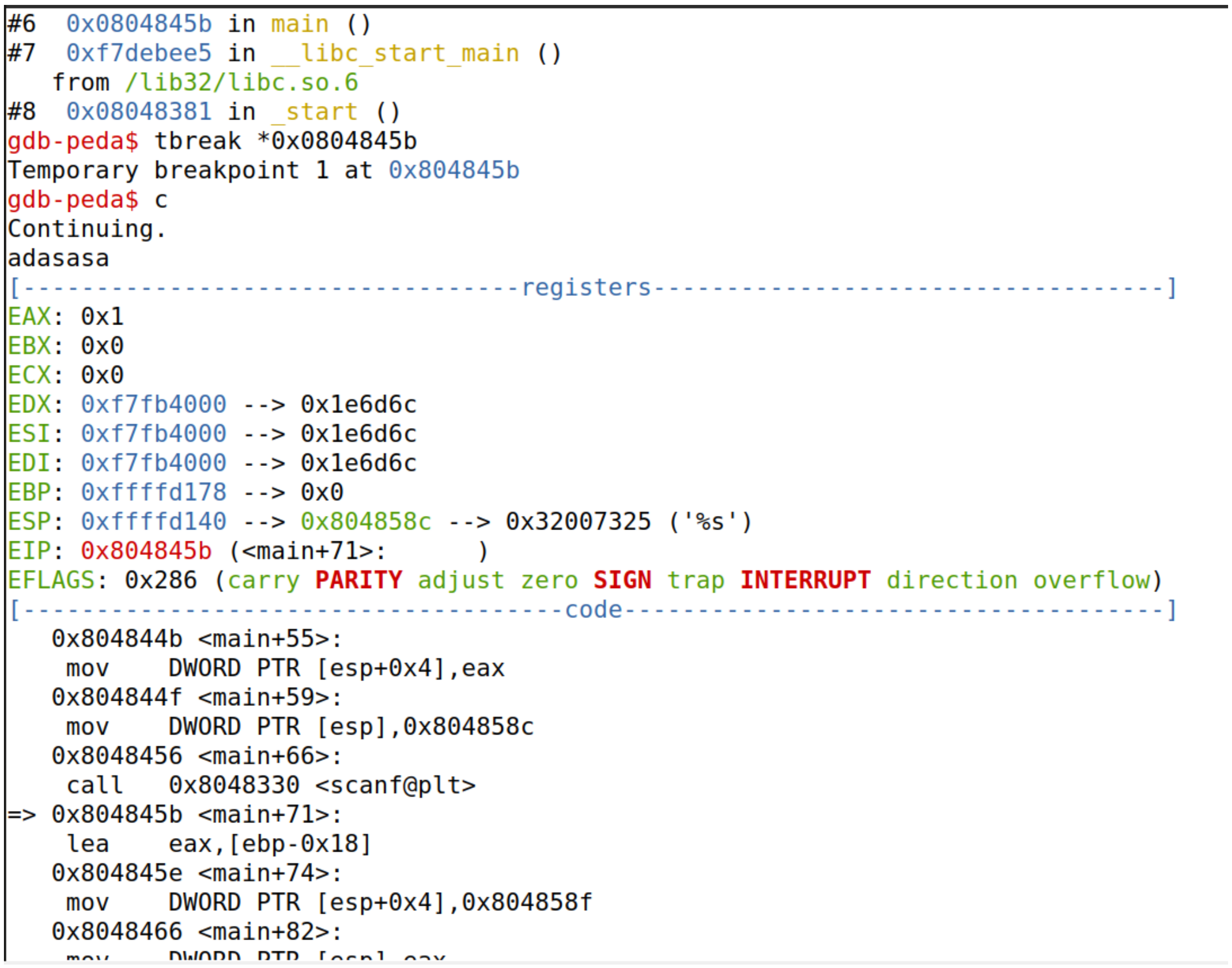
CS255 - Computer Security
LAB 1: Reverse Engineering

The files downloaded from the given folder IOLI-CRACKME are:

1. crackme0x00
2. crackme0x01
3. crackme0x02
4. crackme0x03
5. crackme0x04
6. crackme0x05
7. crackme0x06

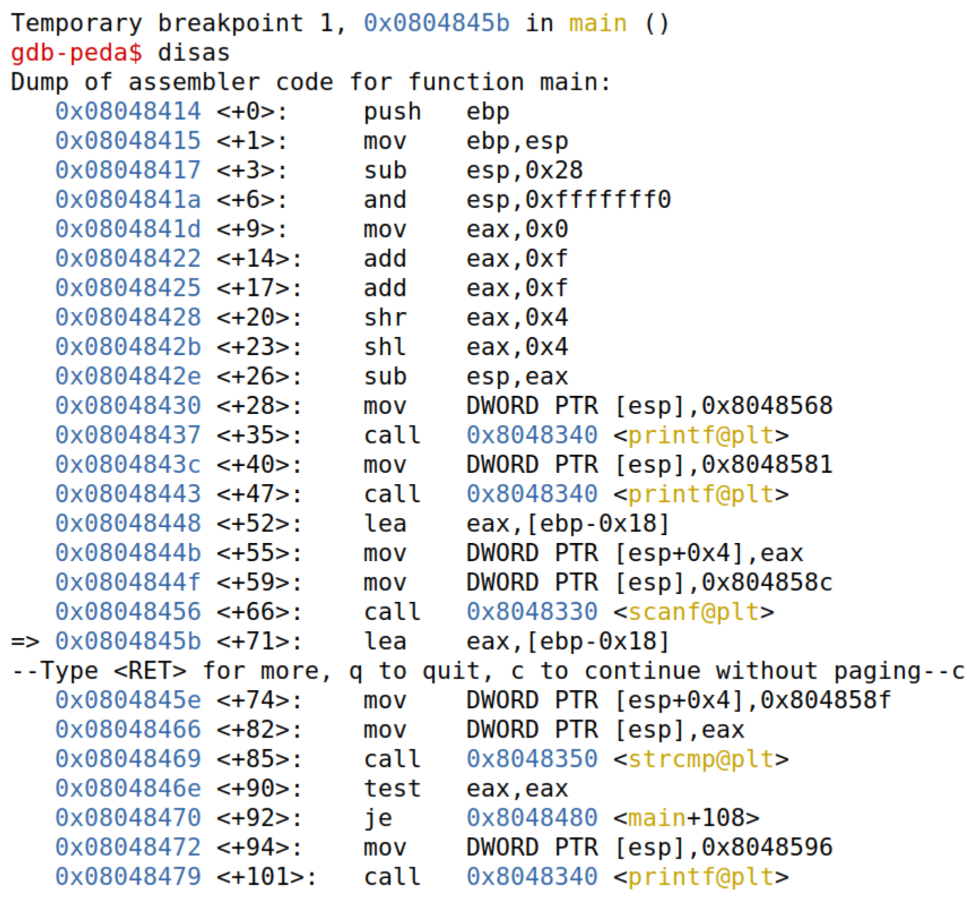
1. crackme0x00:

The file has been debugged as described in the tutorial, and the running of the commands gives us the following output.



