Reverse Engineering using Ghidra

-Devasheesh Vaid

- **Ghidra**

To perform the task of reverse engineering, first I downloaded Ghidra, which is a free and open-source reverse engineering tool and unzipped the downloaded software.

To start Ghidra. Go to the directory containing Ghidra files, open terminal and run the below command-

./ghidraRun

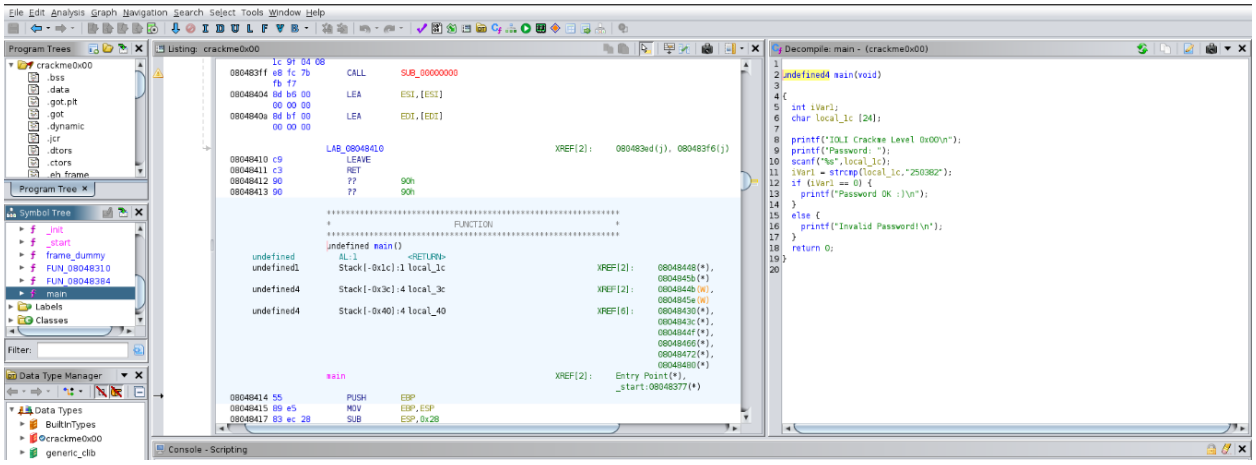
It will open the Ghidra GUI. Then I worked on each binary file given by analyzing the assembly code and the corresponding decompiled code.

---

1. crackme0x00

Password: 250382

Open the file in ghidra and from the symbol tree (on the left) click on the main() function. It will open the the assembly code and the corresponding decompiled code, as below.



- **main()**

On observing the decompiled code as shown below, we see that after taking an input for password as a string, we compare that input string stored in local_lc to the string “250382”, using strcmp(). The strcmp() gives the value as 0 when the two strings in comparison are equal. So, if the iVarl stores 0, in other words if the input string is same as “250382”, it prints “Password OK :)”.

```
Decompile: main - (crackme0x00)
1
2  undefined4 main(void)
3
4  {
5    int iVar1;
6    char local_1c [24];
7
8    printf("IOLI Crackme Level 0x00\n");
9    printf("Password: ");
10   scanf("%s",local_1c);
11   iVar1 = strcmp(local_1c,"250382");
12   if (iVar1 == 0) {
13     printf("Password OK :)\n");
14   }
15   else {
16     printf("Invalid Password!\n");
17   }
18   return 0;
19 }
20
```

On entering the password determined, we see the success message as below.

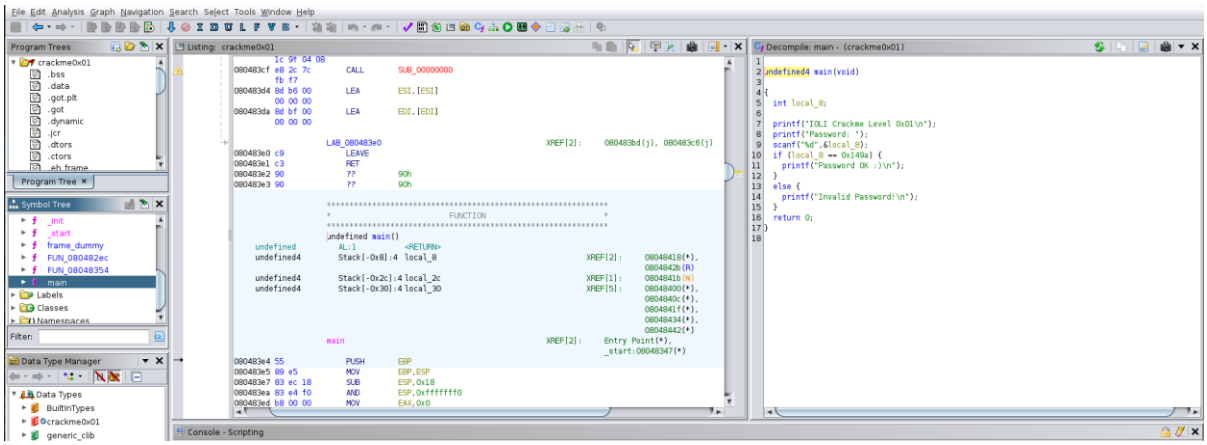
```
ub@ub-VirtualBox:~/Downloads/lab1/IOLI-crackme$ ./crackme0x00
IOLI Crackme Level 0x00
Password: 250382
Password OK :)
ub@ub-VirtualBox:~/Downloads/lab1/IOLI-crackme$
```

