CM0626 – Software Security - Reverse Engineering with Ghidra

In this exercise, we will use Ghidra to reverse engineer a well-known crack-me suite called IOLI. Download it from Moodle. Please have a copy of Intel Assembler code table, since we will go throughout disassembled source code: it is also available on moodle.

The aim of this set of crack-me is to find a secret (password) attacking the executable. Difficulty in defeating protections schemes increases from 0 to 9.

However these are 32-bit binary executables, so if you run it in a 64-bit linux distribution you should install some extra libraries (see the last 3 commands below).

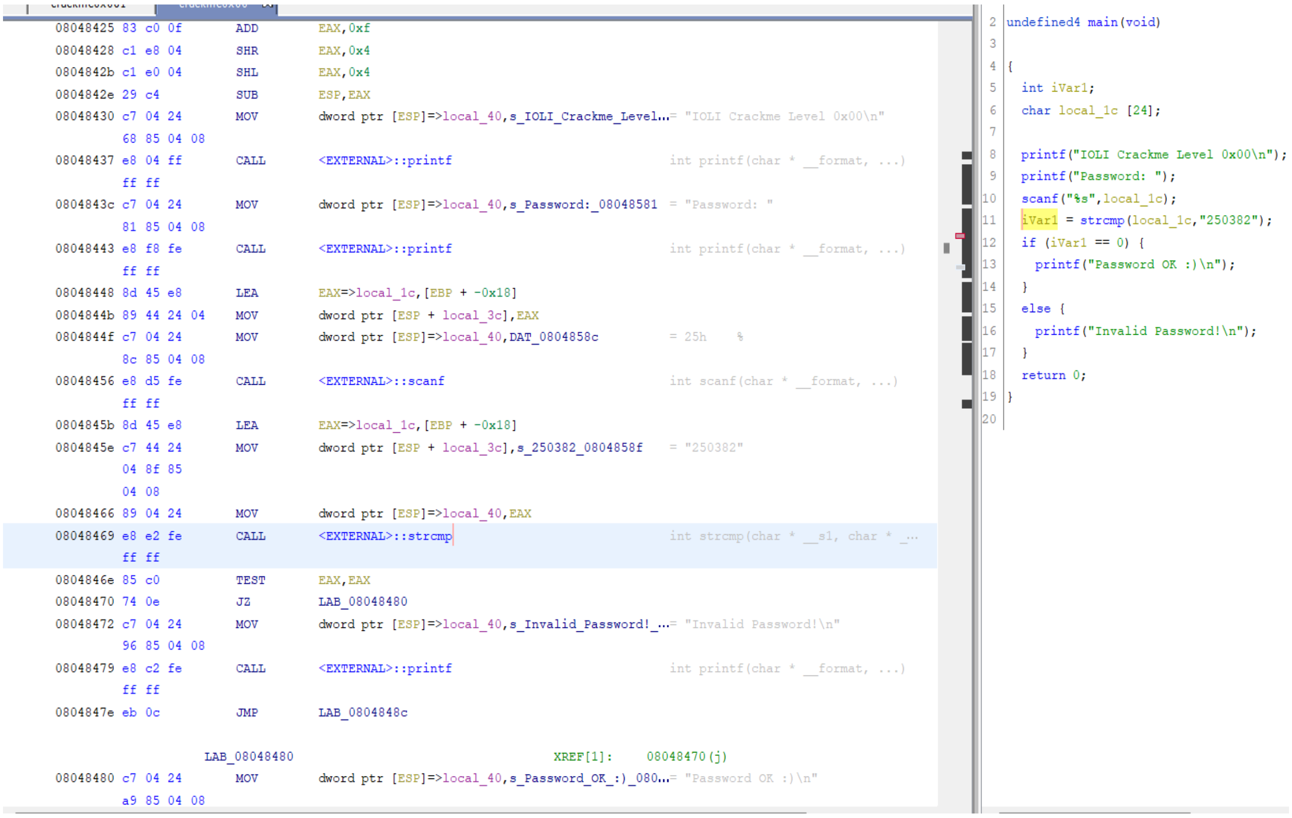
```
root@kali:~/reveng# dpkg --add-architecture i386
root@kali:~/reveng# apt-get update
root@kali:~/reveng# apt-get install lib32z1 lib32ncurses5 lib32stdc++6
```

These binary files can be run on Linux on x86 (32-bit) architecture, but they can be analyzed with Ghidra on any platform.