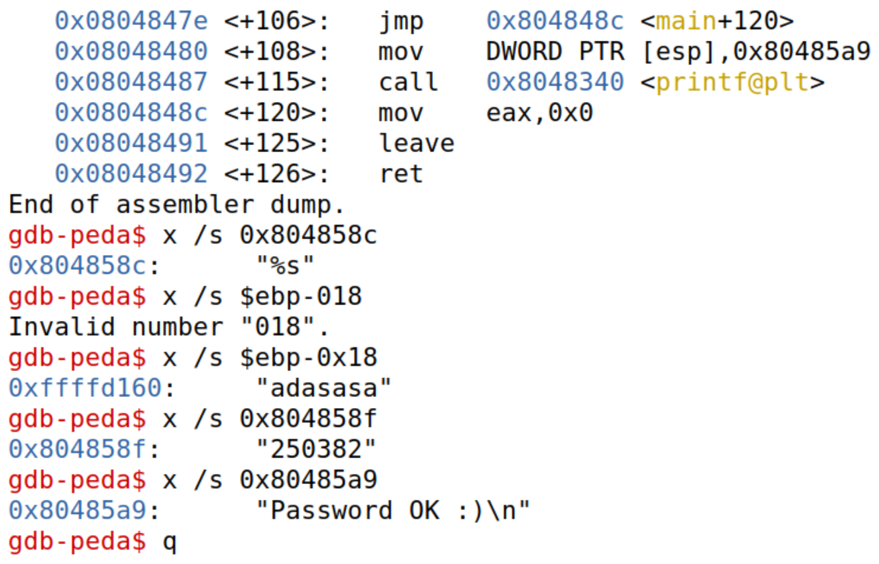
```
#6  0x0804845b in main ()
#7  0xf7debee5 in __libc_start_main ()
   from /lib32/libc.so.6
#8  0x08048381 in _start ()
gdb-peda$ tbreak *0x0804845b
Temporary breakpoint 1 at 0x804845b
gdb-peda$ c
Continuing.
adasasa
[----------------------------------registers-----------------------------------]
EAX: 0x1
EBX: 0x0
ECX: 0x0
EDX: 0xf7fb4000 --> 0x1e6d6c
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0xffffd178 --> 0x0
ESP: 0xffffd140 --> 0x804858c --> 0x32007325 ('%s')
EIP: 0x804845b (<main+71>:      )
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x804844b <main+55>:
    mov    DWORD PTR [esp+0x4],eax
   0x804844f <main+59>:
    mov    DWORD PTR [esp],0x804858c
   0x8048456 <main+66>:
    call   0x8048330 <scanf@plt>
=> 0x804845b <main+71>:
    lea    eax,[ebp-0x18]
   0x804845e <main+74>:
    mov    DWORD PTR [esp+0x4],0x804858f
   0x8048466 <main+82>:
```

The input of the program is taken after the application of breakpoint at the start of the main, at 0x804845b. This gets stored in the line <main+71>, in the address ebp-0x18 of the base pointer stack. This is found by the command disas.



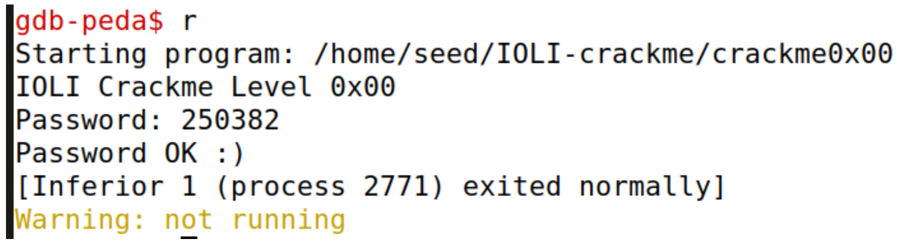
```
Temporary breakpoint 1, 0x0804845b in main ()
gdb-peda$ disas
Dump of assembler code for function main:
   0x08048414 <+0>:     push   ebp
   0x08048415 <+1>:     mov    ebp,esp
   0x08048417 <+3>:     sub    esp,0x28
   0x0804841a <+6>:     and    esp,0xfffffff0
   0x0804841d <+9>:     mov    eax,0x0
   0x08048422 <+14>:    add    eax,0xf
   0x08048425 <+17>:    add    eax,0xf
   0x08048428 <+20>:    shr    eax,0x4
   0x0804842b <+23>:    shl    eax,0x4
   0x0804842e <+26>:    sub    esp,eax
   0x08048430 <+28>:    mov    DWORD PTR [esp],0x8048568
   0x08048437 <+35>:    call   0x8048340 <printf@plt>
   0x0804843c <+40>:    mov    DWORD PTR [esp],0x8048581
   0x08048443 <+47>:    call   0x8048340 <printf@plt>
   0x08048448 <+52>:    lea    eax,[ebp-0x18]
   0x0804844b <+55>:    mov    DWORD PTR [esp+0x4],eax
   0x0804844f <+59>:    mov    DWORD PTR [esp],0x804858c
   0x08048456 <+66>:    call   0x8048330 <scanf@plt>
=> 0x0804845b <+71>:    lea    eax,[ebp-0x18]
--Type <RET> for more, q to quit, c to continue without paging--c
   0x0804845e <+74>:    mov    DWORD PTR [esp+0x4],0x804858f
   0x08048466 <+82>:    mov    DWORD PTR [esp],eax
   0x08048469 <+85>:    call   0x8048350 <strcmp@plt>
   0x0804846e <+90>:    test   eax,eax
   0x08048470 <+92>:    je     0x8048480 <main+108>
   0x08048472 <+94>:    mov    DWORD PTR [esp],0x8048596
   0x08048479 <+101>:   call   0x8048340 <printf@plt>
```

The function call <strcmp@plt> calls the string compare function, which checks the values of pointers moved in the previous two iterations. As we know the value of eax is ebp-0x18 since it stored the value of the input, the other pointer has to be the password value.



```
   0x0804847e <+106>:   jmp    0x804848c <main+120>
   0x08048480 <+108>:   mov    DWORD PTR [esp],0x80485a9
   0x08048487 <+115>:   call   0x8048340 <printf@plt>
   0x0804848c <+120>:   mov    eax,0x0
   0x08048491 <+125>:   leave
   0x08048492 <+126>:   ret
End of assembler dump.
gdb-peda$ x /s 0x804858c
0x804858c:      "%s"
gdb-peda$ x /s $ebp-018
Invalid number "018".
gdb-peda$ x /s $ebp-0x18
0xffffd160:     "adasasa"
gdb-peda$ x /s 0x804858f
0x804858f:      "250382"
gdb-peda$ x /s 0x80485a9
0x80485a9:      "Password OK :)\n"
gdb-peda$ q
```

