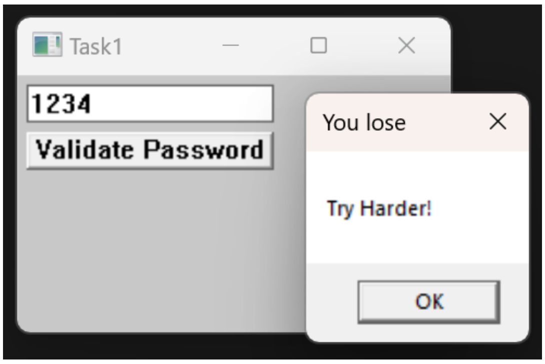
Reverse Engineering Lab 4

Dilirici Radu, 510

Task 1: Debugging in Windows

In primul rand am rulat programul si am incercat o parola aleatoare.

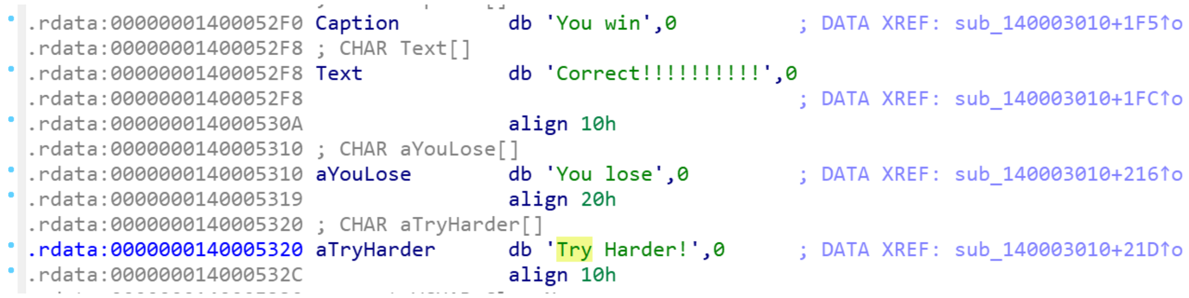


1234

You lose

Try Harder!

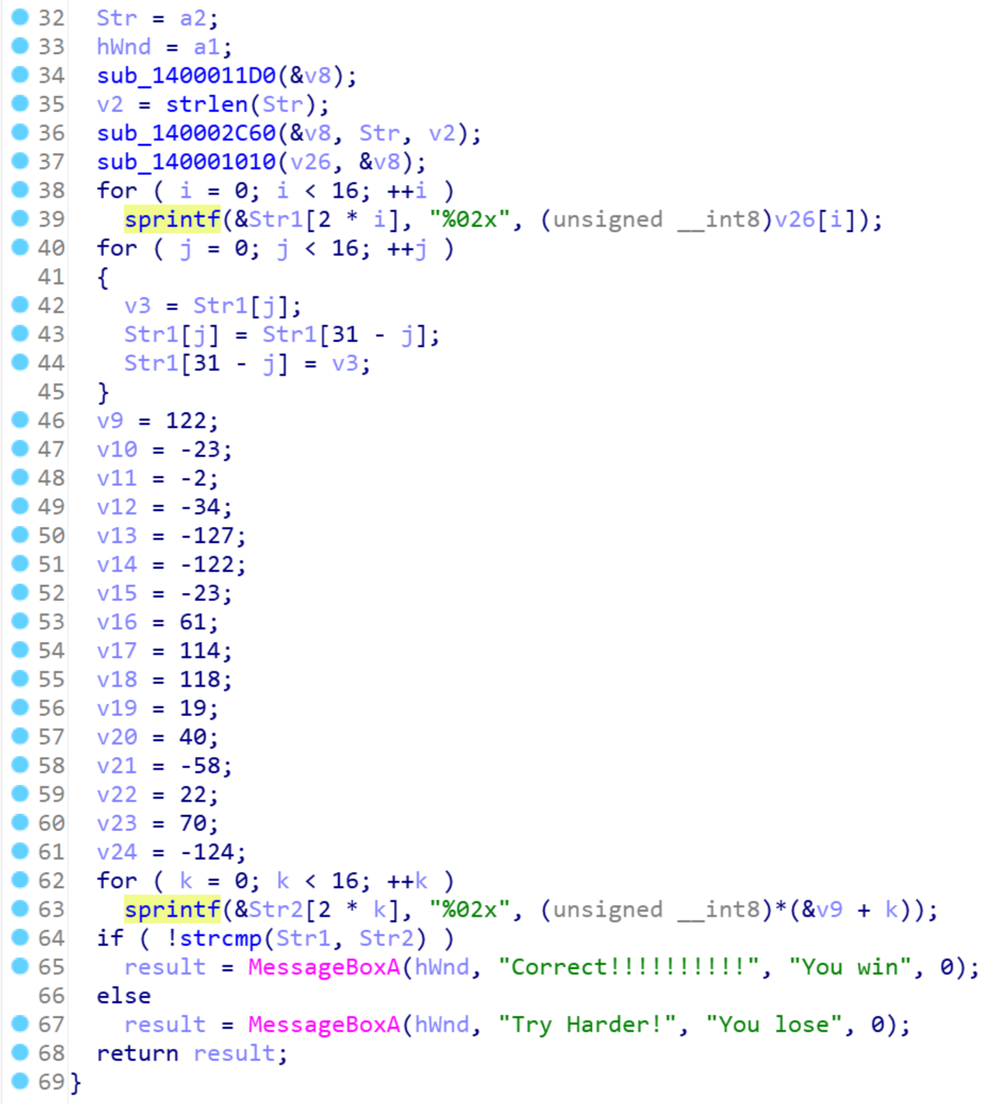
Apoi, am cautat textul *“Try Harder”* in **.rodata**. Uitandu-ma la referinta stringului, am vazult ca este folosit in functia **sub_140003010**, care este responsabila pentru verificarea parolei.



```
.rdata:00000001400052F0 Caption         db 'You win',0          ; DATA XREF: sub_140003010+1F5↑o
.rdata:00000001400052F8 ; CHAR Text[]
.rdata:00000001400052F8 Text            db 'Correct!!!!!!!!!!',0
.rdata:00000001400052F8                                         ; DATA XREF: sub_140003010+1FC↑o
.rdata:000000014000530A                 align 10h
.rdata:0000000140005310 ; CHAR aYouLose[]
.rdata:0000000140005310 aYouLose        db 'You lose',0         ; DATA XREF: sub_140003010+216↑o
.rdata:0000000140005319                 align 20h
.rdata:0000000140005320 ; CHAR aTryHarder[]
.rdata:0000000140005320 aTryHarder      db 'Try Harder!',0      ; DATA XREF: sub_140003010+21D↑o
.rdata:000000014000532C                 align 10h
```

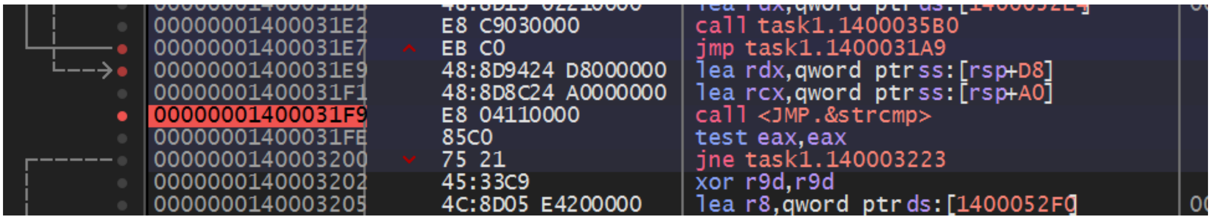
De asemenea, folosind aceasta abordare am gasit si functia **main** sub numele **sub_140002E30**. Am identificat functia **sub_1400035B0** ca fiind **sprintf**. Aceasta se poate observa in urmatoare imagine.