Crackme Level 0x00

Let’s open it with Ghidra and select the main function

Function main()



```
08048425 83 c0 0f       ADD     EAX,0xf
08048428 c1 e8 04       SHR     EAX,0x4
0804842b c1 e0 04       SHL     EAX,0x4
0804842e 29 c4          SUB     ESP,EAX
08048430 c7 04 24       MOV     dword ptr [ESP]=>local_40,s_IOLI_Crackme_Level…= "IOLI Crackme Level 0x00\n"
         68 85 04 08
08048437 e8 04 ff       CALL    <EXTERNAL>::printf                          int printf(char * __format, ...)
         ff ff
0804843c c7 04 24       MOV     dword ptr [ESP]=>local_40,s_Password:_08048581 = "Password: "
         81 85 04 08
08048443 e8 f8 fe       CALL    <EXTERNAL>::printf                          int printf(char * __format, ...)
         ff ff
08048448 8d 45 e8       LEA     EAX=>local_1c,[EBP + -0x18]
0804844b 89 44 24 04    MOV     dword ptr [ESP + local_3c],EAX
0804844f c7 04 24       MOV     dword ptr [ESP]=>local_40,DAT_0804858c       = 25h    %
         8c 85 04 08
08048456 e8 d5 fe       CALL    <EXTERNAL>::scanf                           int scanf(char * __format, ...)
         ff ff
0804845b 8d 45 e8       LEA     EAX=>local_1c,[EBP + -0x18]
0804845e c7 44 24       MOV     dword ptr [ESP + local_3c],s_250382_0804858f   = "250382"
         04 8f 85
         04 08
08048466 89 04 24       MOV     dword ptr [ESP]=>local_40,EAX
08048469 e8 e2 fe       CALL    <EXTERNAL>::strcmp                          int strcmp(char * __s1, char * _…
         ff ff
0804846e 85 c0          TEST    EAX,EAX
08048470 74 0e          JZ      LAB_08048480
08048472 c7 04 24       MOV     dword ptr [ESP]=>local_40,s_Invalid_Password!_…= "Invalid Password!\n"
         96 85 04 08
08048479 e8 c2 fe       CALL    <EXTERNAL>::printf                          int printf(char * __format, ...)
         ff ff
0804847e eb 0c          JMP     LAB_0804848c

                  LAB_08048480                             XREF[1]:     08048470(j)
08048480 c7 04 24       MOV     dword ptr [ESP]=>local_40,s_Password_OK_:)_080…= "Password OK :)\n"
         a9 85 04 08
```

```
undefined4 main(void)

{
  int iVar1;
  char local_1c [24];

  printf("IOLI Crackme Level 0x00\n");
  printf("Password: ");
  scanf("%s",local_1c);
  iVar1 = strcmp(local_1c,"250382");
  if (iVar1 == 0) {
    printf("Password OK :)\n");
  }
  else {
    printf("Invalid Password!\n");
  }
  return 0;
}
```

The pseudocode is:

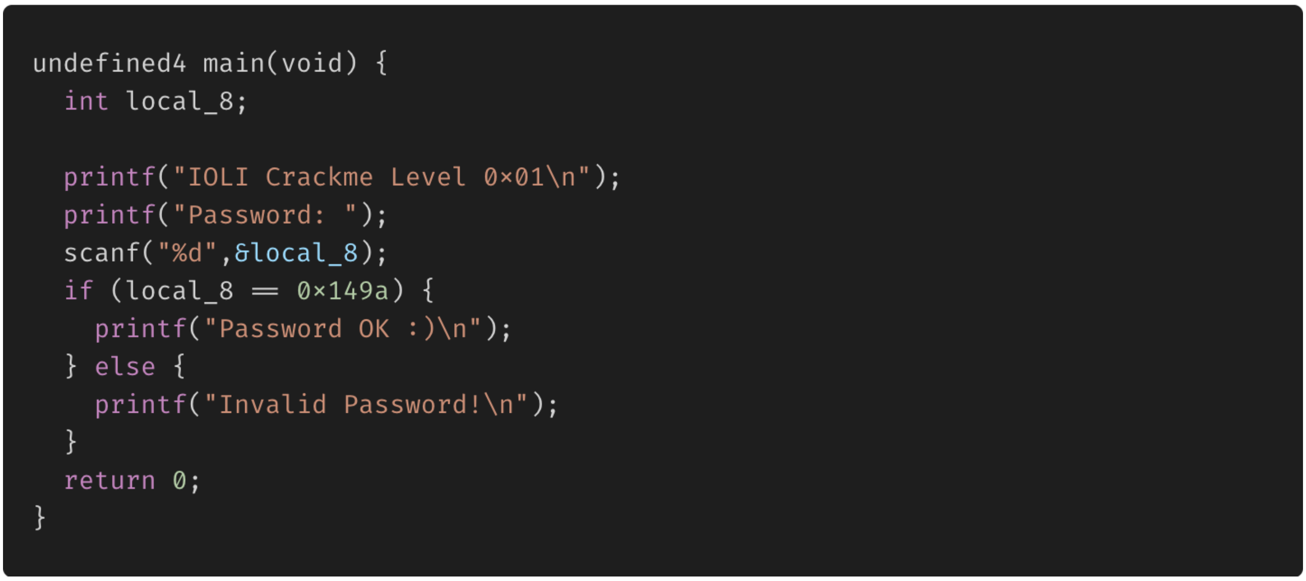
1. 0x08048456: Instruction scanf to take in input the password
2. 0x08048469: Instruction strcmp to compare the input password with the visible password
3. 0x0804846e: If the return value of the strcmp function is 0 jump to the point that the program prints “Password ok”
4. 0x08048479: Otherwise set invalid password as parameter of printf function
5. 0x08048492: Terminate the program