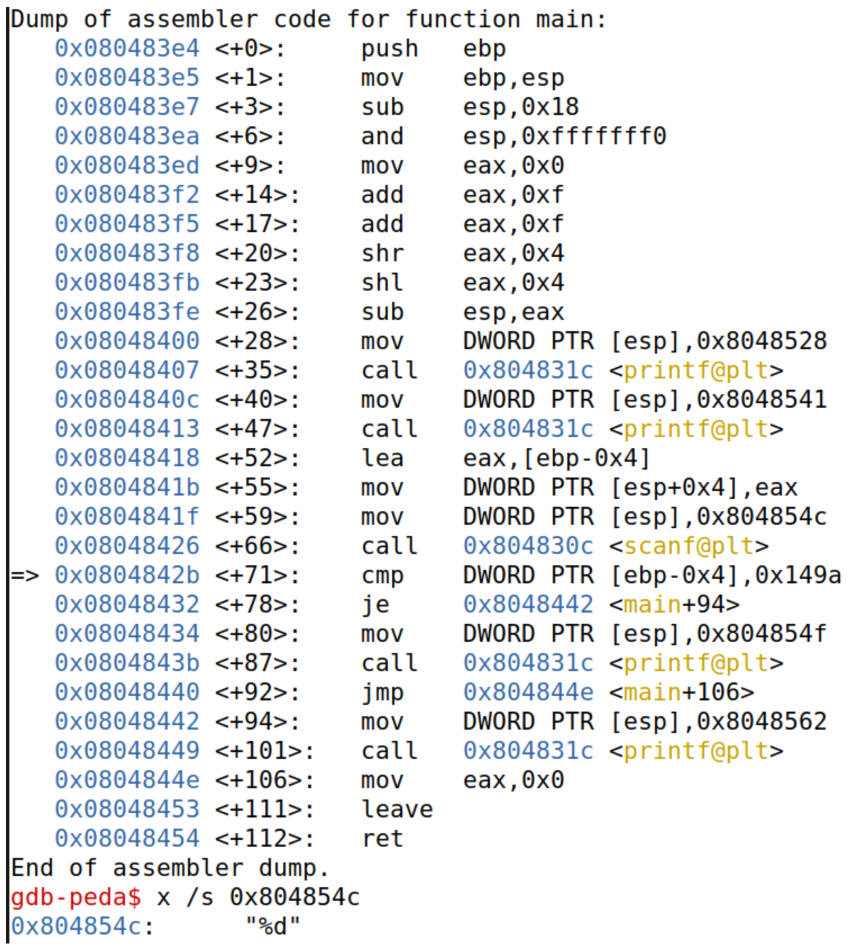
The values in the pointers referenced after that give us the “Password OK :)\n”, thus giving us the clue that the scanned input is compared to the previously present address value, which is “250382”. Using that password value, the program works.



```
gdb-peda$ r
Starting program: /home/seed/IOLI-crackme/crackme0x00
IOLI Crackme Level 0x00
Password: 250382
Password OK :)
[Inferior 1 (process 2771) exited normally]
Warning: not running
```

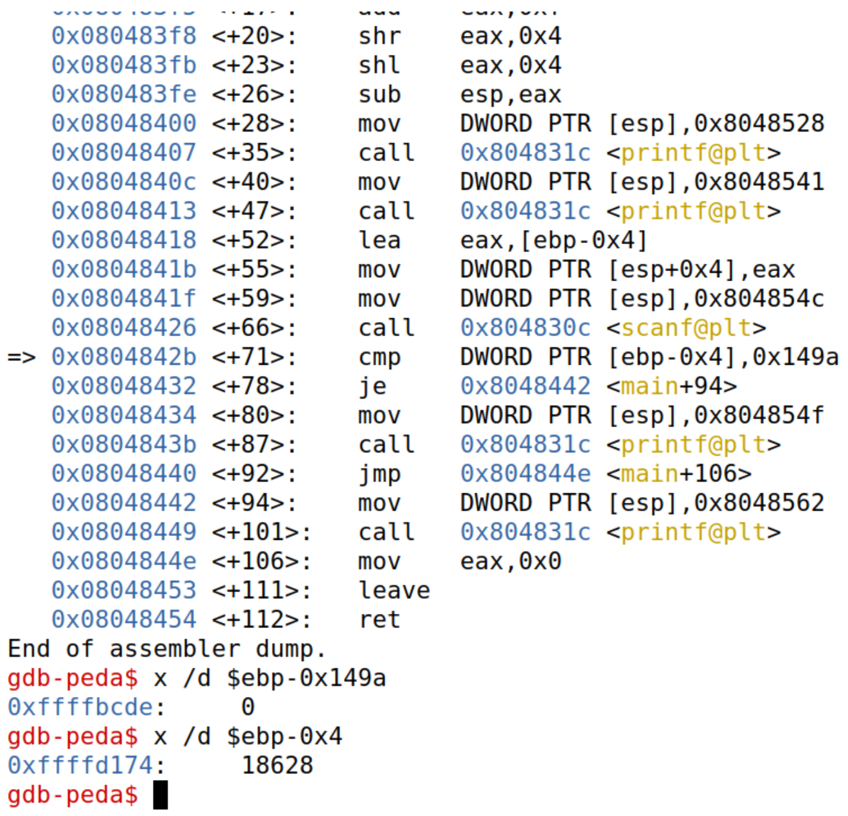
2. crackme0x01:

The suggested method in the example was to determine the presence of “%d” in the pointer “0x804854c”, which takes the input and then uses “CMP” to the given input in the pointer “ebp-0x4”, as we find out if the password entered was right or wrong.



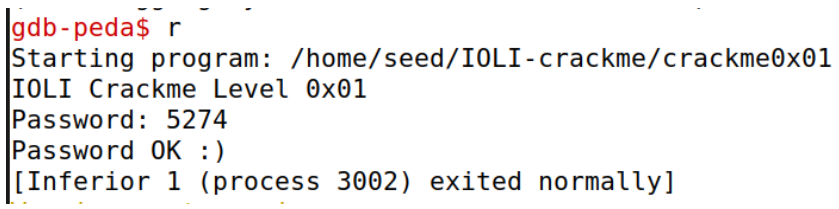
```
Dump of assembler code for function main:
   0x080483e4 <+0>:     push   ebp
   0x080483e5 <+1>:     mov    ebp,esp
   0x080483e7 <+3>:     sub    esp,0x18
   0x080483ea <+6>:     and    esp,0xfffffff0
   0x080483ed <+9>:     mov    eax,0x0
   0x080483f2 <+14>:    add    eax,0xf
   0x080483f5 <+17>:    add    eax,0xf
   0x080483f8 <+20>:    shr    eax,0x4
   0x080483fb <+23>:    shl    eax,0x4
   0x080483fe <+26>:    sub    esp,eax
   0x08048400 <+28>:    mov    DWORD PTR [esp],0x8048528
   0x08048407 <+35>:    call   0x804831c <printf@plt>
   0x0804840c <+40>:    mov    DWORD PTR [esp],0x8048541
   0x08048413 <+47>:    call   0x804831c <printf@plt>
   0x08048418 <+52>:    lea    eax,[ebp-0x4]
   0x0804841b <+55>:    mov    DWORD PTR [esp+0x4],eax
   0x0804841f <+59>:    mov    DWORD PTR [esp],0x804854c
   0x08048426 <+66>:    call   0x804830c <scanf@plt>
=> 0x0804842b <+71>:    cmp    DWORD PTR [ebp-0x4],0x149a
   0x08048432 <+78>:    je     0x8048442 <main+94>
   0x08048434 <+80>:    mov    DWORD PTR [esp],0x804854f
   0x0804843b <+87>:    call   0x804831c <printf@plt>
   0x08048440 <+92>:    jmp    0x804844e <main+106>
   0x08048442 <+94>:    mov    DWORD PTR [esp],0x8048562
   0x08048449 <+101>:   call   0x804831c <printf@plt>
   0x0804844e <+106>:   mov    eax,0x0
   0x08048453 <+111>:   leave
   0x08048454 <+112>:   ret
End of assembler dump.
gdb-peda$ x /s 0x804854c
0x804854c:      "%d"
```