In acelasi timp, putem observa ca la finalul functiei se compara doua stringuri, iar conditia de succes depinde de rezultatul operatiei.



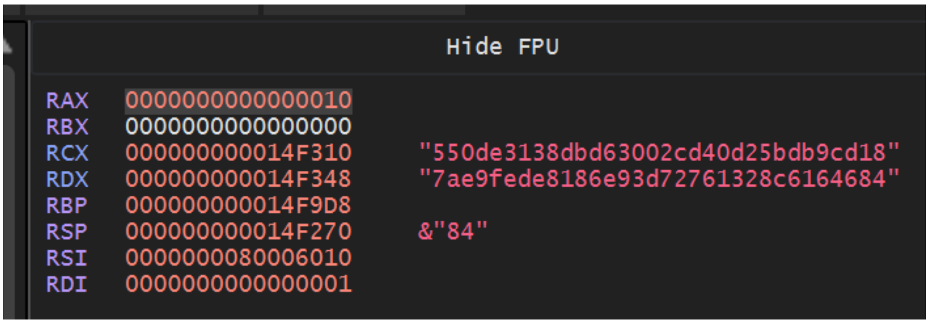
```
Str = a2;
hWnd = a1;
sub_1400011D0(&v8);
v2 = strlen(Str);
sub_140002C60(&v8, Str, v2);
sub_140001010(v26, &v8);
for ( i = 0; i < 16; ++i )
  sprintf(&Str1[2 * i], "%02x", (unsigned __int8)v26[i]);
for ( j = 0; j < 16; ++j )
{
  v3 = Str1[j];
  Str1[j] = Str1[31 - j];
  Str1[31 - j] = v3;
}
v9 = 122;
v10 = -23;
v11 = -2;
v12 = -34;
v13 = -127;
v14 = -122;
v15 = -23;
v16 = 61;
v17 = 114;
v18 = 118;
v19 = 19;
v20 = 40;
v21 = -58;
v22 = 22;
v23 = 70;
v24 = -124;
for ( k = 0; k < 16; ++k )
  sprintf(&Str2[2 * k], "%02x", (unsigned __int8)*(&v9 + k));
if ( !strcmp(Str1, Str2) )
  result = MessageBoxA(hWnd, "Correct!!!!!!!!!!", "You win", 0);
else
  result = MessageBoxA(hWnd, "Try Harder!", "You lose", 0);
return result;
}
```

In **x64dbg** am pus un Breakpoint inainte de a se chema functia de comparare.

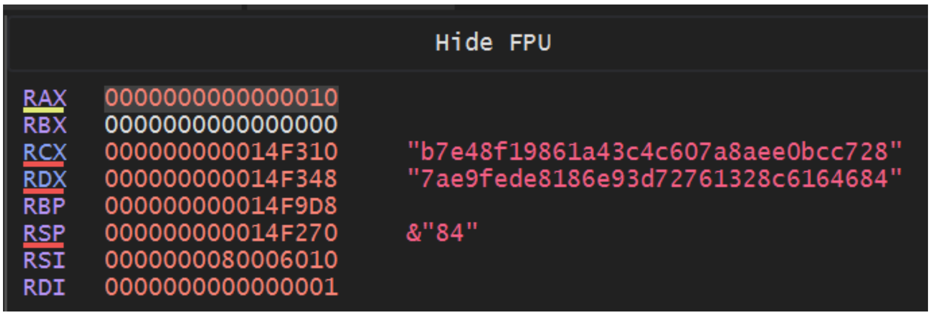


```
00000001400031E2     E8 C9030000           call task1.1400035B0
00000001400031E7     EB C0                 jmp task1.1400031A9
00000001400031E9     48:8D9424 D8000000    lea rdx,qword ptr ss:[rsp+D8]
00000001400031F1     48:8D8C24 A0000000    lea rcx,qword ptr ss:[rsp+A0]
00000001400031F9     E8 04110000           call <JMP.&strcmp>
00000001400031FE     85C0                  test eax,eax
0000000140003200     75 21                 jne task1.140003223
0000000140003202     45:33C9               xor r9d,r9d
0000000140003205     4C:8D05 E4200000      lea r8,qword ptr ds:[1400052F0
```

Dupa rularea mai multor teste cu date de intrare diferite, am observat ca inputul ajunge, dupa niste procesare, in **RCX** si cel mai probabil se compara cu valoarea din **RDX**.

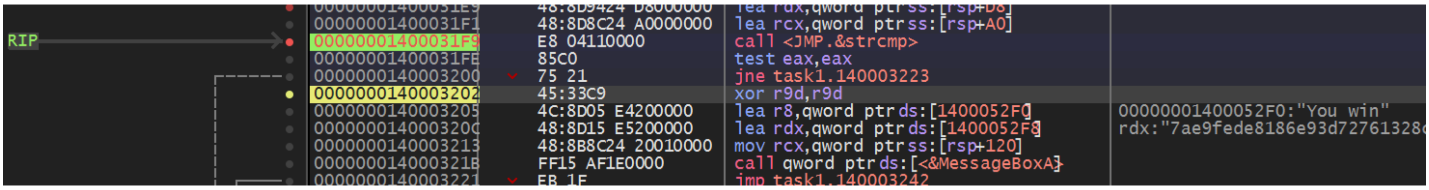


```
Hide FPU

RAX   0000000000000010
RBX   0000000000000000
RCX   000000000014F310     "550de3138dbd63002cd40d25bdb9cd18"
RDX   000000000014F348     "7ae9fede8186e93d72761328c6164684"
RBP   000000000014F9D8
RSP   000000000014F270     &"84"
RSI   0000000080006010
RDI   0000000000000001
```

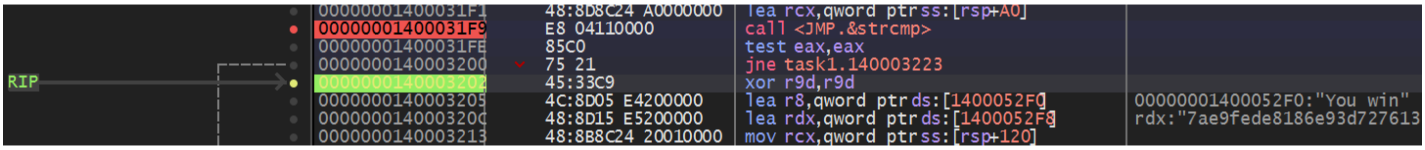


```
Hide FPU

RAX   0000000000000010
RBX   0000000000000000
RCX   000000000014F310     "b7e48f19861a43c4c607a8aee0bcc728"
RDX   000000000014F348     "7ae9fede8186e93d72761328c6164684"
RBP   000000000014F9D8
RSP   000000000014F270     &"84"
RSI   0000000080006010
RDI   0000000000000001
```

Am fortat acceptarea parolei prin setarea **RIP** la adresa de dupa **jne**, ca si cum am fi satisfacut conditia.