Logical flow:

1. Enter the password
2. Compare it with the visible string 250382
3. If they are equal it prints ok
4. Otherwise it prints incorrect password
5. So we can observe that in this login schema we can see immediately the password and it is very very simple to hacker it. In fact we can patch the if instruction and change the if condition to make program prints the correct password message also if the input password is wrong.

Level 0x01

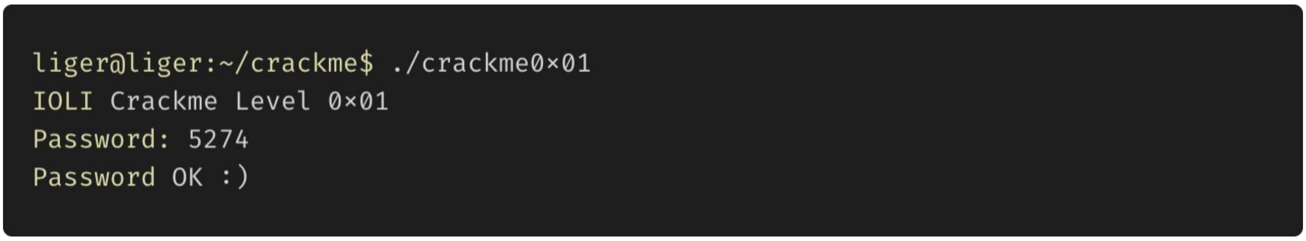
We start right away by importing the challenge into *Ghidra* and decompiling it. Once the analysis is done, we start looking for clues into the main function:



```
undefined4 main(void) {
  int local_8;

  printf("IOLI Crackme Level 0x01\n");
  printf("Password: ");
  scanf("%d",&local_8);
  if (local_8 == 0x149a) {
    printf("Password OK :)\n");
  } else {
    printf("Invalid Password!\n");
  }
  return 0;
}
```

It seems pretty obvious that in order to get printed *"Password Ok"* in the terminal we need to enter the value 0x149a, which in decimal representation is equals to 5274 (to recall that we can get the conversion right into Ghidra by just right clicking the value and select the appropriate representation).

Finally, we test out our guess by launching the challenge and inserting 5274 as password:

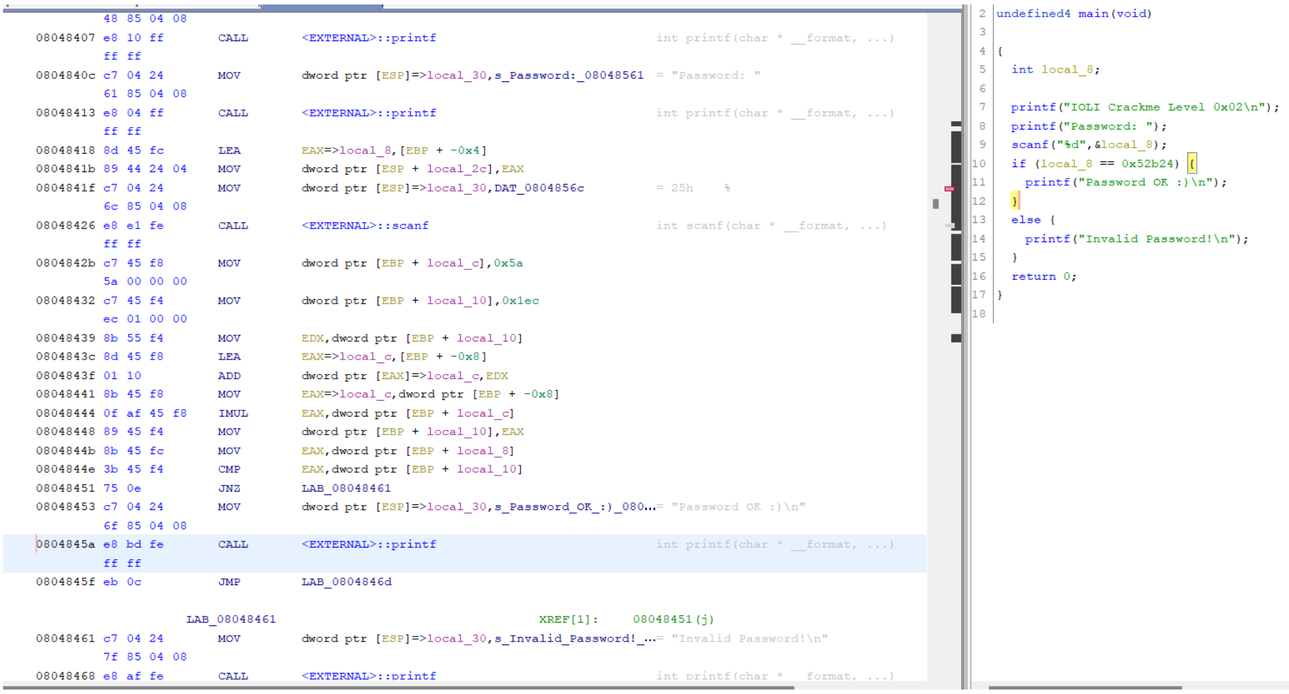


```
liger@liger:~/crackme$ ./crackme0×01
IOLI Crackme Level 0×01
Password: 5274
Password OK :)
```