---

2. crackme0x01

Password: 5274 (Hex: 0x149a)

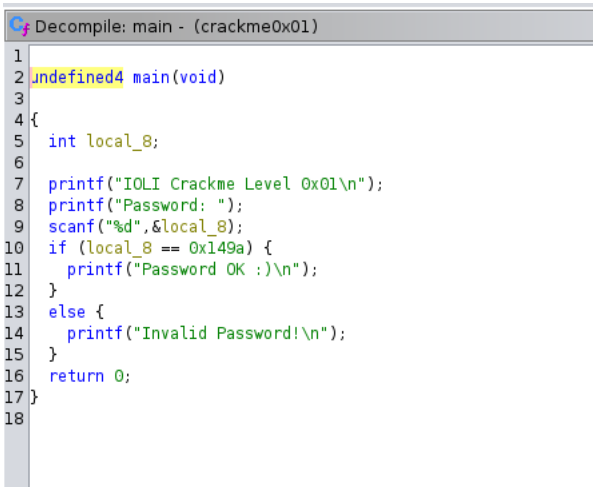
Open the file in ghidra and from the symbol tree (on the left) click on the main() function. It will open the the assembly code and the corresponding decompiled code, as below.



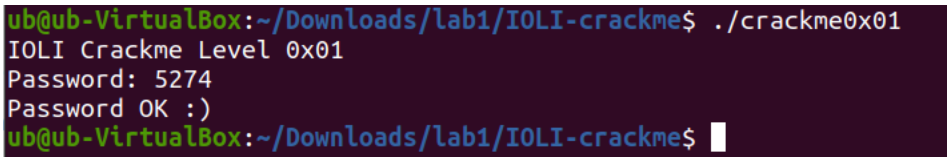
- **main()**

  On observing the decompiled code as shown below, we see that after taking an input for password as an integer, we compare that input stored in local_8 to a number in hexadecimal representation "0x149a", which translates to 5274 in decimal representation. In line 10 we compare the user input,

local_8, to 0x149a. So, if the integer input is 5274 (decimal for 0x149a), it prints “Password OK :)” message.



On entering the password determined, we see the success message as below.