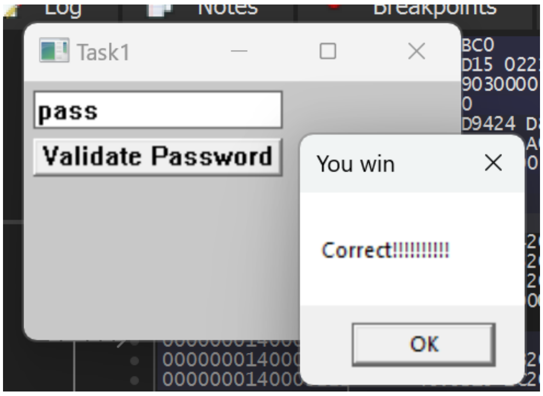


```
       00000001400031E9   48:8D9424 D8000000   lea rdx,qword ptr ss:[rsp+D8]
       00000001400031F1   48:8D8C24 A0000000   lea rcx,qword ptr ss:[rsp+A0]
RIP -> 00000001400031F9   E8 04110000          call <JMP.&strcmp>
       00000001400031FE   85C0                 test eax,eax
       0000000140003200   75 21                jne task1.140003223
       0000000140003202   45:33C9              xor r9d,r9d
       0000000140003205   4C:8D05 E4200000     lea r8,qword ptr ds:[1400052F0]   00000001400052F0:"You win"
       000000014000320C   48:8D15 E5200000     lea rdx,qword ptr ds:[1400052F8]  rdx:"7ae9fede8186e93d72761328
       0000000140003213   48:8B8C24 20010000   mov rcx,qword ptr ss:[rsp+120]
       000000014000321B   FF15 AF1E0000        call qword ptr ds:[<&MessageBoxA>]
       0000000140003221   EB 1F                jmp task1.140003242
```



```
       00000001400031F1   48:8D8C24 A0000000   lea rcx,qword ptr ss:[rsp+A0]
       00000001400031F9   E8 04110000          call <JMP.&strcmp>
       00000001400031FE   85C0                 test eax,eax
       0000000140003200   75 21                jne task1.140003223
RIP -> 0000000140003202   45:33C9              xor r9d,r9d
       0000000140003205   4C:8D05 E4200000     lea r8,qword ptr ds:[1400052F0]   00000001400052F0:"You win"
       000000014000320C   48:8D15 E5200000     lea rdx,qword ptr ds:[1400052F8]  rdx:"7ae9fede8186e93d727613
       0000000140003213   48:8B8C24 20010000   mov rcx,qword ptr ss:[rsp+120]
```

Astfel, am programul a “crezut” ca am introdus parola corecta.

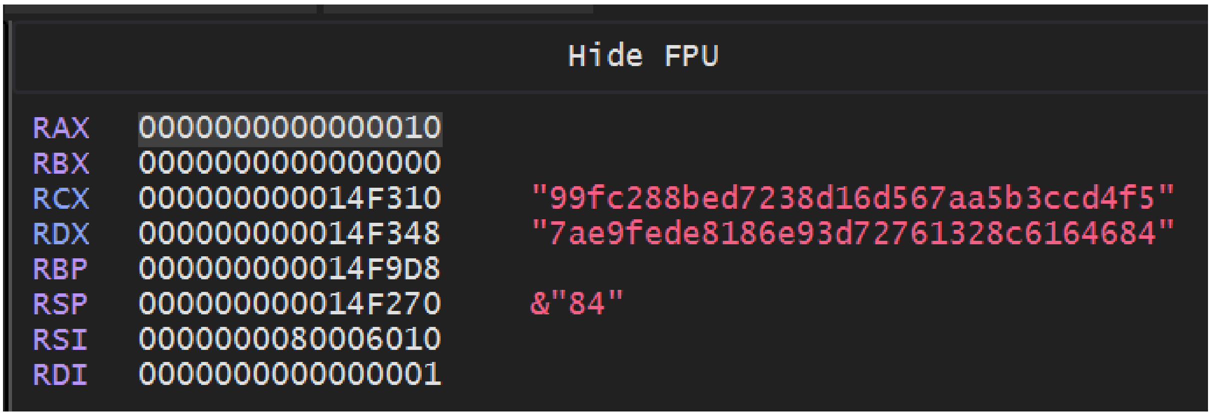


pass

You win

Correct!!!!!!!!!

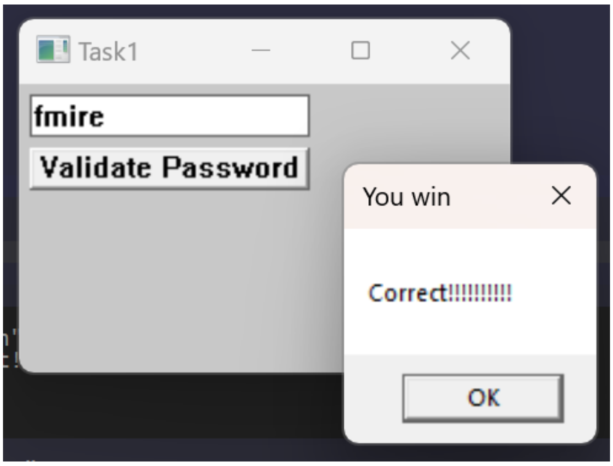
Introducerea parolei *“password”* rezulta in sirul *“99fc288bed7238d16d567aa5b3ccd4f5”*. Dupa cateva cautari, am aflat ca inversul acestuia (*“5f4dcc3b5aa765d61d8327deb882cf99”*) este hash-ul MD5 corespunzator textului *“password”*.



```
                       Hide FPU

RAX   0000000000000010
RBX   0000000000000000
RCX   000000000014F310     "99fc288bed7238d16d567aa5b3ccd4f5"
RDX   000000000014F348     "7ae9fede8186e93d72761328c6164684"
RBP   000000000014F9D8
RSP   000000000014F270     &"84"
RSI   0000000080006010
RDI   0000000000000001
```

De aici, putem trage concluzia ca programul calculeaza hash-ul MD5 al inputului, il inverseaza, dupa care il compara cu un alt string. In acest caz, pentru a afla parola, putem urma procesul in sens invers.

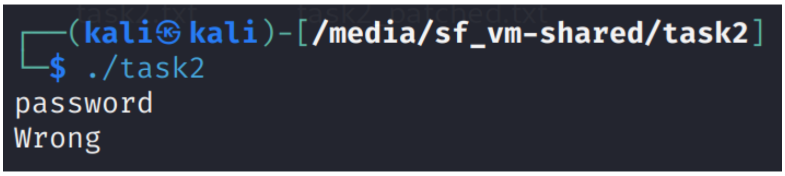
Inversam stringul cu care se compara, de unde rezulta textul *“4864616c82316727d39e6818edef9ea7”*. Dupa utilizarea catorva site-uri care “inverseaza” hash-uri MD5, gasim ca parola este *“fmire”*. Am avut succes cu pagina https://md5.web-max.ca/index.php.



Correct!!!!!!!!!!