And it worked!

Crackme Level 0x02

Function main()



```
         48 85 04 08
08048407 e8 10 ff      CALL     <EXTERNAL>::printf                         int printf(char * __format, ...)
         ff ff
0804840c c7 04 24      MOV      dword ptr [ESP]=>local_30,s_Password:_08048561  = "Password: "
         61 85 04 08
08048413 e8 04 ff      CALL     <EXTERNAL>::printf                         int printf(char * __format, ...)
         ff ff
08048418 8d 45 fc      LEA      EAX=>local_8,[EBP + -0x4]
0804841b 89 44 24 04   MOV      dword ptr [ESP + local_2c],EAX
0804841f c7 04 24      MOV      dword ptr [ESP]=>local_30,DAT_0804856c      = 25h    %
         6c 85 04 08
08048426 e8 e1 fe      CALL     <EXTERNAL>::scanf                          int scanf(char * __format, ...)
         ff ff
0804842b c7 45 f8      MOV      dword ptr [EBP + local_c],0x5a
         5a 00 00 00
08048432 c7 45 f4      MOV      dword ptr [EBP + local_10],0x1ec
         ec 01 00 00
08048439 8b 55 f4      MOV      EDX,dword ptr [EBP + local_10]
0804843c 8d 45 f8      LEA      EAX=>local_c,[EBP + -0x8]
0804843f 01 10         ADD      dword ptr [EAX]=>local_c,EDX
08048441 8b 45 f8      MOV      EAX=>local_c,dword ptr [EBP + -0x8]
08048444 0f af 45 f8   IMUL     EAX,dword ptr [EBP + local_c]
08048448 89 45 f4      MOV      dword ptr [EBP + local_10],EAX
0804844b 8b 45 fc      MOV      EAX,dword ptr [EBP + local_8]
0804844e 3b 45 f4      CMP      EAX,dword ptr [EBP + local_10]
08048451 75 0e         JNZ      LAB_08048461
08048453 c7 04 24      MOV      dword ptr [ESP]=>local_30,s_Password_OK_:)_080…= "Password OK :)\n"
         6f 85 04 08
0804845a e8 bd fe      CALL     <EXTERNAL>::printf                         int printf(char * __format, ...)
         ff ff
0804845f eb 0c         JMP      LAB_0804846d

                 LAB_08048461                                  XREF[1]:     08048451(j)
08048461 c7 04 24      MOV      dword ptr [ESP]=>local_30,s_Invalid_Password!_…= "Invalid Password!\n"
         7f 85 04 08
08048468 e8 af fe      CALL     <EXTERNAL>::printf                         int printf(char * __format, ...)
```

```
undefined4 main(void)

{
  int local_8;

  printf("IOLI Crackme Level 0x02\n");
  printf("Password: ");
  scanf("%d",&local_8);
  if (local_8 == 0x52b24) {
    printf("Password OK :)\n");
  }
  else {
    printf("Invalid Password!\n");
  }
  return 0;
}
```

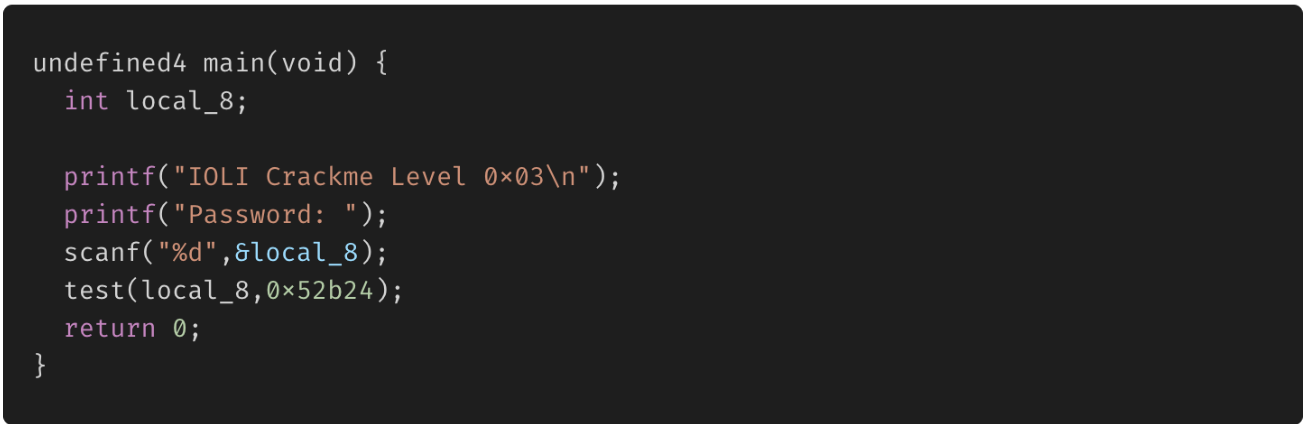
The pseudocode is the same of crackme 0 and the unique difference is that the password is not immediately visible but calculated with this pseudocode:

1. Var1 = 0x5a = 90
2. Var2 = 0x1ec = 492
3. Var3 = Var1 + Var2 = 582
4. Var4 = Var3 * Var3 = 582 * 582 = 338724

So in the decompiled code we must click the left button of the mouse to see directly its decimal value. In fact the converted value of 0x52b24 is 338724

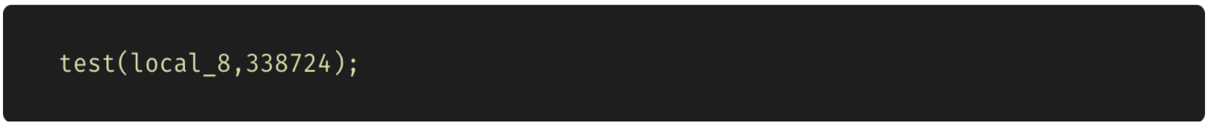
Crackme Level 0x03

We start with the same procedure as before, meaning that we import the challenge into *Ghidra* and start analysing it.
Then, we jump right away into the main function:



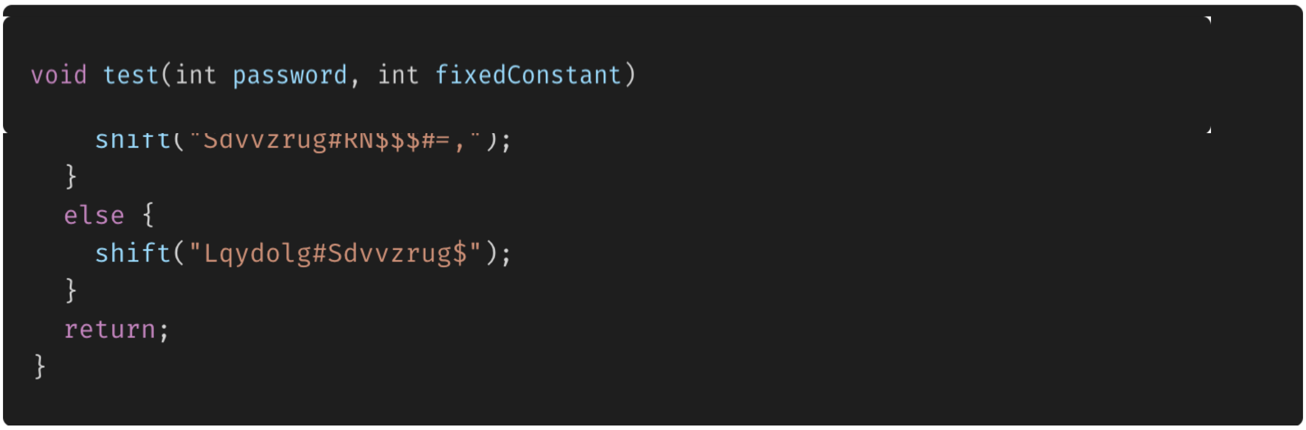
```
undefined4 main(void) {
  int local_8;

  printf("IOLI Crackme Level 0x03\n");
  printf("Password: ");
  scanf("%d",&local_8);
  test(local_8,0x52b24);
  return 0;
}
```

This time it's pretty clear that the logic that checks the password is not in here, but rather inside the test function, but before moving on and looking at it, we are going to convert the hard-coded value (0x52b24) into decimal representation:



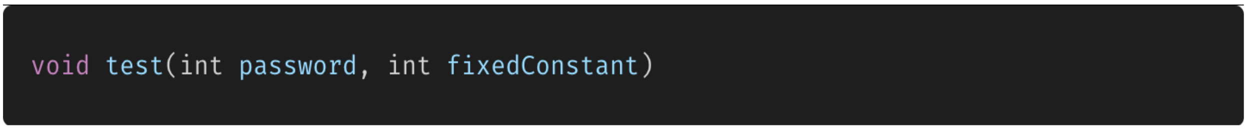
```
test(local_8,338724);
```

Now, here's the test function:



```
void test(int password, int fixedConstant)

    shift("Sdvvzrug#RN$$$#=,");
  }
  else {
    shift("Lqydolg#Sdvvzrug$");
  }
  return;
}
```

Remember that param_1 is the inserted password and param_2 is the hard-coded value. Therefore, the test function signature can be rewritten as:



```
void test(int password, int fixedConstant)
```

It seems that it simply checks the inserted password with the previously encountered *fixed value* and it calls a new function, shift, passing two different hard-coded string based if the two test's function parameters are equals or not.