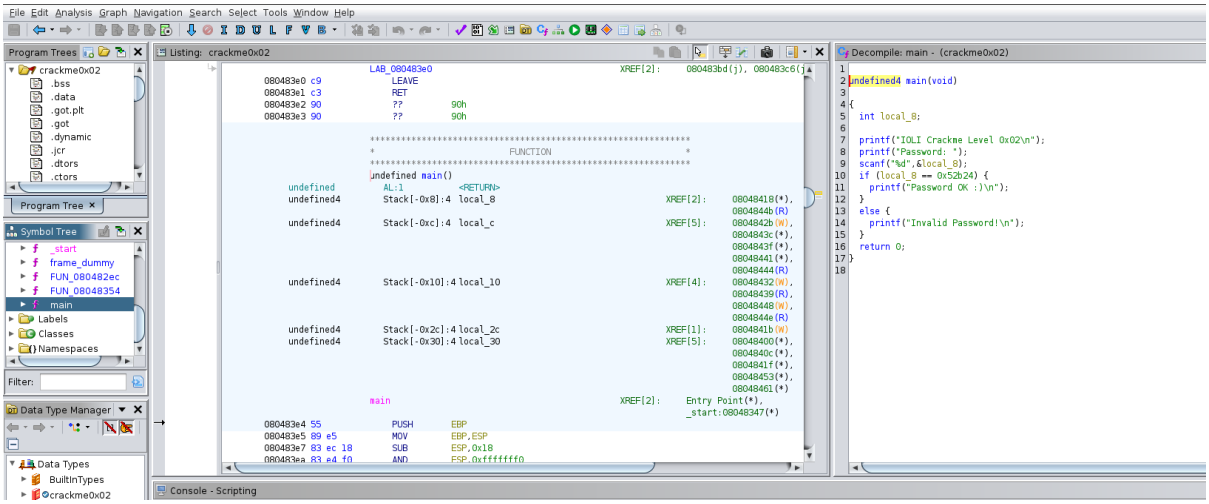


---

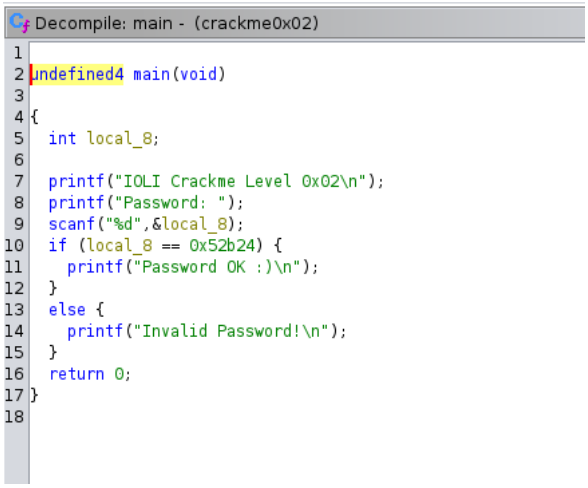
3. crackme0x02

Password: 338724 (Hex: 0x52b24)

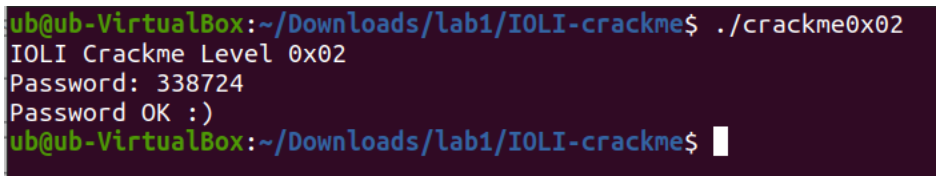
Open the file in ghidra and from the symbol tree (on the left) click on the main() function. It will open the the assembly code and the corresponding decompiled code, as below.



- **main()**
  This is similar to the previous case. On observing the decompiled code as shown below, we see that after taking an input for password as an integer, we compare that input stored in local_8 to a number in hexadecimal representation “0x52b24”, which translates to 338724 in decimal representation. In line 10 we compare the user input, local_8, to 0x52b24. So, if the integer input is 338724 (decimal for 0x52b24), it prints “Password OK :)” message.



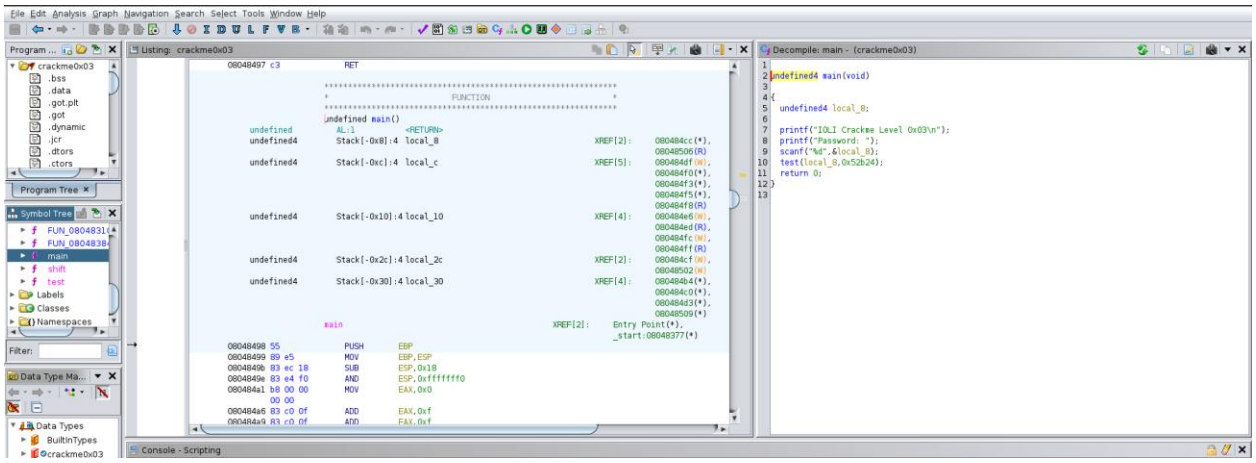
On entering the password determined, we see the success message as below.



