]
004b58d2 c7 04 24 00 69 4c 00 MOV dword ptr [ESP], local_90, DAT_004c6900
004b58d9 89 44 24 04 MOV dword ptr [ESP + local_8c], EAX
004b58dd e8 1e df f4 ff CALL FUN_00403800_input_string
004b58e2 8b 45 e4 MOV EAX, dword ptr [ESP + local_24]
004b58e5 3b 45 0c CMP EAX, dword ptr [ESP + local_3c]
004b58e8 75 a6 JNZ LAB_004b5890
004b58ea 89 44 24 08 MOV dword ptr [ESP + local_88], EAX
004b58ee 8b 45 c8 MOV EAX, dword ptr [ESP + local_40]

31 local_70 = 0xffffffff;
32 FUN_00401360(&local_40, &DAT_004b904e, (int)&DAT_004b9054);
33 user_input_pass = local_20;
34 local_70 = 2;
35 FUN_00401360(user_input_pass, &DAT_004b9055, (int)&DAT_004b905f);
36 local_70 = 2;
37 FUN_004b3dc0((int *)&DAT_004c6860, "inserisci la password per accedere al programma ");
38 FUN_00403800_input_string((int *)&DAT_004c6900, (int *)&user_input_pass);
39 do {
40     local_70 = 2;
41     FUN_004b0060((int *)&DAT_004c6860, "password errata.\n", &lab1);
42     FUN_004b0060((int *)&DAT_004c6860, "inserisci la password per accedere al programma ", &lab30);
43     FUN_00403800_input_string((int *)&DAT_004c6900, (int *)&user_input_pass);
44     while (local_24 != local_30) {
45         iVar1 = memcmp(user_input_pass, local_40, local_24);
46     } while (iVar1 != 0);
47     FUN_004b0060((int *)&DAT_004c6860, "benvenuto", &lab9);
48     FUN_004b0070((int *)&DAT_004c6860);
49     if (user_input_pass != local_20) {
50         free(user_input_pass);
51     }
52     if (local_40 != local_30) {
53         free(local_40);
54     }
55     FUN_00427a70(&local_74);
56     return 0;
57 }
  
```

We see that the check is done with 2 nested loops. The first one verifies that the length of the string is the expected one, therefore we can already identify which variables refer to the length of the password. Then the **memcmp()** function is called which compares two strings in memory and returns an integer indicating if one string is greater than another or they are the same. If the returned result is 0, it means that the strings are the same.

So, let's change a few more variable names.

```

004b5812 8d 45 d0      LEA     EAX=>local_38, [EBP + -0x30]
004b5815 8d 4d c8      LEA     ECX=>good_password, [EBP + -0x38]
004b5818 89 45 c8      MOV     dword ptr [EBP + good_password], EAX
004b581b c7 44 24 04 54 90 4b 00  MOV     dword ptr [ESP + local_8c], DAT_004b9054
004b5823 c7 04 24 4e 90 4b 00  MOV     dword ptr [ESP=>local_90, DAT_004b904e
004b582a c7 45 98 ff ff ff ff  MOV     dword ptr [EBP + local_70], 0xffffffff
004b5831 e8 2a bb f4 ff      CALL    FUN_00401360
004b5836 8d 45 e8      LEA     EAX=>local_20, [EBP + -0x18]
004b5839 83 ec 08      SUB     ESP, 0x8
004b583c 8d 4d e0      LEA     ECX=>user_input_pass, [EBP + -0x20]
004b583f 89 45 e0      MOV     dword ptr [EBP + user_input_pass], EAX
004b5842 c7 44 24 04 5f 90 4b 00  MOV     dword ptr [ESP + local_8c], DAT_004b905f
004b584a c7 04 24 55 90 4b 00  MOV     dword ptr [ESP=>local_90, DAT_004b9055
004b5851 c7 45 98 01 00 00 00  MOV     dword ptr [EBP + local_70], 0x1
004b5858 e8 03 bb f4 ff      CALL    FUN_00401360
004b585d 83 ec 08      SUB     ESP, 0x8
004b5860 c7 44 24 04 60 90 4b 00  MOV     dword ptr [ESP + local_8c], s_inserisci_la_password
004b5868 c7 04 24 60 68 4c 00  MOV     dword ptr [ESP=>local_90, DAT_004c6860
004b586f c7 45 98 02 00 00 00  MOV     dword ptr [EBP + local_70], 0x2
004b5876 e8 45 d5 ff ff      CALL    FUN_004b2dc0
004b587b 8d 45 e0      LEA     EAX=>user_input_pass, [EBP + -0x20]
004b587e c7 04 24 00 69 4c 00  MOV     dword ptr [ESP=>local_90, DAT_004c6900
004b5885 89 44 24 04      MOV     dword ptr [ESP + local_8c], EAX
004b5889 e8 72 df f4 ff      CALL    FUN_00403800_input_string
004b588e 66 90      NOP

```