So, for now let's jump into the shift function, but we will come back later regarding the meaning of these two different strings.

```
void shift(char *param_1) {
  size_t sVar1;
  uint local_80;
  char local_7c [120];

  local_80 = 0;
  while( true ) {
    sVar1 = strlen(param_1);
    if (sVar1 <= local_80) break;
    local_7c[local_80] = param_1[local_80] + -3;
    local_80 = local_80 + 1;
  }
  local_7c[local_80] = '\0';
  printf("%s\n",local_7c);
  return;
}
```

Again, the shift signature can be seen as:

```
void shift(char *fixedString)
```

In this final function we can observe that the passed string is analysed character by character and for each one of them subtracts -3 (in *ASCII* representation). Then, the resulting value is saved into the local_7c string variable, which at the end will be printed.
Basically, we are emulating a ***shift cipher*** with key=-3 in order to decrypt the given cipher text.

So, we just need to decrypt those two strings passed to the shift function (inside the test function) in order to understand which is the right string that corresponds to "Password Ok". To do that we can use a very simple python script like this one (which actually emulate the shift function):

```
def shift(fixedString):
    output = ""
    for letter in fixedString:
        output += chr(ord(letter) - 3)
    print(output)

shift("Sdvvzrug#RN$$$#=,")
shift("Lqydolg#Sdvvzrug$")
```

And once we run it, we will get:

```
Password OK!!! :)
Invalid Password!
```

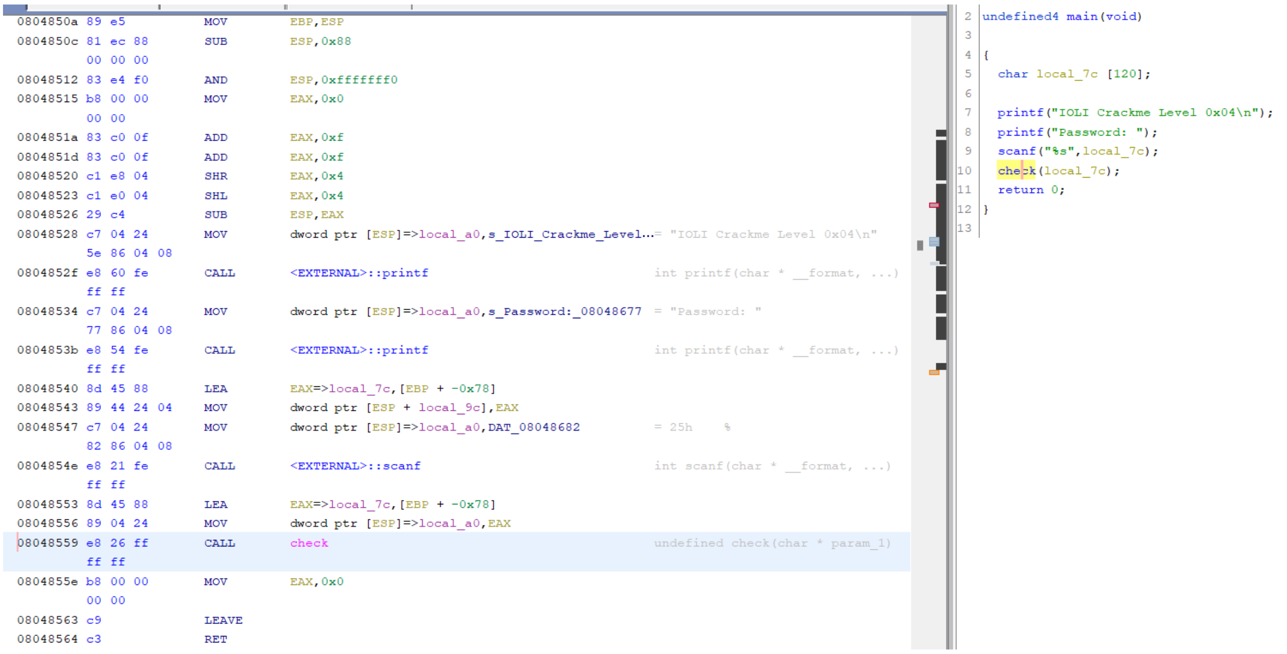
Meaning that the inserted password needs to be equals to 338724 if we want to get printed "Password Ok". So, we test it all out by running the actual crackme0x03:

```
liger@liger:~/crackme$ ./crackme0×03
IOLI Crackme Level 0×03
Password: 338724
Password OK!!! :)
```

And as predicted, it worked!