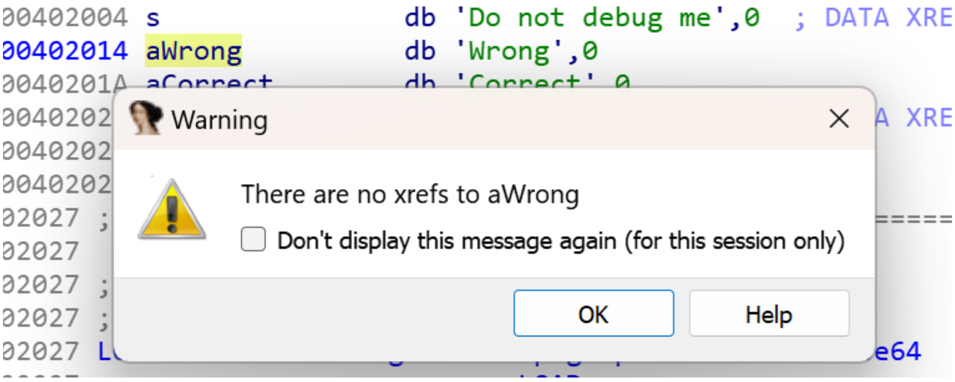
Task 2: Debugging in Linux

La rularea programului observam ca este transmis mesajul *“Wrong”* la introducerea unei parole gresite.



```
┌──(kali㉿kali)-[/media/sf_vm-shared/task2]
└─$ ./task2
password
Wrong
```

Folosind aceeasi metoda ca pana acum, nu putem ajunge la functia de verificare. IDA nu poate gasi nicio referinta catre stringul *“Wrong”*. In acest moment nu am stiut care era cauza.



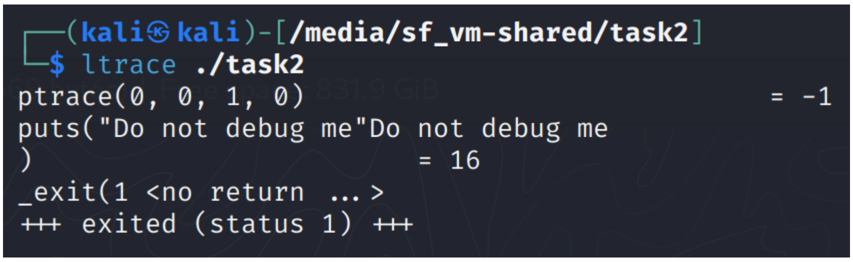
```
00402004 s                 db 'Do not debug me',0  ; DATA XRE
00402014 aWrong            db 'Wrong',0
0040201A aCorrect          db 'Correct',0
```

Warning

There are no xrefs to aWrong

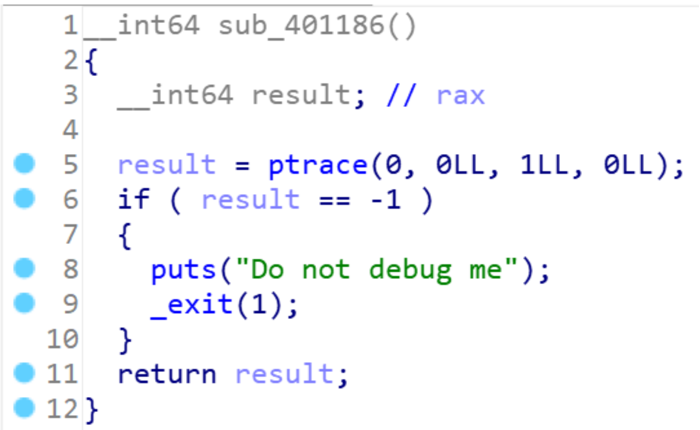
☐ Don't display this message again (for this session only)

De altfel, nu putem folosi **ltrace** pe programul respectiv. Este aruncata o eroare si mesajul *“Do not debug me”*.



```
┌──(kali㉿kali)-[/media/sf_vm-shared/task2]
└─$ ltrace ./task2
ptrace(0, 0, 1, 0)                                   = -1
puts("Do not debug me"Do not debug me
)                                      = 16
_exit(1 <no return ...>
+++ exited (status 1) +++
```

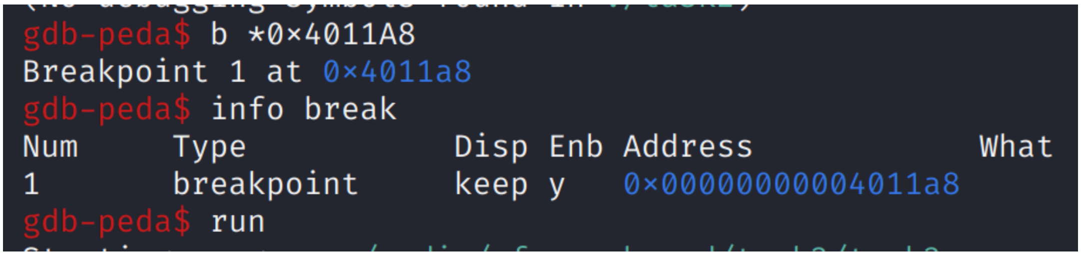
Folosind IDA, gasim acest segment de cod in care se foloseste **ptrace**.



```
__int64 sub_401186()
{
  __int64 result; // rax

  result = ptrace(0, 0LL, 1LL, 0LL);
  if ( result == -1 )
  {
    puts("Do not debug me");
    _exit(1);
  }
  return result;
}
```

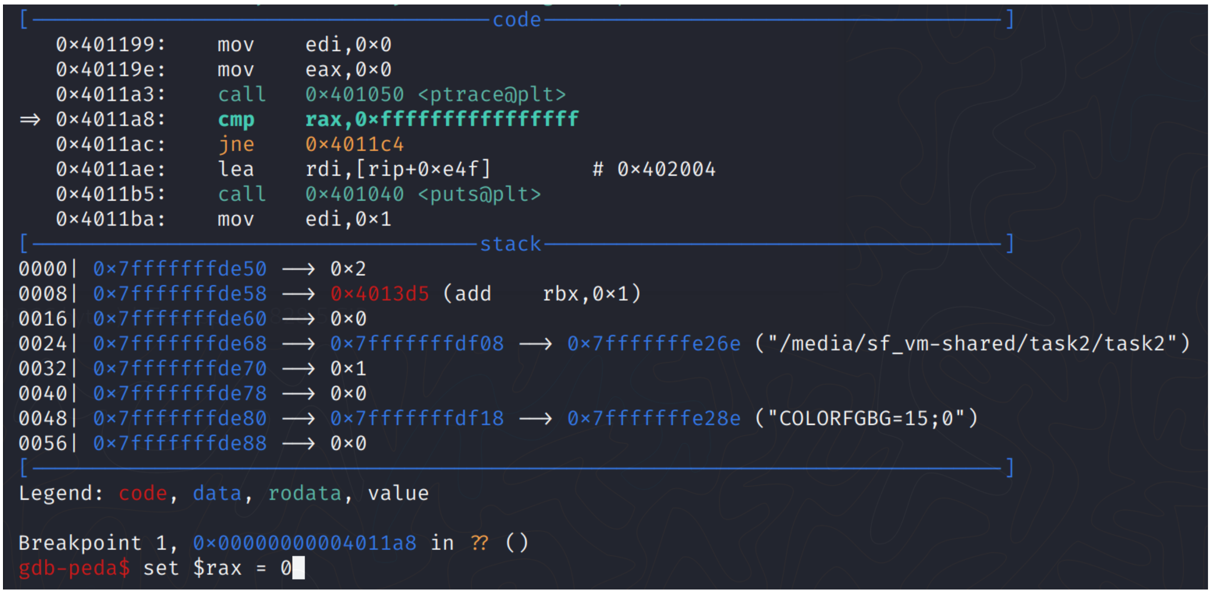
Un proces poate avea un singur tracer atasat la un moment dat. Din acest motiv, programul nu poate continua (in mod normal) daca incercam sa folsim **ltrace** sau **gdb**.