```

27 local_50 = &LAB_004b5945;
28 FUN_00427910(&local_74);
29 FUN_004270a0();
30 good_password = local_38;
31 local_70 = 0xffffffff;
32 FUN_00401360(&good_password, &DAT_004b904e, (int)&DAT_004b9054);
33 user_input_pass = local_20;
34 local_70 = 1;
35 FUN_00401360(user_input_pass, &DAT_004b9055, (int)&DAT_004b905f);
36 local_70 = 2;
37 FUN_004b2dc0((int *)&DAT_004c6860, "inserisci la password per accedere al pro
38 FUN_00403800_input_string((int *)&DAT_004c6900, (int *)&user_input_pass);
39 do {
40     do {
41         local_70 = 2;
42         FUN_004b0060((int *)&DAT_004c6860, "password errata.\n", 0x11);
43         FUN_004b0060((int *)&DAT_004c6860, "inserisci la password per accedere al
44         FUN_00403800_input_string((int *)&DAT_004c6900, (int *)&user_input_pass);
45     } while (string_len != local_3c);
46     pass_no_ok = memcmp(user_input_pass, good_password, string_len);
47 } while (pass_no_ok != 0);
48 FUN_004b0060((int *)&DAT_004c6860, "benvenuto", 9);
49 FUN_004b0d70((int *)&DAT_004c6860);
50 if (user_input_pass != local_20) {
51     free(user_input_pass);
52 }
53 if (good_password != local_38) {
54     free(good_password);
55 }

```

We can now see a variable that we will call **"good\_password"** in which it is assumed that we have the password that is the correct one. A little further up, before entering the loop, a function is called with reference to this variable. Let's see what it does.

```

Listing: password.exe - (5 addresses selected)
password.exe
0040139d 8d 44 24 1c      LEA     EAX=>local_10, [ESP + 0x1c]
004013a1 c7 44 24 04 00 00 00 00  MOV     dword ptr [ESP + local_28], 0x0
004013a9 89 f1      MOV     this, ESI
004013ab 89 04 24      MOV     dword ptr [ESP=>local_2c, EAX
004013ae e8 5d 33 0a 00  CALL    FUN_004a4710
004013b3 52      PUSH    EDX
004013b4 52      PUSH    EDX
004013b5 8b 54 24 1c      MOV     EDI, dword ptr [ESP + local_10]
004013b9 89 06      MOV     dword ptr [ESI], EAX
004013bb 89 56 08      MOV     dword ptr [ESI + 0x8], EDI
004013be 89 5c 24 08      MOV     dword ptr [ESP + local_24], EDI
004013c2 89 7c 24 04      MOV     dword ptr [ESP + local_28], EDI
004013c6 89 04 24      MOV     dword ptr [ESP=>local_2c, EAX
004013c9 e8 52 f7 02 00  CALL    memcpy
004013ce 8b 44 24 1c      MOV     EAX, dword ptr [ESP + local_10]
004013d2 8b 16      MOV     EDI, dword ptr [ESI]
004013d4 89 46 04      MOV     dword ptr [ESI + 0x4], EAX
004013d7 c6 04 02 00      MOV     byte ptr [EDI + EAX*0x1], 0x0
004013db 83 04 20      ADD     ESP, 0x20
004013de 5b      POP     EBX
004013df 5e      POP     ESI

```

```

Decompile: FUN_00401360_memcpy - (password.exe)
1 void __thiscall FUN_00401360_memcpy(void *this, undefined *mem_addr, int size)
2 {
3     undefined *Det;
4     uint _Size;
5     uint local_10;
6     if (mem_addr == (undefined *)0x0) && (size != 0) {
7         /* WARNING: Subroutine does not return */
8         FUN_00402a00("basic_string::M_construct null not valid");
9     }
10     _Size = size - (int)mem_addr;
11     local_10 = _Size;
12     if (_Size < 0x10) {
13         Det = (undefined *)this;
14         if (_Size == 1) {
15             *Det = *mem_addr;
16             goto LAB_004013ce;
17         }
18     } else {
19         _Det = (undefined *)FUN_004a4710(&local_10, 0);
20         *Det = _Det;
21         *(uint *)((int)this + 8) = local_10;
22     }
23     memcpy(_Det, mem_addr, _Size);
24     LAB_004013ce:
25     *(uint *)((int)this + 4) = local_10;
26     *(undefined *)((int *)this + local_10) = 0;
27     return;
28 }

```

This function copies the contents of one memory area over another. A copy of literal strings. So let's see what is in that memory area pointed to by **&DAT\_004b904e**.

