CS255 – Lab 1 – Reverse Engineering

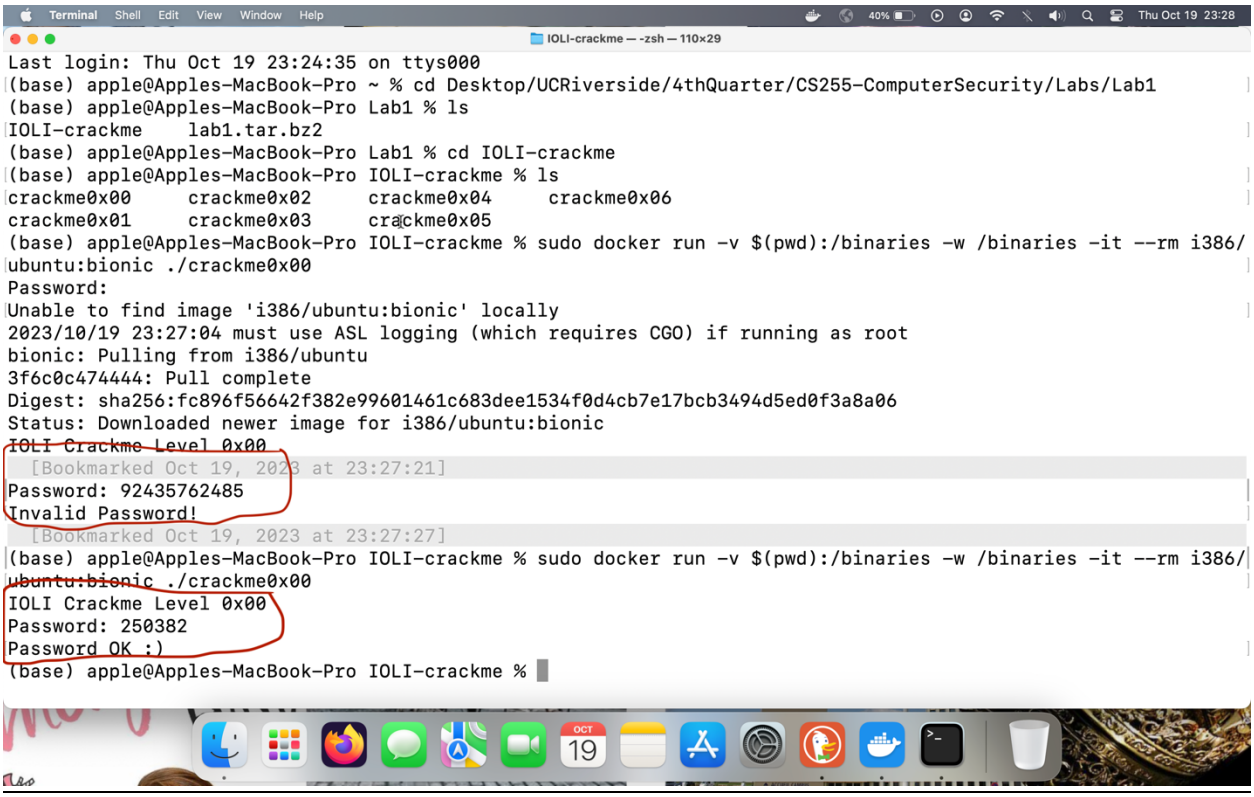
- *Lohith Kumar Bhambore*

Setup

- For analyzing these crackme files and running them, I used Ghidra and Docker desktop….
- Following the instructions as posted in the video on Yuja, I installed Ghidra to decompile and analyze the code files
- To run the code, I installed docker desktop and used the command shared on piazza – << sudo docker run -v $(pwd):/binaries -w /binaries -it --rm i386/ubuntu:bionic ./crackme0x00>>