4. crackme0x03:

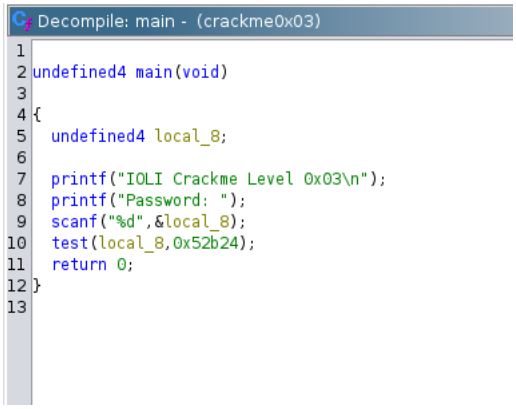
Password: 338724 (Hex: 0x52b24)

Open the file in ghidra and from the symbol tree (on the left) click on the main() function. It will open the the assembly code and the corresponding decompiled code, as below.



- **main()**

On observing the decompiled code as shown below, we see that on receiving the input from the user we call test() which takes as parameter the input number (local_8) and '0x52b24' (Hex for 338724). So we go to the test() to understand its execution.



- **test()**

To this test(), the first parameter passed is the user input and the second parameter is '0x52b24' (Hex for 338724), as discussed above. Here, we see that on comparing the two parameters we call another function, shift() but with different values for each case, when the values match and when they don't. So now moving on to the shift().

Decompile: test - (crackme0x03)

```
1
2  void test(int param_1,int param_2)
3
4  {
5    if (param_1 == param_2) {
6      shift("Sdvvzrug#RN$$$#=,");
7    }
8    else {
9      shift("Lqydolg#Sdvvzrug$");
10   }
11   return;
12 }
13
```

- **Shift()**
  In the shift(), the function simply performs the shift of -3 on the character string is passed to it, similar to the Caesar cipher with shift value of 3.
  So, the string it takes when param1=param2 in previous function test() is "Sdvvzrug#RN$$$#=," which, on performing the shift to the ascii of each character by -3, translates to "Password OK!!! :)".
  And when param1!=param2 in test(), the passed string, "Lqydolg#Sdvvzrug$", in shift() translates to "Invalid Password!".

Decompile: shift - (crackme0x03)

```
1
2  void shift(char *param_1)
3
4  {
5    size_t sVar1;
6    uint local_80;
7    char local_7c [120];
8
9    local_80 = 0;
10   while( true ) {
11     sVar1 = strlen(param_1);
12     if (sVar1 <= local_80) break;
13     local_7c[local_80] = param_1[local_80] + -3;
14     local_80 = local_80 + 1;
15   }
16   local_7c[local_80] = '\0';
17   printf("%s\n",local_7c);
18   return;
19 }
20
```

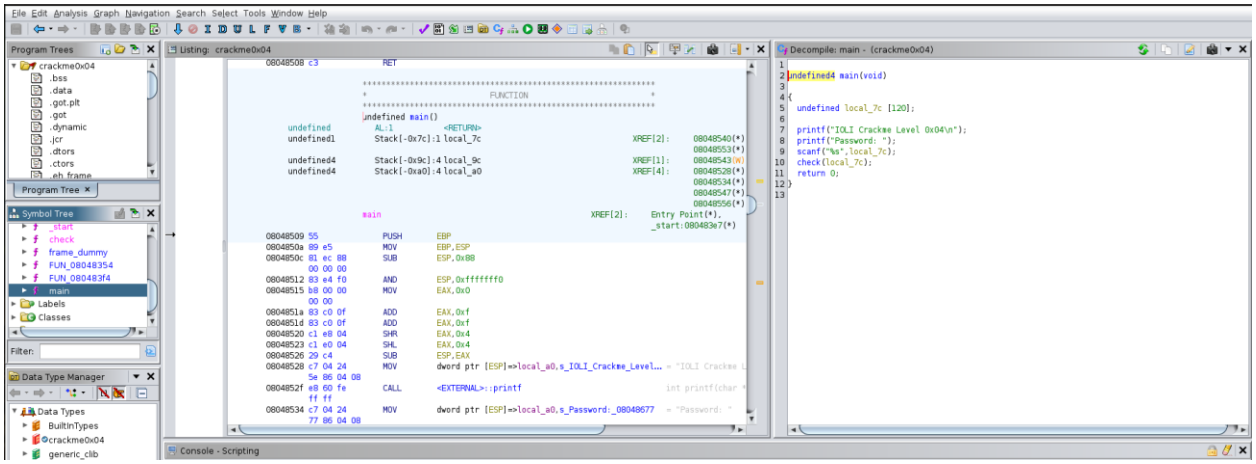
On entering the password determined, we see the success message as below.

```
ub@ub-VirtualBox:~/Downloads/lab1/IOLI-crackme$ ./crackme0x03
IOLI Crackme Level 0x03
Password: 338724
Password OK!!! :)
ub@ub-VirtualBox:~/Downloads/lab1/IOLI-crackme$
```