```

a_basic_string::M_construct_null_n_004b... XREF[1]: FUN_00401360_memcpy:00401378
004b9024 62 61 73 69 63 5f 73 ... ds "basic_string::M_construct null not valid"
004b904e 64      DAT_004b904e      ??      64h      d      XREF[1]: FUN_004b57d0_main:004b5823 (*)
004b904f 6a      ??      6Ah      j
004b9050 65      ??      65h      e
004b9051 6a      ??      6Ah      j
004b9052 69      ??      69h      i
004b9053 65      ??      65h      e
004b9054 00      DAT_004b9054      ??      00h
004b9055 00      DAT_004b9055      ??      00h
004b9056 67      ??      67h      g
004b9057 67      ??      67h      g
004b9058 6b      ??      6Bh      k
004b9059 66      ??      66h      f
004b905a 67      ??      67h      g
004b905b 6a      ??      6Ah      j
004b905c 66      ??      66h      f
004b905d 72      ??      72h      z
004b905e 67      ??      67h      g

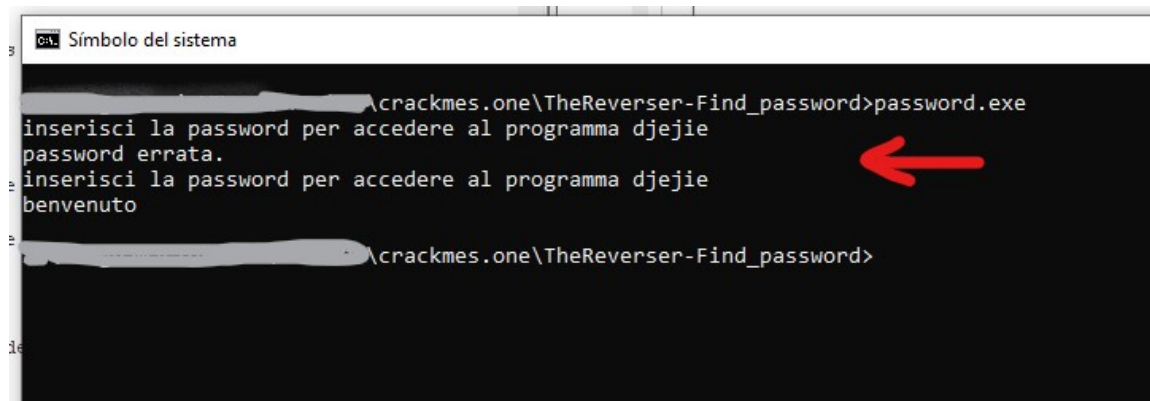
```

```

30 good_password = local_38;
31 local_70 = 0xffffffff;
32 FUN_00401360_memcpy(good_password, &DAT_004b904e, (int)&DAT_004b9054);
33 user_input_pass = local_20;
34 local_70 = 1;
35 FUN_00401360_memcpy(user_input_pass, &DAT_004b9055, (int)&DAT_004b905f);
36 local_70 = 2;
37 FUN_004b2dc0((int *)&DAT_004c6860, "inserisci la password per accedere al progr
38 FUN_00403800_input_string((int *)&DAT_004c6900, (int *)&user_input_pass);
39 do {
40     do {
41         local_70 = 2;
42         FUN_004b0060((int *)&DAT_004c6860, "password errata.\n", 0x11);
43         FUN_004b0060((int *)&DAT_004c6860, "inserisci la password per accedere al p
44         FUN_00403800_input_string((int *)&DAT_004c6900, (int *)&user_input_pass);
45     } while (string_len != local_3c);
46     pass_no_ok = memcmp(user_input_pass, good_password, string_len);
47 } while (pass_no_ok != 0);
48 FUN_004b0060((int *)&DAT_004c6860, "benvenuto", 9);
49 FUN_004b0d70((int *)&DAT_004c6860);
50 if (user_input_pass != local_20) {
51     free(user_input_pass);
52 }
53 if (good_password != local_38) {
54     free(good_password);
55 }
56 FUN_00427a70(&local_74);
57 return 0;
58 }

```

Here we can see that the text there is **"djejie"**. Can this be the password? Surely. Let's check it ...



```
Símbolo del sistema
crackmes.one\TheReverser-Find_password>password.exe
inserisci la password per accedere al programma djeje
password errata.
inserisci la password per accedere al programma djeje
benvenuto
crackmes.one\TheReverser-Find_password>
```

Here we can see how the first time it tells us that the password is incorrect, but in the next attempt if it accepts it. This is probably done to mislead the user, or perhaps it is a programmer error. In any case we already have the solution.

## That's all folks!