However, the register “ebp-0x149a”, which is used to compare to the input, is empty.



```
   0x080483f8 <+20>:    shr    eax,0x4
   0x080483fb <+23>:    shl    eax,0x4
   0x080483fe <+26>:    sub    esp,eax
   0x08048400 <+28>:    mov    DWORD PTR [esp],0x8048528
   0x08048407 <+35>:    call   0x804831c <printf@plt>
   0x0804840c <+40>:    mov    DWORD PTR [esp],0x8048541
   0x08048413 <+47>:    call   0x804831c <printf@plt>
   0x08048418 <+52>:    lea    eax,[ebp-0x4]
   0x0804841b <+55>:    mov    DWORD PTR [esp+0x4],eax
   0x0804841f <+59>:    mov    DWORD PTR [esp],0x804854c
   0x08048426 <+66>:    call   0x804830c <scanf@plt>
=> 0x0804842b <+71>:    cmp    DWORD PTR [ebp-0x4],0x149a
   0x08048432 <+78>:    je     0x8048442 <main+94>
   0x08048434 <+80>:    mov    DWORD PTR [esp],0x804854f
   0x0804843b <+87>:    call   0x804831c <printf@plt>
   0x08048440 <+92>:    jmp    0x804844e <main+106>
   0x08048442 <+94>:    mov    DWORD PTR [esp],0x8048562
   0x08048449 <+101>:   call   0x804831c <printf@plt>
   0x0804844e <+106>:   mov    eax,0x0
   0x08048453 <+111>:   leave
   0x08048454 <+112>:   ret
End of assembler dump.
gdb-peda$ x /d $ebp-0x149a
0xffffbcde:     0
gdb-peda$ x /d $ebp-0x4
0xffffd174:     18628
gdb-peda$
```

Which gives us the conclusion that, it must either compare to the value in the register or the value of the number that's subtracted from the "ebp" value. The value of 0x1459a must correspond to a hexadecimal value that is compared to the normal input. The value, in decimal, turns out to be 5274. Using that value in the input, we finally crack the program.



```
gdb-peda$ r
Starting program: /home/seed/IOLI-crackme/crackme0x01
IOLI Crackme Level 0x01
Password: 5274
Password OK :)
[Inferior 1 (process 3002) exited normally]
```

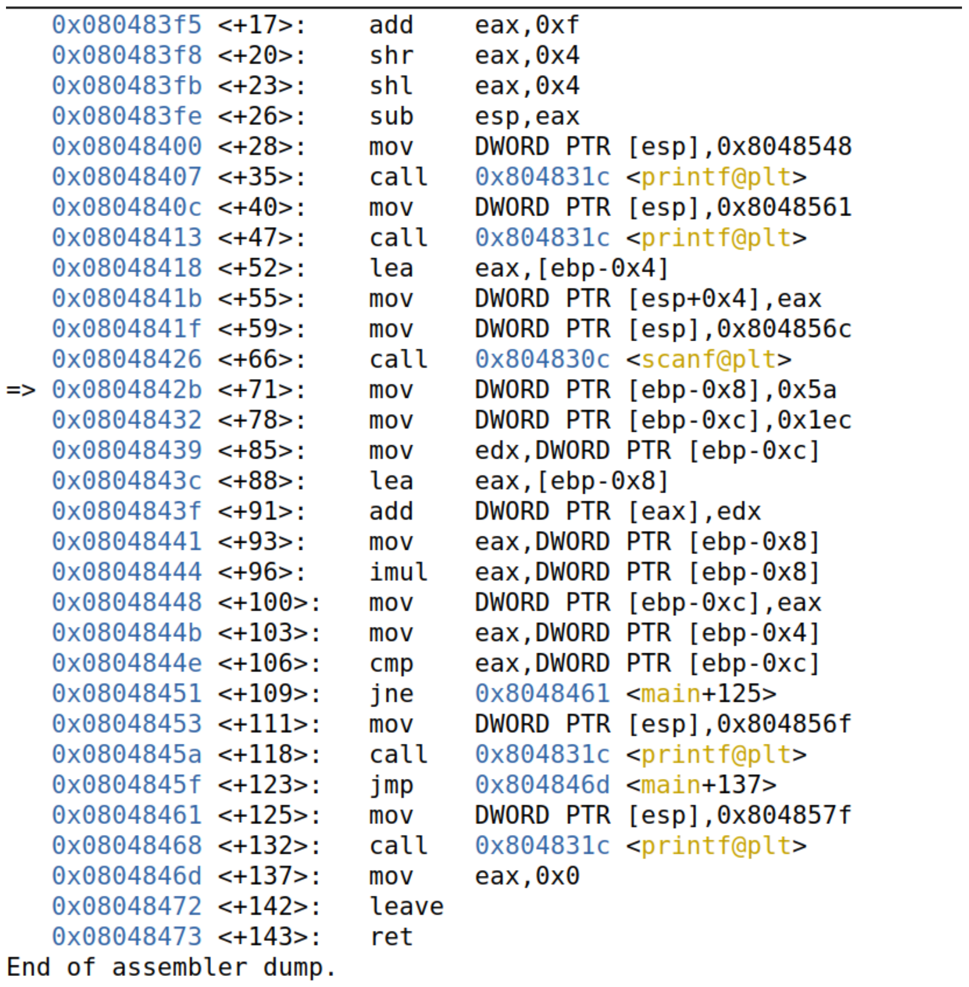
3. crackme0x02:

The disassembler of the program ends up giving the following code snippet:

x=x+y;

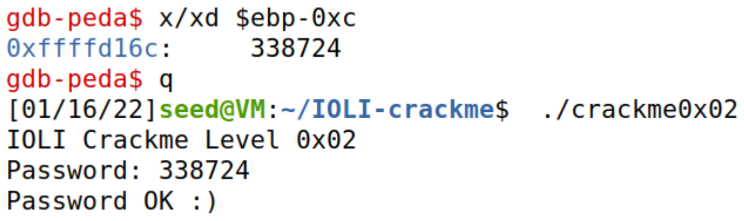
x=x*x;