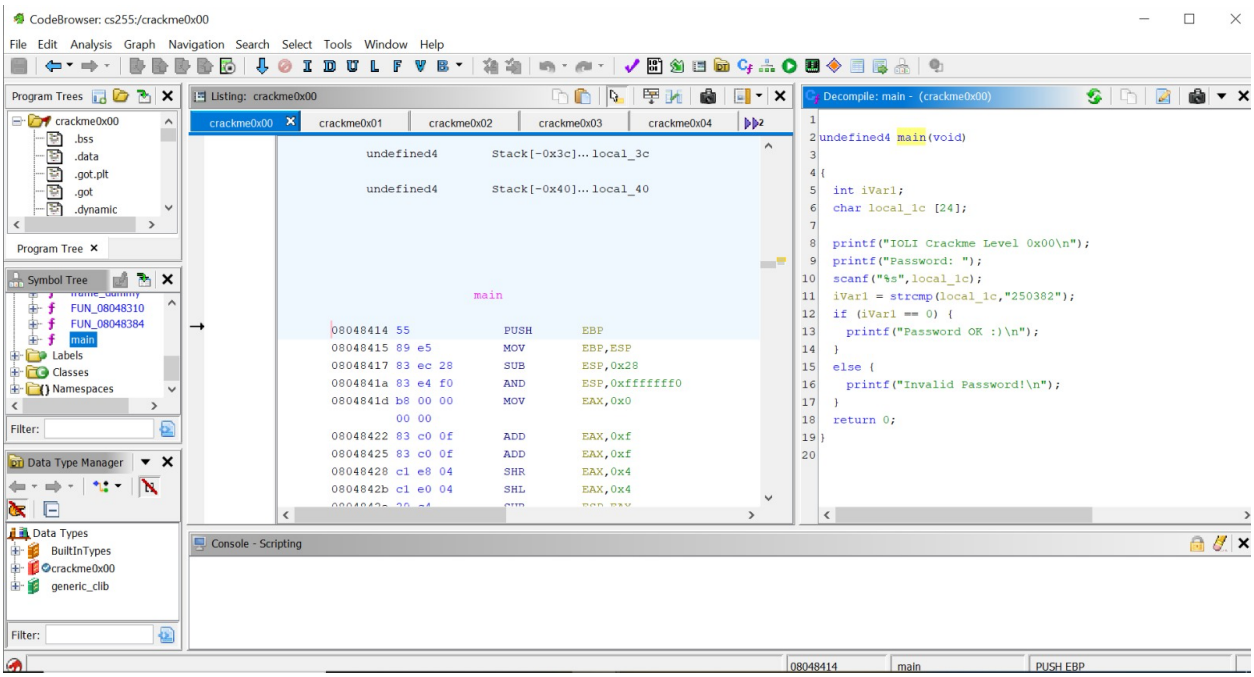
Crackme0x00

Password – *250382*

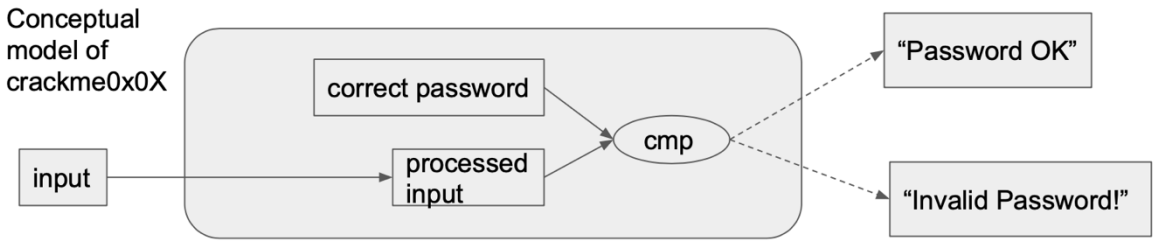


Analysis –

- Using Ghidra to decompile binary file crackme0x00 we can find the main function (as seen in the screenshot below)



- A static analysis of the main function reveals that the file reads input from the user using scanf and stores it in a string variable called local_1c
- The next step is comparison of local_1c and correct password. Shown below is a generic model of how we can get “Password OK” or “Invalid Password” (reference – slides by Zhengchuan Liang)



- In this case the program uses strcmp function which returns 0 if the two strings being compared are the same. The result is stored in a variable called iVar1.
- Using strcmp, the comparison is with the processed input and the correct password which is exposed and the value is “250382”
- And it prints “Password OK :)” if the input string is the same as “250382”.
- In all other cases it prints “Invalid password!”