5. crackme0x04:

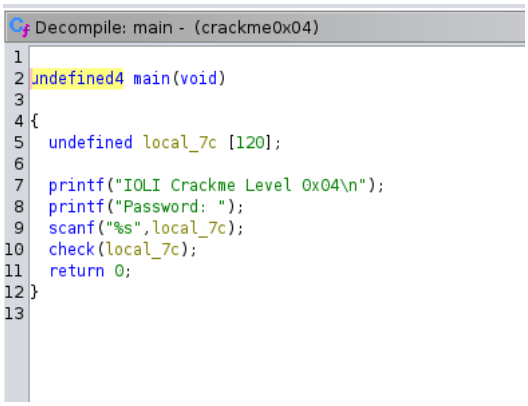
Password: Any number with the sum of first n numbers=15, where 2<=n<=password length

Open the file in ghidra and from the symbol tree (on the left) click on the main() function. It will open the the assembly code and the corresponding decompiled code, as below.



- **main()**

  On observing the main(), we see that on taking the input as a string, a call to check() is made with the input string as parameter. So we further investigate check().



- **check()**

  Inside check() we analyse the while loop which only terminates if we get the password or exceed the length of the input password.

  While the length of the string is less than the length of the password, in each iteration, we take the character at location local_10 (starting from 0) and store it to local_11. Then using sscanf(), we read the input from local_11 and store it as an integer value to local_8.

  In each iteration we then add the values stored at local_8 to local_c (initialized to 0) creating an eventual sum at each step. Any any step if the sum equals 0xf or 15, we break out of the loop and print “Password OK!”. Otherwise, if we reach the end of the input string and the sum never becomes 15, we print “Password Incorrect!”

Decompile: check - (crackme0x04)

```
1
2  void check(char *param_1)
3
4  {
5    size_t sVar1;
6    char local_11;
7    uint local_10;
8    int local_c;
9    int local_8;
10
11   local_c = 0;
12   local_10 = 0;
13   while( true ) {
14     sVar1 = strlen(param_1);
15     if (sVar1 <= local_10) {
16       printf("Password Incorrect!\n");
17       return;
18     }
19     local_11 = param_1[local_10];
20     sscanf(&local_11,"%d",&local_8);
21     local_c = local_c + local_8;
22     if (local_c == 0xf) break;
23     local_10 = local_10 + 1;
24   }
25   printf("Password OK!\n");
26                     /* WARNING: Subroutine does not return */
27   exit(0);
28 }
29
```