Thus by converting the value "0x5a" which is assigned to the first register, we get the decimal value of x as 90. Doing the same with "0x1ec" will give us a value of y as 492. This gives us the final computed value of 338724, stored in the register 0xc from ebp, ie., ebp-0xc. That is our expected result.



```
   0x080483f5 <+17>:    add    eax,0xf
   0x080483f8 <+20>:    shr    eax,0x4
   0x080483fb <+23>:    shl    eax,0x4
   0x080483fe <+26>:    sub    esp,eax
   0x08048400 <+28>:    mov    DWORD PTR [esp],0x8048548
   0x08048407 <+35>:    call   0x804831c <printf@plt>
   0x0804840c <+40>:    mov    DWORD PTR [esp],0x8048561
   0x08048413 <+47>:    call   0x804831c <printf@plt>
   0x08048418 <+52>:    lea    eax,[ebp-0x4]
   0x0804841b <+55>:    mov    DWORD PTR [esp+0x4],eax
   0x0804841f <+59>:    mov    DWORD PTR [esp],0x804856c
   0x08048426 <+66>:    call   0x804830c <scanf@plt>
=> 0x0804842b <+71>:    mov    DWORD PTR [ebp-0x8],0x5a
   0x08048432 <+78>:    mov    DWORD PTR [ebp-0xc],0x1ec
   0x08048439 <+85>:    mov    edx,DWORD PTR [ebp-0xc]
   0x0804843c <+88>:    lea    eax,[ebp-0x8]
   0x0804843f <+91>:    add    DWORD PTR [eax],edx
   0x08048441 <+93>:    mov    eax,DWORD PTR [ebp-0x8]
   0x08048444 <+96>:    imul   eax,DWORD PTR [ebp-0x8]
   0x08048448 <+100>:   mov    DWORD PTR [ebp-0xc],eax
   0x0804844b <+103>:   mov    eax,DWORD PTR [ebp-0x4]
   0x0804844e <+106>:   cmp    eax,DWORD PTR [ebp-0xc]
   0x08048451 <+109>:   jne    0x8048461 <main+125>
   0x08048453 <+111>:   mov    DWORD PTR [esp],0x804856f
   0x0804845a <+118>:   call   0x804831c <printf@plt>
   0x0804845f <+123>:   jmp    0x804846d <main+137>
   0x08048461 <+125>:   mov    DWORD PTR [esp],0x804857f
   0x08048468 <+132>:   call   0x804831c <printf@plt>
   0x0804846d <+137>:   mov    eax,0x0
   0x08048472 <+142>:   leave
   0x08048473 <+143>:   ret
End of assembler dump.
```

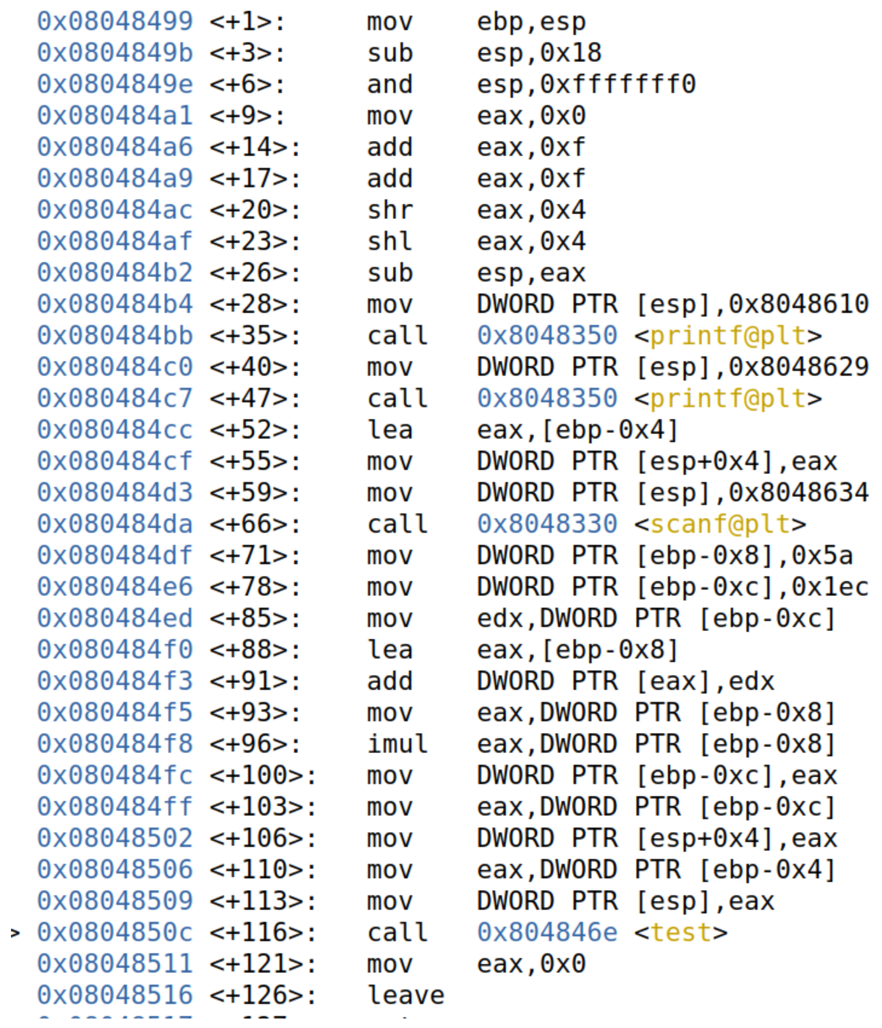
The value is known by iterating through the code after the breakpoints, and thus reaching the comparing value at the “CMP” command.



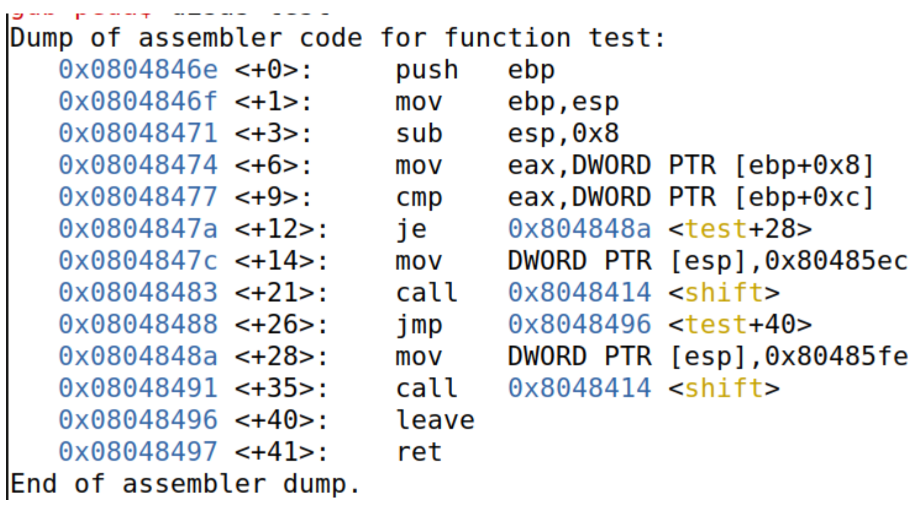
```
gdb-peda$ x/xd $ebp-0xc
0xffffd16c:     338724
gdb-peda$ q
[01/16/22]seed@VM:~/IOLI-crackme$  ./crackme0x02
IOLI Crackme Level 0x02
Password: 338724
Password OK :)
```

4. crackme0x03:

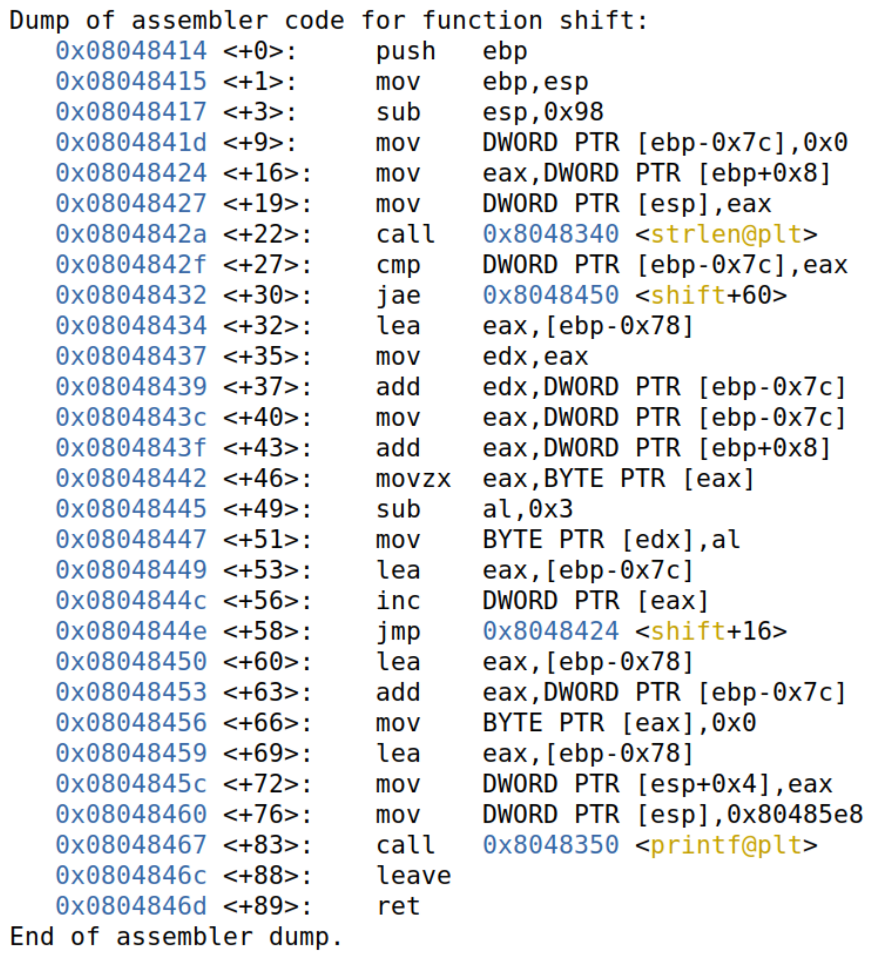
The code happens to be the same, except for the ending snippet, which redirects to the function test. The function test redirects to another function shift, and both of them are explored. These happen to be just the code for displaying the string of “Invalid password” or “Password OK.” Thus the same way as the previous problem, the value iterated until the <+100> line, stored in <ebp-0xc>, is 338724 again. Finally, it worked to this program.



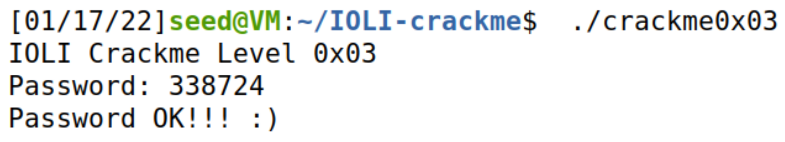
```
   0x08048499 <+1>:     mov    ebp,esp
   0x0804849b <+3>:     sub    esp,0x18
   0x0804849e <+6>:     and    esp,0xfffffff0
   0x080484a1 <+9>:     mov    eax,0x0
   0x080484a6 <+14>:    add    eax,0xf
   0x080484a9 <+17>:    add    eax,0xf
   0x080484ac <+20>:    shr    eax,0x4
   0x080484af <+23>:    shl    eax,0x4
   0x080484b2 <+26>:    sub    esp,eax
   0x080484b4 <+28>:    mov    DWORD PTR [esp],0x8048610
   0x080484bb <+35>:    call   0x8048350 <printf@plt>
   0x080484c0 <+40>:    mov    DWORD PTR [esp],0x8048629
   0x080484c7 <+47>:    call   0x8048350 <printf@plt>
   0x080484cc <+52>:    lea    eax,[ebp-0x4]
   0x080484cf <+55>:    mov    DWORD PTR [esp+0x4],eax
   0x080484d3 <+59>:    mov    DWORD PTR [esp],0x8048634
   0x080484da <+66>:    call   0x8048330 <scanf@plt>
   0x080484df <+71>:    mov    DWORD PTR [ebp-0x8],0x5a
   0x080484e6 <+78>:    mov    DWORD PTR [ebp-0xc],0x1ec
   0x080484ed <+85>:    mov    edx,DWORD PTR [ebp-0xc]
   0x080484f0 <+88>:    lea    eax,[ebp-0x8]
   0x080484f3 <+91>:    add    DWORD PTR [eax],edx
   0x080484f5 <+93>:    mov    eax,DWORD PTR [ebp-0x8]
   0x080484f8 <+96>:    imul   eax,DWORD PTR [ebp-0x8]
   0x080484fc <+100>:   mov    DWORD PTR [ebp-0xc],eax
   0x080484ff <+103>:   mov    eax,DWORD PTR [ebp-0xc]
   0x08048502 <+106>:   mov    DWORD PTR [esp+0x4],eax
   0x08048506 <+110>:   mov    eax,DWORD PTR [ebp-0x4]
   0x08048509 <+113>:   mov    DWORD PTR [esp],eax
 > 0x0804850c <+116>:   call   0x804846e <test>
   0x08048511 <+121>:   mov    eax,0x0
   0x08048516 <+126>:   leave
```



```
Dump of assembler code for function test:
   0x0804846e <+0>:     push   ebp
   0x0804846f <+1>:     mov    ebp,esp
   0x08048471 <+3>:     sub    esp,0x8
   0x08048474 <+6>:     mov    eax,DWORD PTR [ebp+0x8]
   0x08048477 <+9>:     cmp    eax,DWORD PTR [ebp+0xc]
   0x0804847a <+12>:    je     0x804848a <test+28>
   0x0804847c <+14>:    mov    DWORD PTR [esp],0x80485ec
   0x08048483 <+21>:    call   0x8048414 <shift>
   0x08048488 <+26>:    jmp    0x8048496 <test+40>
   0x0804848a <+28>:    mov    DWORD PTR [esp],0x80485fe
   0x08048491 <+35>:    call   0x8048414 <shift>
   0x08048496 <+40>:    leave
   0x08048497 <+41>:    ret
End of assembler dump.
```