Am deschis programul cu **gdb** si am setat un **breakpoint** inainte de aceasta verificare (comparatia cu -1).



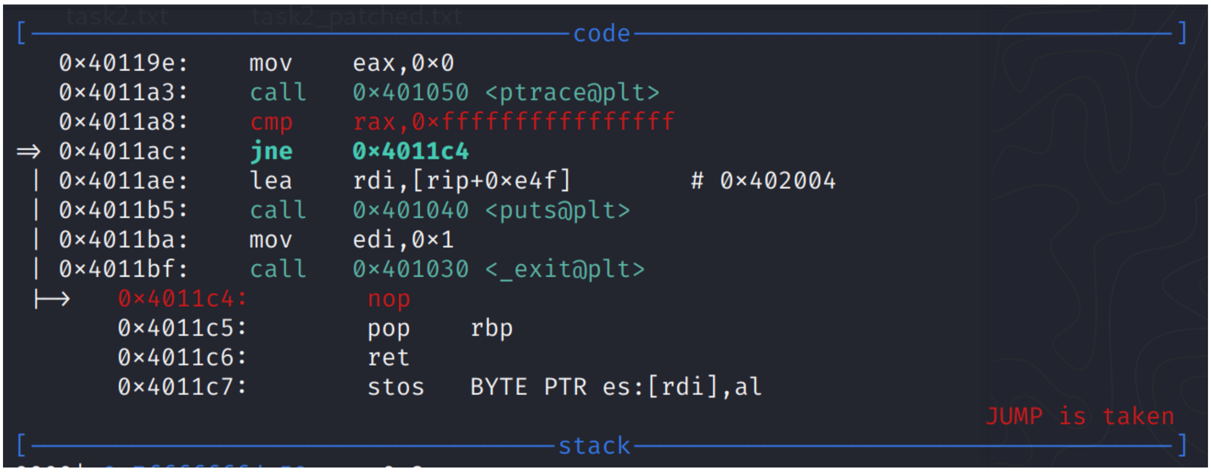
```
gdb-peda$ b *0x4011A8
Breakpoint 1 at 0x4011a8
gdb-peda$ info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00000000004011a8
gdb-peda$ run
```

La oprirea programului, am modificat registrul **RAX** astfel incat sa fie diferit de **-1**.



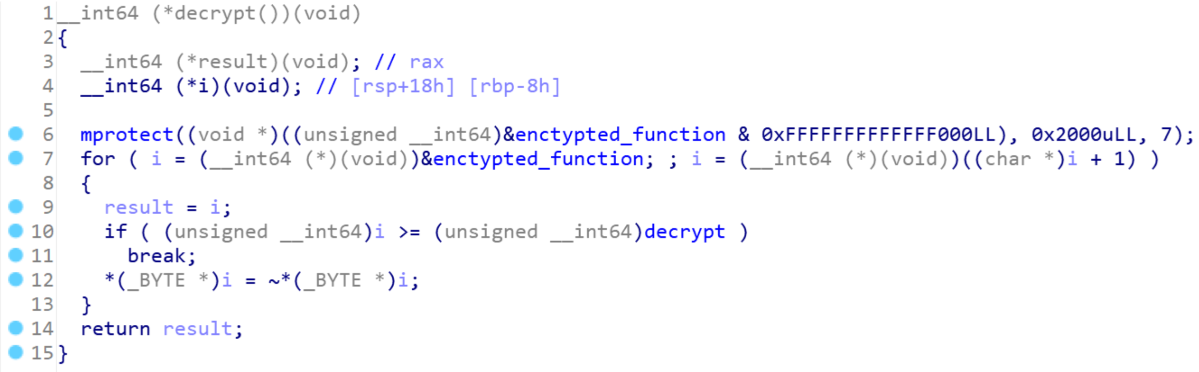
```
[------------------------------------code------------------------------------]
   0x401199:    mov    edi,0x0
   0x40119e:    mov    eax,0x0
   0x4011a3:    call   0x401050 <ptrace@plt>
=> 0x4011a8:    cmp    rax,0xffffffffffffffff
   0x4011ac:    jne    0x4011c4
   0x4011ae:    lea    rdi,[rip+0xe4f]        # 0x402004
   0x4011b5:    call   0x401040 <puts@plt>
   0x4011ba:    mov    edi,0x1
[-----------------------------------stack------------------------------------]
0000| 0x7fffffffde50 --> 0x2
0008| 0x7fffffffde58 --> 0x4013d5 (add    rbx,0x1)
0016| 0x7fffffffde60 --> 0x0
0024| 0x7fffffffde68 --> 0x7fffffffdf08 --> 0x7fffffffe26e ("/media/sf_vm-shared/task2/task2")
0032| 0x7fffffffde70 --> 0x1
0040| 0x7fffffffde78 --> 0x0
0048| 0x7fffffffde80 --> 0x7fffffffdf18 --> 0x7fffffffe28e ("COLORFGBG=15;0")
0056| 0x7fffffffde88 --> 0x0
[----------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x00000000004011a8 in ?? ()
gdb-peda$ set $rax = 0
```

In acest caz programul sare la adresa **0x4011c4**, evitand o iesire prematura.



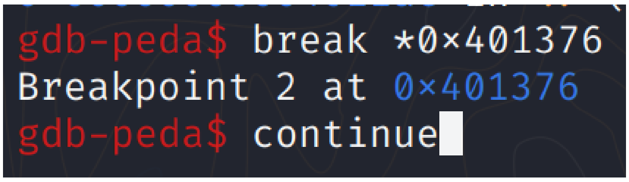
```
[------------------------------------code------------------------------------]
   0x40119e:    mov    eax,0x0
   0x4011a3:    call   0x401050 <ptrace@plt>
   0x4011a8:    cmp    rax,0xffffffffffffffff
=> 0x4011ac:    jne    0x4011c4
 | 0x4011ae:    lea    rdi,[rip+0xe4f]        # 0x402004
 | 0x4011b5:    call   0x401040 <puts@plt>
 | 0x4011ba:    mov    edi,0x1
 | 0x4011bf:    call   0x401030 <_exit@plt>
 |->   0x4011c4:        nop
       0x4011c5:        pop    rbp
       0x4011c6:        ret
       0x4011c7:        stos   BYTE PTR es:[rdi],al
                                                                  JUMP is taken
[-----------------------------------stack------------------------------------]
```

Putem observa ca este decriptata memoria incepand de la adresa functiei secrete, numita aici **encrypted_function** pana la functia de decriptare **decrypt**. Acest lucru se intampla pentru ca cele doua functii se afla una dupa cealalta. Deci se decripteaza toata functia **encrypted_function**. Decriptarea presupune negarea bitilor.