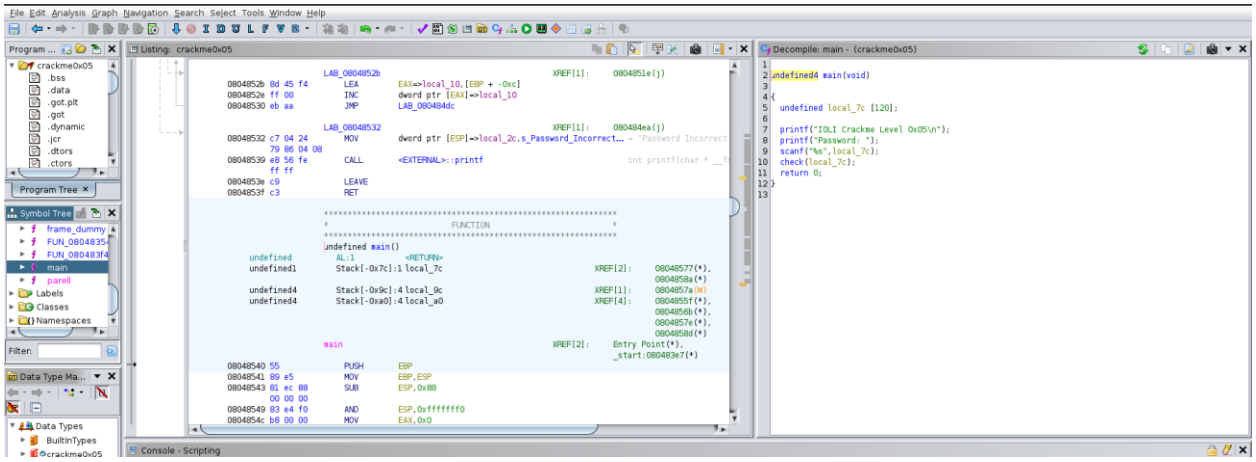
On entering different passwords, we see the corresponding message, as below. The password is accepted only if the sum of first n digits becomes equal to 15, for 2<=n<=length(password).

```
ub@ub-VirtualBox:~/Downloads/lab1/IOLI-crackme$ ./crackme0x04
IOLI Crackme Level 0x04
Password: 5631239
Password OK!
ub@ub-VirtualBox:~/Downloads/lab1/IOLI-crackme$ ./crackme0x04
IOLI Crackme Level 0x04
Password: 96
Password OK!
ub@ub-VirtualBox:~/Downloads/lab1/IOLI-crackme$ ./crackme0x04
IOLI Crackme Level 0x04
Password: 3333333
Password OK!
ub@ub-VirtualBox:~/Downloads/lab1/IOLI-crackme$ ./crackme0x04
IOLI Crackme Level 0x04
Password: 823117
Password OK!
ub@ub-VirtualBox:~/Downloads/lab1/IOLI-crackme$ ./crackme0x04
IOLI Crackme Level 0x04
Password: 77362
Password Incorrect!
ub@ub-VirtualBox:~/Downloads/lab1/IOLI-crackme$
```

6. crackme0x05:

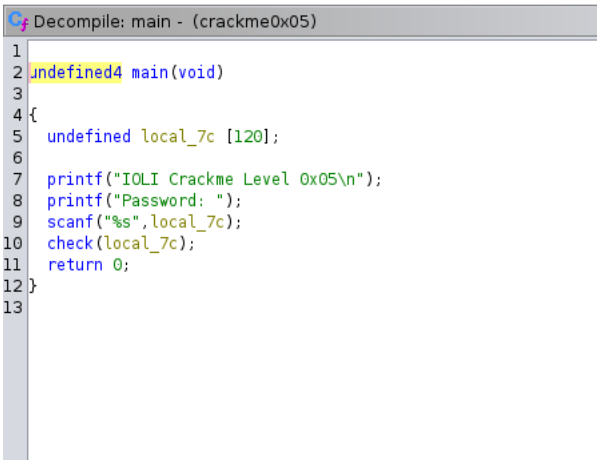
Password: An even number having the sum of first n digits 16, where 2<=n<=length(password)

Open the file in ghidra and from the symbol tree (on the left) click on the main() function. It will open the the assembly code and the corresponding decompiled code, as below.



- **main()**

On observing the main(), we see that on taking the input as a string, a call to check() is made with the input string as parameter. So we further investigate check().



- **check()**

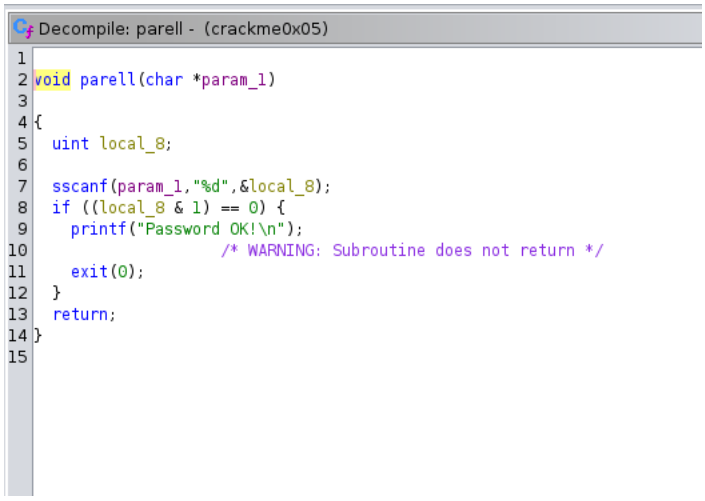
Inside check() we analyze the while loop which only terminates when we exceed the length of the input password, in case the entered string is not a password.

While the length of the string is less than the length of the password, in each iteration, we take the character at location local_10 (starting from 0) and store it to local_11. Then using sscanf(), we read the input from local_11 and store it as an integer value to local_8.