Crackme Level 0x04

Function main()



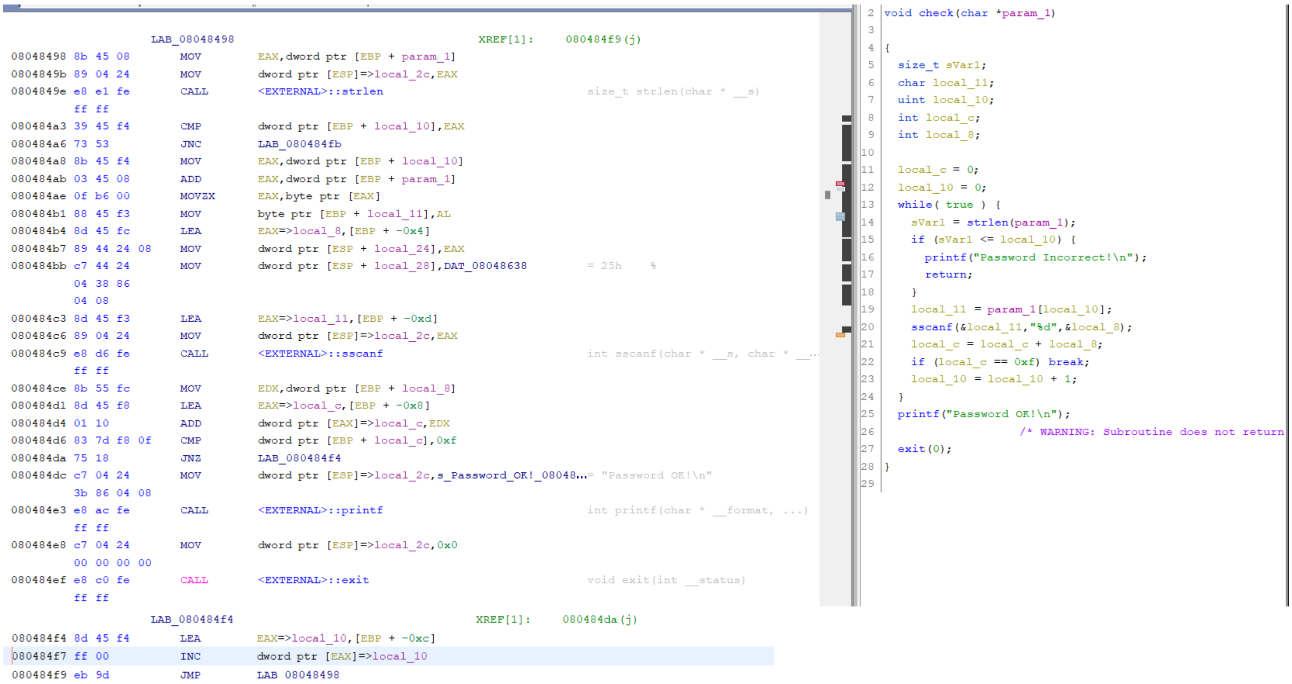
```
0804850a 89 e5          MOV     EBP,ESP
0804850c 81 ec 88       SUB     ESP,0x88
         00 00 00
08048512 83 e4 f0       AND     ESP,0xfffffff0
08048515 b8 00 00       MOV     EAX,0x0
         00 00
0804851a 83 c0 0f       ADD     EAX,0xf
0804851d 83 c0 0f       ADD     EAX,0xf
08048520 c1 e8 04       SHR     EAX,0x4
08048523 c1 e0 04       SHL     EAX,0x4
08048526 29 c4          SUB     ESP,EAX
08048528 c7 04 24       MOV     dword ptr [ESP]=>local_a0,s_IOLI_Crackme_Level…= "IOLI Crackme Level 0x04\n"
         5e 86 04 08
0804852f e8 60 fe       CALL    <EXTERNAL>::printf                         int printf(char * __format, ...)
         ff ff
08048534 c7 04 24       MOV     dword ptr [ESP]=>local_a0,s_Password:_08048677  = "Password: "
         77 86 04 08
0804853b e8 54 fe       CALL    <EXTERNAL>::printf                         int printf(char * __format, ...)
         ff ff
08048540 8d 45 88       LEA     EAX=>local_7c,[EBP + -0x78]
08048543 89 44 24 04    MOV     dword ptr [ESP + local_9c],EAX
08048547 c7 04 24       MOV     dword ptr [ESP]=>local_a0,DAT_08048682          = 25h    %
         82 86 04 08
0804854e e8 21 fe       CALL    <EXTERNAL>::scanf                          int scanf(char * __format, ...)
         ff ff
08048553 8d 45 88       LEA     EAX=>local_7c,[EBP + -0x78]
08048556 89 04 24       MOV     dword ptr [ESP]=>local_a0,EAX
08048559 e8 26 ff       CALL    check                                      undefined check(char * param_1)
         ff ff
0804855e b8 00 00       MOV     EAX,0x0
         00 00
08048563 c9             LEAVE
08048564 c3             RET
```

```
undefined4 main(void)

{
  char local_7c [120];

  printf("IOLI Crackme Level 0x04\n");
  printf("Password: ");
  scanf("%s",local_7c);
  check(local_7c);
  return 0;
}
```