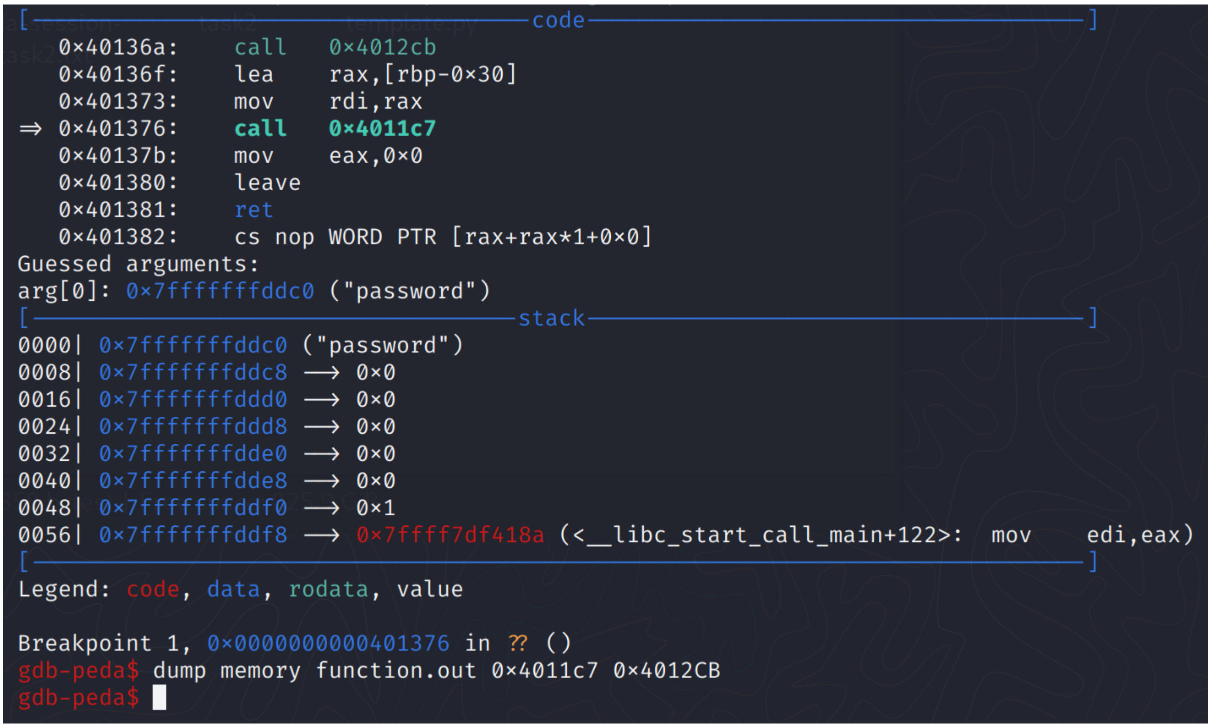
```
__int64 (*decrypt())(void)
{
  __int64 (*result)(void); // rax
  __int64 (*i)(void); // [rsp+18h] [rbp-8h]

  mprotect((void *)((unsigned __int64)&enctypted_function & 0xFFFFFFFFFFFFF000LL), 0x2000uLL, 7);
  for ( i = (__int64 (*)(void))&enctypted_function; ; i = (__int64 (*)(void))((char *)i + 1) )
  {
    result = i;
    if ( (unsigned __int64)i >= (unsigned __int64)decrypt )
      break;
    *(_BYTE *)i = ~*(_BYTE *)i;
  }
  return result;
}
```

Am setat un nou **breakpoint** inainte de chemarea functiei de verificare a parolei. In momentul acela, functia este decriptata.



```
gdb-peda$ break *0x401376
Breakpoint 2 at 0x401376
gdb-peda$ continue
```

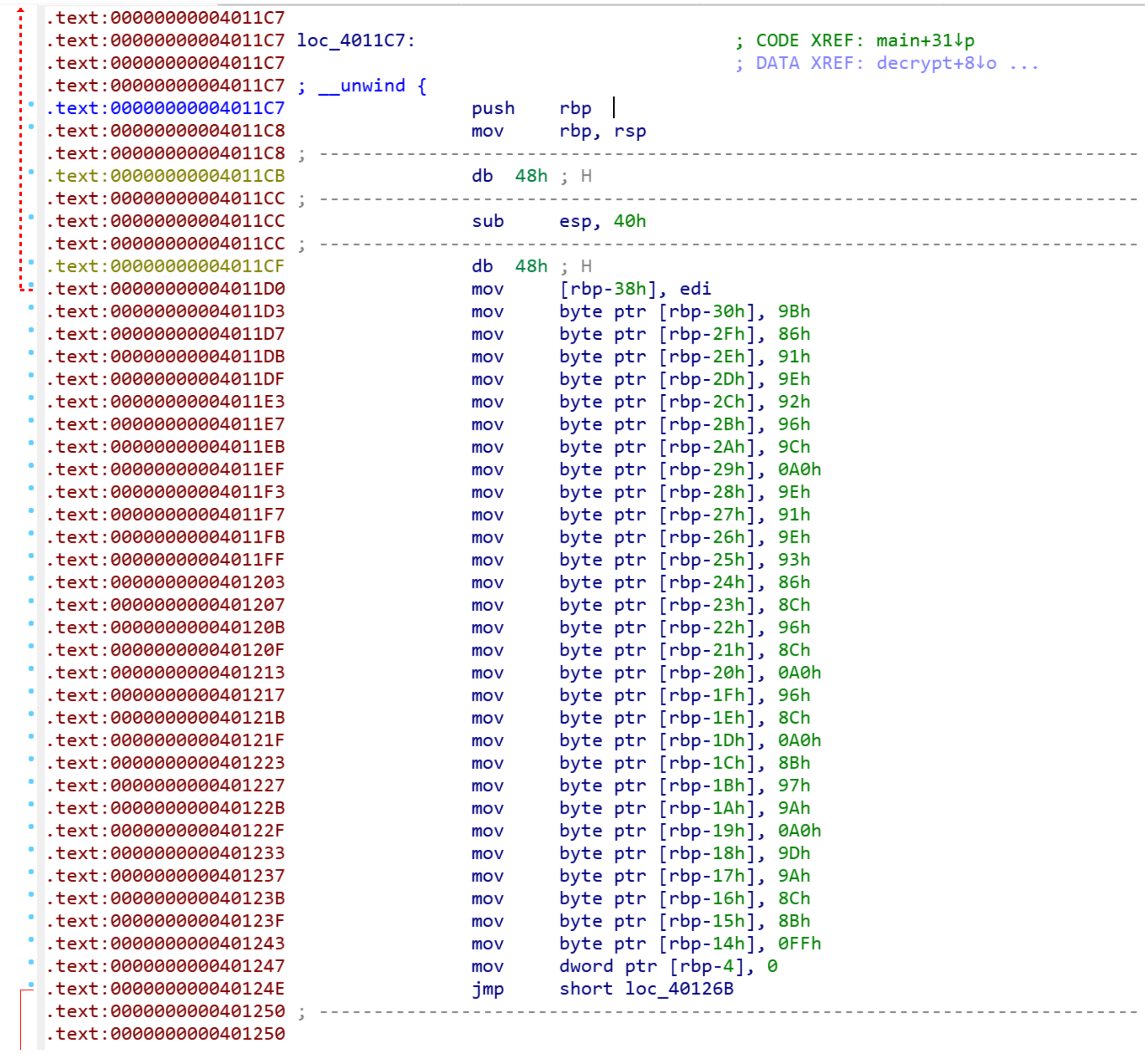
Cand programul ajunge in acest punct, putem extrage functia din memoria programului prin comanda **dump**.



```
[----------------------------------code----------------------------------]
   0x40136a:    call   0x4012cb
   0x40136f:    lea    rax,[rbp-0x30]
   0x401373:    mov    rdi,rax
=> 0x401376:    call   0x4011c7
   0x40137b:    mov    eax,0x0
   0x401380:    leave
   0x401381:    ret
   0x401382:    cs nop WORD PTR [rax+rax*1+0x0]
Guessed arguments:
arg[0]: 0x7fffffffddc0 ("password")
[----------------------------------stack---------------------------------]
0000| 0x7fffffffddc0 ("password")
0008| 0x7fffffffddc8 --> 0x0
0016| 0x7fffffffddd0 --> 0x0
0024| 0x7fffffffddd8 --> 0x0
0032| 0x7fffffffdde0 --> 0x0
0040| 0x7fffffffdde8 --> 0x0
0048| 0x7fffffffddf0 --> 0x1
0056| 0x7fffffffddf8 --> 0x7ffff7df418a (<__libc_start_call_main+122>:  mov    edi,eax)
[------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x0000000000401376 in ?? ()
gdb-peda$ dump memory function.out 0x4011c7 0x4012CB
gdb-peda$
```