In each iteration we then add the values stored at local_8 to local_c (initialized to 0) creating an eventual sum at each step. At any step if the sum equals 0x10 or 16, we break out of the loop and make another

function call to parell(). Otherwise, if we reach the end of the input string and the sum never becomes 16, we print “Password Incorrect!”

- **parell()**
  On analyzing the parell(), which takes the entire input string as parameter param_1, it first reads the param_1 and stores as integer in local_8. It then performs a bitwise AND with 1, if the result is 0 then it prints “Password OK!” and exits the execution. In other words, compares the LSB of input with 1, if the number is even (LSB=0), it accepts it as password, else it rejects it when the password is odd. Otherwise it returns to check() and resumes its execution, eventually printing out “Password Incorrect!”



On entering different passwords, we see the corresponding message, as below. The password is accepted only if the number is even and sum of first n digits is equal to 16, for 2<=n<=length(password).

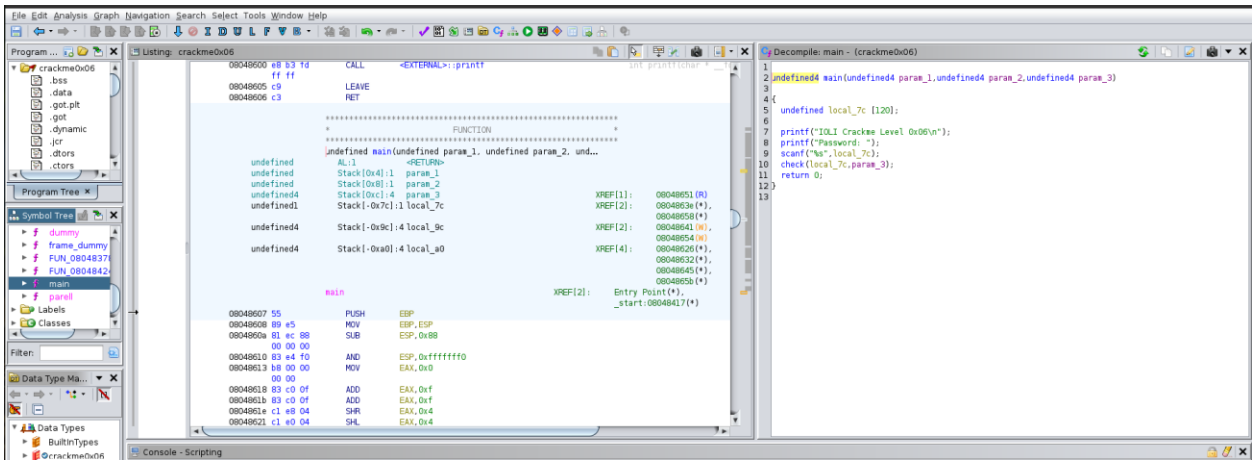
```
ub@ub-VirtualBox:~/Downloads/lab1/IOLI-crackme$ ./crackme0x05
IOLI Crackme Level 0x05
Password: 8446532
Password OK!
ub@ub-VirtualBox:~/Downloads/lab1/IOLI-crackme$ ./crackme0x05
IOLI Crackme Level 0x05
Password: 916
Password OK!
ub@ub-VirtualBox:~/Downloads/lab1/IOLI-crackme$ ./crackme0x05
IOLI Crackme Level 0x05
Password: 743222
Password OK!
ub@ub-VirtualBox:~/Downloads/lab1/IOLI-crackme$ ./crackme0x05
IOLI Crackme Level 0x05
Password: 961
Password Incorrect!
ub@ub-VirtualBox:~/Downloads/lab1/IOLI-crackme$
```

---

7. crackme0x06 (*optional*):

Password: An even number whose sum of first n integers =16, for 2<=n<=length(password), and an environment variable whose name starts with "LOL".

Open the file in ghidra and from the symbol tree (on the left) click on the main() function. It will open the the assembly code and the corresponding decompiled code, as below.