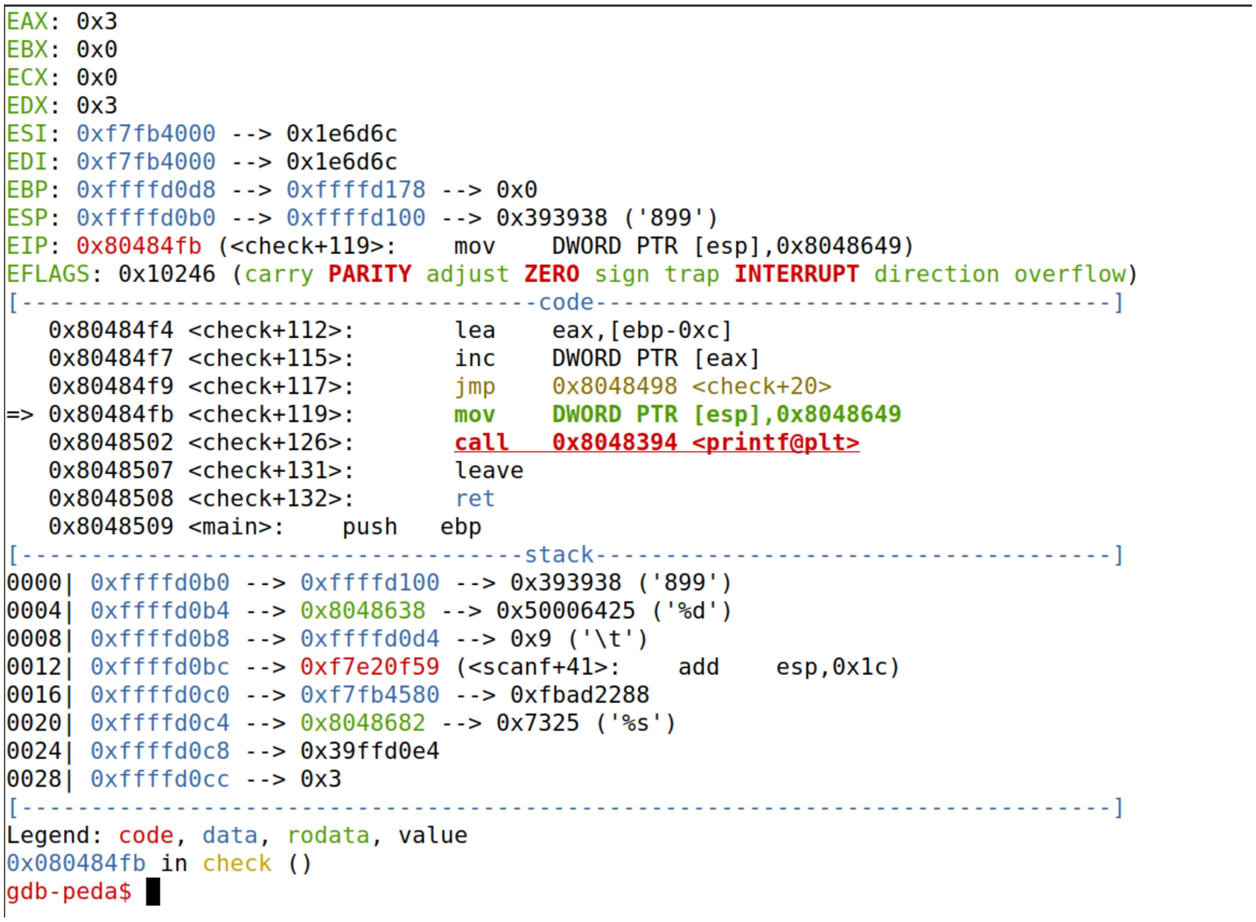
```
Dump of assembler code for function shift:
   0x08048414 <+0>:     push   ebp
   0x08048415 <+1>:     mov    ebp,esp
   0x08048417 <+3>:     sub    esp,0x98
   0x0804841d <+9>:     mov    DWORD PTR [ebp-0x7c],0x0
   0x08048424 <+16>:    mov    eax,DWORD PTR [ebp+0x8]
   0x08048427 <+19>:    mov    DWORD PTR [esp],eax
   0x0804842a <+22>:    call   0x8048340 <strlen@plt>
   0x0804842f <+27>:    cmp    DWORD PTR [ebp-0x7c],eax
   0x08048432 <+30>:    jae    0x8048450 <shift+60>
   0x08048434 <+32>:    lea    eax,[ebp-0x78]
   0x08048437 <+35>:    mov    edx,eax
   0x08048439 <+37>:    add    edx,DWORD PTR [ebp-0x7c]
   0x0804843c <+40>:    mov    eax,DWORD PTR [ebp-0x7c]
   0x0804843f <+43>:    add    eax,DWORD PTR [ebp+0x8]
   0x08048442 <+46>:    movzx  eax,BYTE PTR [eax]
   0x08048445 <+49>:    sub    al,0x3
   0x08048447 <+51>:    mov    BYTE PTR [edx],al
   0x08048449 <+53>:    lea    eax,[ebp-0x7c]
   0x0804844c <+56>:    inc    DWORD PTR [eax]
   0x0804844e <+58>:    jmp    0x8048424 <shift+16>
   0x08048450 <+60>:    lea    eax,[ebp-0x78]
   0x08048453 <+63>:    add    eax,DWORD PTR [ebp-0x7c]
   0x08048456 <+66>:    mov    BYTE PTR [eax],0x0
   0x08048459 <+69>:    lea    eax,[ebp-0x78]
   0x0804845c <+72>:    mov    DWORD PTR [esp+0x4],eax
   0x08048460 <+76>:    mov    DWORD PTR [esp],0x80485e8
   0x08048467 <+83>:    call   0x8048350 <printf@plt>
   0x0804846c <+88>:    leave
   0x0804846d <+89>:    ret
End of assembler dump.
```



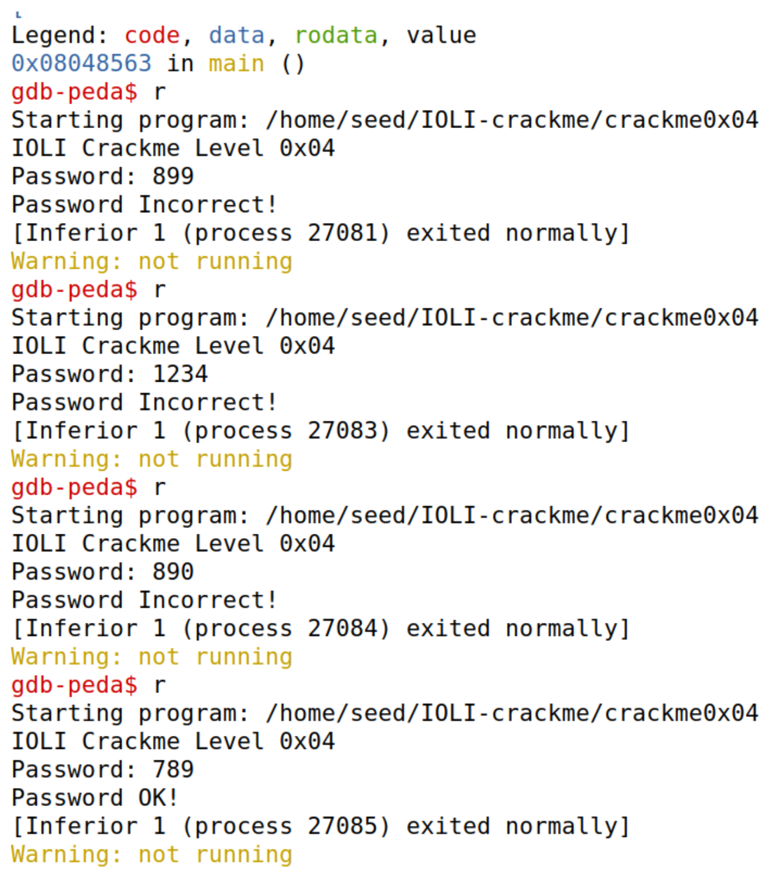
```
[01/17/22]seed@VM:~/IOLI-crackme$  ./crackme0x03
IOLI Crackme Level 0x03
Password: 338724
Password OK!!! :)
```