Am creat un script pentru IDA, disponibil in **patch_bytes.py**, care suprascrie bitii din functia criptata cu cei decriptati, adica continutul fisierului **function.out**.

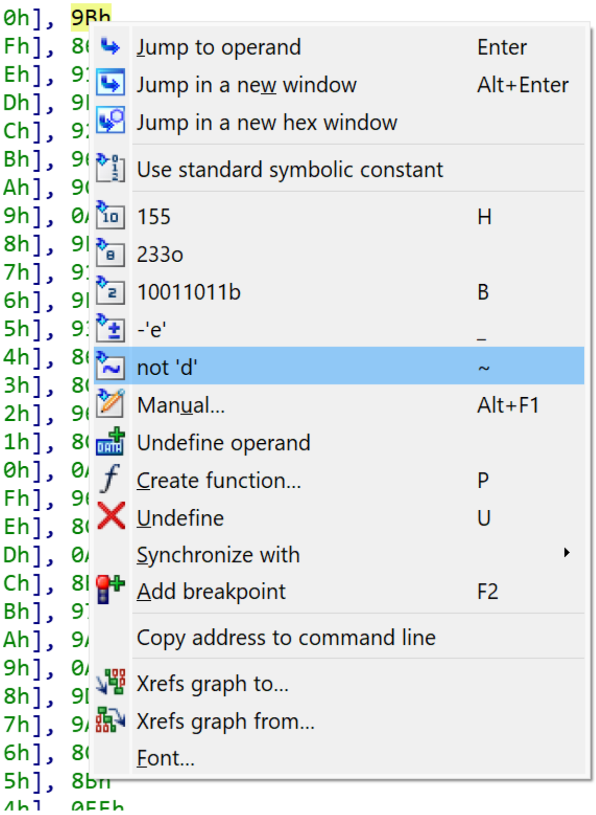
Dupa aceasta actiune, putem observa ca sunt introduse niste valori intr-o lista. Prespunerea este ca aceste valori reprezinta parola corecta. IDA a refuzat sa transforme codul assembly in pseudocod. Este posibil sa fi folosit un range gresit la extragerea memoriei.



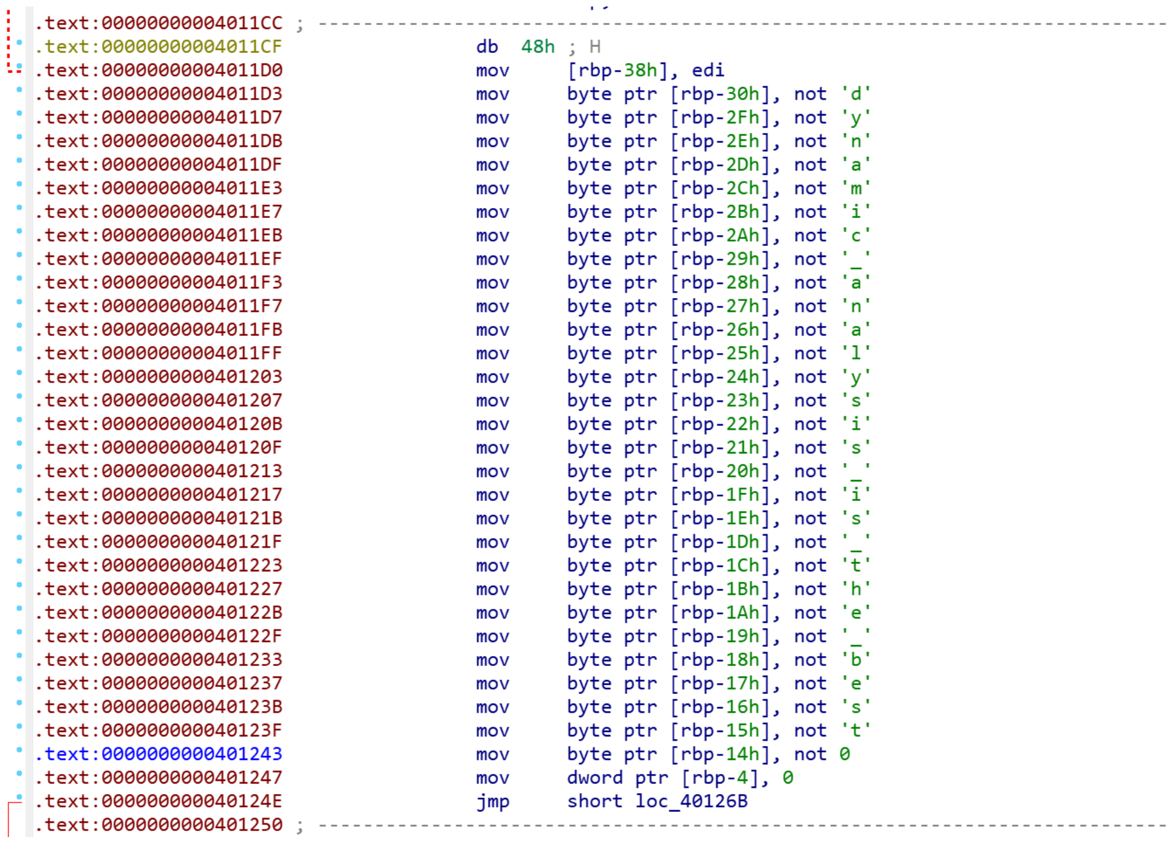
```
.text:00000000004011C7
.text:00000000004011C7 loc_4011C7:                             ; CODE XREF: main+31↓p
.text:00000000004011C7                                         ; DATA XREF: decrypt+8↓o ...
.text:00000000004011C7 ; __unwind {
.text:00000000004011C7                 push    rbp
.text:00000000004011C8                 mov     rbp, rsp
.text:00000000004011C8 ; ---------------------------------------------------------------------------
.text:00000000004011CB                 db  48h ; H
.text:00000000004011CC ; ---------------------------------------------------------------------------
.text:00000000004011CC                 sub     esp, 40h
.text:00000000004011CC ; ---------------------------------------------------------------------------
.text:00000000004011CF                 db  48h ; H
.text:00000000004011D0                 mov     [rbp-38h], edi
.text:00000000004011D3                 mov     byte ptr [rbp-30h], 9Bh
.text:00000000004011D7                 mov     byte ptr [rbp-2Fh], 86h
.text:00000000004011DB                 mov     byte ptr [rbp-2Eh], 91h
.text:00000000004011DF                 mov     byte ptr [rbp-2Dh], 9Eh
.text:00000000004011E3                 mov     byte ptr [rbp-2Ch], 92h
.text:00000000004011E7                 mov     byte ptr [rbp-2Bh], 96h
.text:00000000004011EB                 mov     byte ptr [rbp-2Ah], 9Ch
.text:00000000004011EF                 mov     byte ptr [rbp-29h], 0A0h
.text:00000000004011F3                 mov     byte ptr [rbp-28h], 9Eh
.text:00000000004011F7                 mov     byte ptr [rbp-27h], 91h
.text:00000000004011FB                 mov     byte ptr [rbp-26h], 9Eh
.text:00000000004011FF                 mov     byte ptr [rbp-25h], 93h
.text:0000000000401203                 mov     byte ptr [rbp-24h], 86h
.text:0000000000401207                 mov     byte ptr [rbp-23h], 8Ch
.text:000000000040120B                 mov     byte ptr [rbp-22h], 96h
.text:000000000040120F                 mov     byte ptr [rbp-21h], 8Ch
.text:0000000000401213                 mov     byte ptr [rbp-20h], 0A0h
.text:0000000000401217                 mov     byte ptr [rbp-1Fh], 96h
.text:000000000040121B                 mov     byte ptr [rbp-1Eh], 8Ch
.text:000000000040121F                 mov     byte ptr [rbp-1Dh], 0A0h
.text:0000000000401223                 mov     byte ptr [rbp-1Ch], 8Bh
.text:0000000000401227                 mov     byte ptr [rbp-1Bh], 97h
.text:000000000040122B                 mov     byte ptr [rbp-1Ah], 9Ah
.text:000000000040122F                 mov     byte ptr [rbp-19h], 0A0h
.text:0000000000401233                 mov     byte ptr [rbp-18h], 9Dh
.text:0000000000401237                 mov     byte ptr [rbp-17h], 9Ah
.text:000000000040123B                 mov     byte ptr [rbp-16h], 8Ch
.text:000000000040123F                 mov     byte ptr [rbp-15h], 8Bh
.text:0000000000401243                 mov     byte ptr [rbp-14h], 0FFh
.text:0000000000401247                 mov     dword ptr [rbp-4], 0
.text:000000000040124E                 jmp     short loc_40126B
.text:0000000000401250 ; ---------------------------------------------------------------------------
.text:0000000000401250
```