- **main()**

On observing the main(), we see that here the main() takes command line arguments among which param_3 contains the list of environment variables. On taking the input as a string, a call to check() is made with the input string and param_3 as parameters. So we further investigate check().

```
Decompile: main - (crackme0x06)
1
2 undefined4 main(undefined4 param_1,undefined4 param_2,undefined4 param_3)
3
4 {
5   undefined local_7c [120];
6
7   printf("IOLI Crackme Level 0x06\n");
8   printf("Password: ");
9   scanf("%s",local_7c);
10  check(local_7c,param_3);
11  return 0;
12 }
13
```

- **check()**
  Inside check() we analyse the while loop which only terminates when we exceed the length of the input password, in case the entered string is not a password.
  While the length of the string is less than the length of the password, in each iteration, we take the character at location local_10 of param_1, i.e. input string (starting from 0), and store it to local_11. Then using sscanf(), we read the input from local_11 and store it as an integer value to local_8.
  In each iteration we then add the values stored at local_8 to local_c (initialized to 0) creating an eventual sum at each step. At any step if the sum equals 0x10 or 16, we break out of the loop and make another function call to parell() with param_1 (input string) and param_2 (env variable list). Otherwise, if we reach the end of the input string and the sum never becomes 16, we print "Password Incorrect!"

```
Decompile: check - (crackme0x06)
1
2 void check(char *param_1,undefined4 param_2)
3
4 {
5   size_t sVar1;
6   char local_11;
7   uint local_10;
8   int local_c;
9   int local_8;
10
11  local_c = 0;
12  local_10 = 0;
13  while( true ) {
14    sVar1 = strlen(param_1);
15    if (sVar1 <= local_10) break;
16    local_11 = param_1[local_10];
17    sscanf(&local_11,"%d",&local_8);
18    local_c = local_c + local_8;
19    if (local_c == 0x10) {
20      parell(param_1,param_2);
21    }
22    local_10 = local_10 + 1;
23  }
24  printf("Password Incorrect!\n");
25  return;
26 }
27
```

- **parell()**
  On analyzing the parell(), which takes param_1 (input string) and param_2 (env variable list) as arguments. It first reads param_1 and stores it as an integer in local_8. It then calls dummy() with local_8 and param_2 as arguments.
  After completing the call to dummy(), it checks if the value of iVarl. If the value if iVarl is not 0, it means we got a match to "LOLO" in the called dummy() and we further check if the input is even

number by performing bitwise AND between LSB and 1. If it is even number, we print “Passowrd OK!” and exit the execution.
Otherwise it returns to check() and resumes its execution, eventually printing out “Password Incorrect!”

Decompile: parell - (crackme0x06)

```
1
2  void parell(char *param_1,undefined4 param_2)
3
4  {
5    int iVar1;
6    int local_c;
7    uint local_8;
8
9    sscanf(param_1,"%d",&local_8);
10   iVar1 = dummy(local_8,param_2);
11   if (iVar1 != 0) {
12     for (local_c = 0; local_c < 10; local_c = local_c + 1) {
13       if ((local_8 & 1) == 0) {
14         printf("Password OK!\n");
15                     /* WARNING: Subroutine does not return */
16         exit(0);
17       }
18     }
19   }
20   return;
21 }
22
```

- **dummy()**
  On inspecting dummy(), it first checks if the value stored at the address pointed by the address local_8 * 4 + param_2 is 0. If so, it returns. Otherwise, it uses strncmp to compare the first 3 characters of the string pointed to by *(char **)(iVarl + param_2) with the string "LOLO." The result of the comparison is stored in the iVarl variable, which will be zero if the first 3 characters of the string match "LOLO" and we return 1 to parell() in that case. If they don't match, iVarl will be non-zero.

Decompile: dummy - (crackme0x06)

```
1
2  undefined4 dummy(undefined4 param_1,int param_2)
3
4  {
5    int iVar1;
6    int local_8;
7
8    local_8 = 0;
9    do {
10     if (*(int *)(local_8 * 4 + param_2) == 0) {
11       return 0;
12     }
13     iVar1 = local_8 * 4;
14     local_8 = local_8 + 1;
15     iVar1 = strncmp(*(char **)(iVar1 + param_2),"LOLO",3);
16   } while (iVar1 != 0);
17   return 1;
18 }
19
```

On entering different passwords, we see the corresponding message, as below. The password is accepted only if it is an even number whose sum of first n integers =16, for 2<=n<=length(password), and we have an environment variable whose name starts with “LOL”.

```
ub@ub-VirtualBox:~/Downloads/lab1/IOLI-crackme$ ./crackme0x06
IOLI Crackme Level 0x06
Password: 916
Password Incorrect!
ub@ub-VirtualBox:~/Downloads/lab1/IOLI-crackme$ export LOLone=1
ub@ub-VirtualBox:~/Downloads/lab1/IOLI-crackme$ ./crackme0x06
IOLI Crackme Level 0x06
Password: 916
Password OK!
ub@ub-VirtualBox:~/Downloads/lab1/IOLI-crackme$
```