The pseudocode is:

1. *0x0804854e: Instruction scanf to take in input the password*
2. *0x08048559: Call check function with parameter the input string*
3. *0x0804855e: Put value 0 to the return value and terminate the program*

Function check()



```
                    LAB_08048498                                    XREF[1]:     080484f9(j)
08048498 8b 45 08        MOV        EAX,dword ptr [EBP + param_1]
0804849b 89 04 24        MOV        dword ptr [ESP]=>local_2c,EAX
0804849e e8 e1 fe        CALL       <EXTERNAL>::strlen                          size_t strlen(char * __s)
         ff ff
080484a3 39 45 f4        CMP        dword ptr [EBP + local_10],EAX
080484a6 73 53           JNC        LAB_080484fb
080484a8 8b 45 f4        MOV        EAX,dword ptr [EBP + local_10]
080484ab 03 45 08        ADD        EAX,dword ptr [EBP + param_1]
080484ae 0f b6 00        MOVZX      EAX,byte ptr [EAX]
080484b1 88 45 f3        MOV        byte ptr [EBP + local_11],AL
080484b4 8d 45 fc        LEA        EAX=>local_8,[EBP + -0x4]
080484b7 89 44 24 08     MOV        dword ptr [ESP + local_24],EAX
080484bb c7 44 24        MOV        dword ptr [ESP + local_28],DAT_08048638     = 25h    %
         04 38 86
         04 08
080484c3 8d 45 f3        LEA        EAX=>local_11,[EBP + -0xd]
080484c6 89 04 24        MOV        dword ptr [ESP]=>local_2c,EAX
080484c9 e8 d6 fe        CALL       <EXTERNAL>::sscanf                          int sscanf(char * __s, char * __...
         ff ff
080484ce 8b 55 fc        MOV        EDX,dword ptr [EBP + local_8]
080484d1 8d 45 f8        LEA        EAX=>local_c,[EBP + -0x8]
080484d4 01 10           ADD        dword ptr [EAX]=>local_c,EDX
080484d6 83 7d f8 0f     CMP        dword ptr [EBP + local_c],0xf
080484da 75 18           JNZ        LAB_080484f4
080484dc c7 04 24        MOV        dword ptr [ESP]=>local_2c,s_Password_OK!_08048…= "Password OK!\n"
         3b 86 04 08
080484e3 e8 ac fe        CALL       <EXTERNAL>::printf                          int printf(char * __format, ...)
         ff ff
080484e8 c7 04 24        MOV        dword ptr [ESP]=>local_2c,0x0
         00 00 00 00
080484ef e8 c0 fe        CALL       <EXTERNAL>::exit                            void exit(int __status)
         ff ff
                    LAB_080484f4                                    XREF[1]:     080484da(j)
080484f4 8d 45 f4        LEA        EAX=>local_10,[EBP + -0xc]
080484f7 ff 00           INC        dword ptr [EAX]=>local_10
080484f9 eb 9d           JMP        LAB 08048498
```

```
void check(char *param_1)

{
  size_t sVar1;
  char local_11;
  uint local_10;
  int local_c;
  int local_8;

  local_c = 0;
  local_10 = 0;
  while( true ) {
    sVar1 = strlen(param_1);
    if (sVar1 <= local_10) {
      printf("Password Incorrect!\n");
      return;
    }
    local_11 = param_1[local_10];
    sscanf(&local_11,"%d",&local_8);
    local_c = local_c + local_8;
    if (local_c == 0xf) break;
    local_10 = local_10 + 1;
  }
  printf("Password OK!\n");
                    /* WARNING: Subroutine does not return
  exit(0);
}
```

The pseudocode is:

1. *0x08048491: Set 0 to the variables local_10 and local_c*
2. *0x0804849e: Call strlen function with the input password and save the result in sVar1*
3. *0x080484a3: If sVar1 is less or equal than local_10 jump to LAB_080484fb that represents the part where the program prints incorrect password and terminate the execution*
4. *0x080484c9: Call sscanf function to convert to int the character in pos local_10 of the input password and save the result in the variable local_8*
5. *0x080484d4: Sum the variable local_8 to the variable local_c*
6. *0x080484d6: If the sum in the variable local_c is equal to 15*
   6.1. *0x080484e3: Printf password ok*
7. *0x080484f7: Make local_10++ and return to the step 2*

Logical flow:

1. *Enter the password*
2. *Pass it to the check function*

Check function controls that the sum of the number in the password is 15 otherwise print incorrect password.