Continutul listei nu rezulta intr-un sir de caractere citibile. Pare ca toate valorile sunt negate. Am aplicat operatia de negare pe fiecare element si le-am transformat in caractere.

Pare ca lista are aceste valori opuse si la rularea normala a programului (lucru observat prin analizarea cu **gdb**).

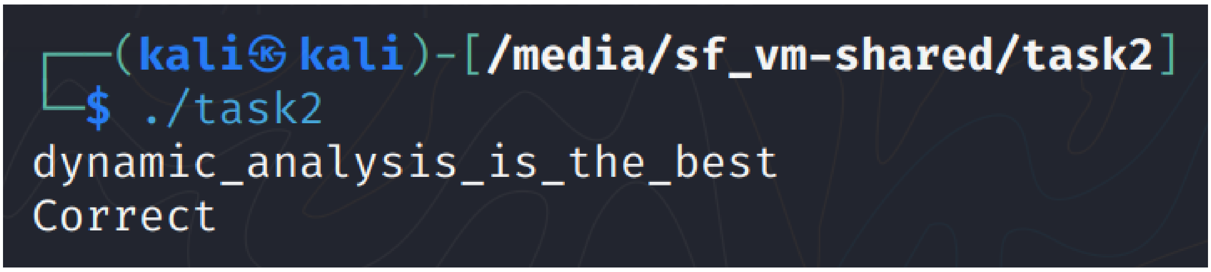


Dupa aplicarea acestor schimbari, putem observa care este parola: “*dynamic_analysis_is_the_best*”.



```
.text:00000000004011CC ; ---------------------------------------------------------------------------
.text:00000000004011CF                 db  48h ; H
.text:00000000004011D0                 mov     [rbp-38h], edi
.text:00000000004011D3                 mov     byte ptr [rbp-30h], not 'd'
.text:00000000004011D7                 mov     byte ptr [rbp-2Fh], not 'y'
.text:00000000004011DB                 mov     byte ptr [rbp-2Eh], not 'n'
.text:00000000004011DF                 mov     byte ptr [rbp-2Dh], not 'a'
.text:00000000004011E3                 mov     byte ptr [rbp-2Ch], not 'm'
.text:00000000004011E7                 mov     byte ptr [rbp-2Bh], not 'i'
.text:00000000004011EB                 mov     byte ptr [rbp-2Ah], not 'c'
.text:00000000004011EF                 mov     byte ptr [rbp-29h], not '_'
.text:00000000004011F3                 mov     byte ptr [rbp-28h], not 'a'
.text:00000000004011F7                 mov     byte ptr [rbp-27h], not 'n'
.text:00000000004011FB                 mov     byte ptr [rbp-26h], not 'a'
.text:00000000004011FF                 mov     byte ptr [rbp-25h], not 'l'
.text:0000000000401203                 mov     byte ptr [rbp-24h], not 'y'
.text:0000000000401207                 mov     byte ptr [rbp-23h], not 's'
.text:000000000040120B                 mov     byte ptr [rbp-22h], not 'i'
.text:000000000040120F                 mov     byte ptr [rbp-21h], not 's'
.text:0000000000401213                 mov     byte ptr [rbp-20h], not '_'
.text:0000000000401217                 mov     byte ptr [rbp-1Fh], not 'i'
.text:000000000040121B                 mov     byte ptr [rbp-1Eh], not 's'
.text:000000000040121F                 mov     byte ptr [rbp-1Dh], not '_'
.text:0000000000401223                 mov     byte ptr [rbp-1Ch], not 't'
.text:0000000000401227                 mov     byte ptr [rbp-1Bh], not 'h'
.text:000000000040122B                 mov     byte ptr [rbp-1Ah], not 'e'
.text:000000000040122F                 mov     byte ptr [rbp-19h], not '_'
.text:0000000000401233                 mov     byte ptr [rbp-18h], not 'b'
.text:0000000000401237                 mov     byte ptr [rbp-17h], not 'e'
.text:000000000040123B                 mov     byte ptr [rbp-16h], not 's'
.text:000000000040123F                 mov     byte ptr [rbp-15h], not 't'
.text:0000000000401243                 mov     byte ptr [rbp-14h], not 0
.text:0000000000401247                 mov     dword ptr [rbp-4], 0
.text:000000000040124E                 jmp     short loc_40126B
.text:0000000000401250 ; ---------------------------------------------------------------------------
```

Iar prin rularea normala a programului, putem observa ca intr-adevar asta este parola.



```
┌──(kali㉿kali)-[/media/sf_vm-shared/task2]
└─$ ./task2
dynamic_analysis_is_the_best
Correct
```