5. crackme0x04:

The program is made to run every step by step, and the resultant is a loop that converts the values of the string input (%s) and makes it to a decimal input per character converted to the integer (%d), the sum of these values is compared to “0xf” or the decimal value 15.



```
EAX: 0x3
EBX: 0x0
ECX: 0x0
EDX: 0x3
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0xffffd0d8 --> 0xffffd178 --> 0x0
ESP: 0xffffd0b0 --> 0xffffd100 --> 0x393938 ('899')
EIP: 0x80484fb (<check+119>:    mov    DWORD PTR [esp],0x8048649)
EFLAGS: 0x10246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x80484f4 <check+112>:       lea    eax,[ebp-0xc]
   0x80484f7 <check+115>:       inc    DWORD PTR [eax]
   0x80484f9 <check+117>:       jmp    0x8048498 <check+20>
=> 0x80484fb <check+119>:       mov    DWORD PTR [esp],0x8048649
   0x8048502 <check+126>:       call   0x8048394 <printf@plt>
   0x8048507 <check+131>:       leave
   0x8048508 <check+132>:       ret
   0x8048509 <main>:    push   ebp
[------------------------------------stack-------------------------------------]
0000| 0xffffd0b0 --> 0xffffd100 --> 0x393938 ('899')
0004| 0xffffd0b4 --> 0x8048638 --> 0x50006425 ('%d')
0008| 0xffffd0b8 --> 0xffffd0d4 --> 0x9 ('\t')
0012| 0xffffd0bc --> 0xf7e20f59 (<scanf+41>:    add    esp,0x1c)
0016| 0xffffd0c0 --> 0xf7fb4580 --> 0xfbad2288
0020| 0xffffd0c4 --> 0x8048682 --> 0x7325 ('%s')
0024| 0xffffd0c8 --> 0x39ffd0e4
0028| 0xffffd0cc --> 0x3
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x080484fb in check ()
gdb-peda$ 
```



```
Legend: code, data, rodata, value
0x08048563 in main ()
gdb-peda$ r
Starting program: /home/seed/IOLI-crackme/crackme0x04
IOLI Crackme Level 0x04
Password: 899
Password Incorrect!
[Inferior 1 (process 27081) exited normally]
Warning: not running
gdb-peda$ r
Starting program: /home/seed/IOLI-crackme/crackme0x04
IOLI Crackme Level 0x04
Password: 1234
Password Incorrect!
[Inferior 1 (process 27083) exited normally]
Warning: not running
gdb-peda$ r
Starting program: /home/seed/IOLI-crackme/crackme0x04
IOLI Crackme Level 0x04
Password: 890
Password Incorrect!
[Inferior 1 (process 27084) exited normally]
Warning: not running
gdb-peda$ r
Starting program: /home/seed/IOLI-crackme/crackme0x04
IOLI Crackme Level 0x04
Password: 789
Password OK!
[Inferior 1 (process 27085) exited normally]
Warning: not running
```

Thus the function compares the digits of the given password, and is accepted if the sum of these digits is 15.

6. crackme0x05:

The program here compares the same way as the previous problem, but this time to equal 16. However, the number is then under the function "parell", this compares if the given number is even or odd, which is thus described by the code snippet "if(!(n&1))". The program was observed to be ignoring the alphabets after the code, but has ignored if the alphabets have been in the first place.

```
EAX: 0x1
EBX: 0xf7fb4000 --> 0x1e6d6c
ECX: 0x0
EDX: 0x8fef9800
ESI: 0xffffcfec --> 0xfbad8001
EDI: 0x8048668 --> 0x50006425 ('%d')
EBP: 0xffffd0cb --> 0x34 ('4')
ESP: 0xffffcfe0 --> 0x0
EIP: 0xf7e20fe9 (<sscanf+137>:  xor    edx,DWORD PTR gs:0x14)
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0xf7e20fda <sscanf+122>:     call   0xf7e226e0
   0xf7e20fdf <sscanf+127>:     add    esp,0x20
   0xf7e20fe2 <sscanf+130>:     mov    edx,DWORD PTR [esp+0xac]
=> 0xf7e20fe9 <sscanf+137>:     xor    edx,DWORD PTR gs:0x14
   0xf7e20ff0 <sscanf+144>:     jne    0xf7e20ffd <sscanf+157>
   0xf7e20ff2 <sscanf+146>:     add    esp,0xbc
   0xf7e20ff8 <sscanf+152>:     pop    ebx
   0xf7e20ff9 <sscanf+153>:     pop    esi
[------------------------------------stack-------------------------------------]
0000| 0xffffcfe0 --> 0x0
0004| 0xffffcfe4 --> 0x0
0008| 0xffffcfe8 --> 0x0
0012| 0xffffcfec --> 0xfbad8001
0016| 0xffffcff0 --> 0xffffd0cc --> 0x0
0020| 0xffffcff4 --> 0xffffd0cc --> 0x0
0024| 0xffffcff8 --> 0xffffd0cb --> 0x34 ('4')
0028| 0xffffcffc --> 0xffffd0cb --> 0x34 ('4')
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0xf7e20fe9 in sscanf () from /lib32/libc.so.6


gdb-peda$ r
Starting program: /home/seed/IOLI-crackme/crackme0x05
IOLI Crackme Level 0x05
Password: 976
Password OK!
[Inferior 1 (process 27420) exited normally]
Warning: not running
gdb-peda$ r
Starting program: /home/seed/IOLI-crackme/crackme0x05
IOLI Crackme Level 0x05
Password: 9934
Password Incorrect!
[Inferior 1 (process 27421) exited normally]
Warning: not running
gdb-peda$ r
Starting program: /home/seed/IOLI-crackme/crackme0x05
IOLI Crackme Level 0x05
Password: 23452
Password OK!
[Inferior 1 (process 27425) exited normally]
Warning: not running
gdb-peda$ r
Starting program: /home/seed/IOLI-crackme/crackme0x05
IOLI Crackme Level 0x05
Password: 4561cd
Password Incorrect!
[Inferior 1 (process 27675) exited normally]
Warning: not running
gdb-peda$ r
Starting program: /home/seed/IOLI-crackme/crackme0x05
IOLI Crackme Level 0x05
Password: 1456r
Password OK!
```