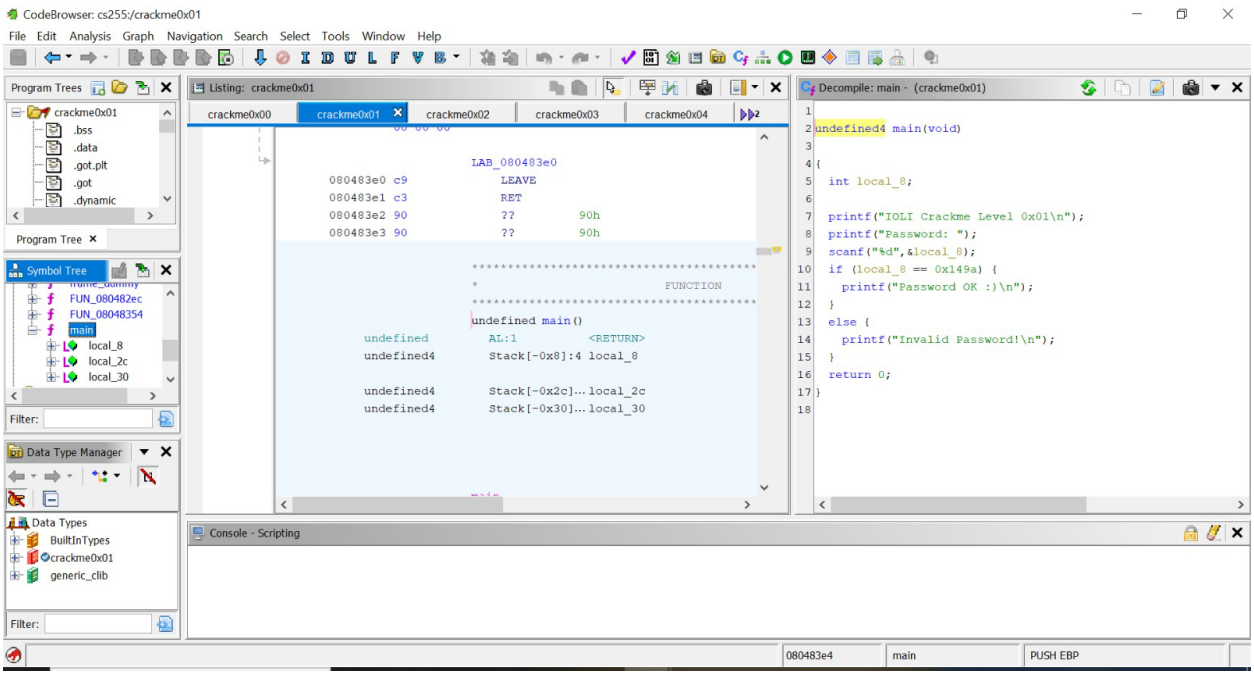
Crackme0x01

Password – *5274*

```
(base) apple@Apples-MacBook-Pro IOLI-crackme % docker run -v $(pwd):/binaries -w /binaries -it --rm i386/ubuntu:bionic ./crackme0x
01
IOLI Crackme Level 0x01
Password: 1348314
Invalid Password!
(base) apple@Apples-MacBook-Pro IOLI-crackme % docker run -v $(pwd):/binaries -w /binaries -it --rm i386/ubuntu:bionic ./crackme0x
01
IOLI Crackme Level 0x01
Password: 5274
Password OK :)
(base) apple@Apples-MacBook-Pro IOLI-crackme %
```

Analysis –

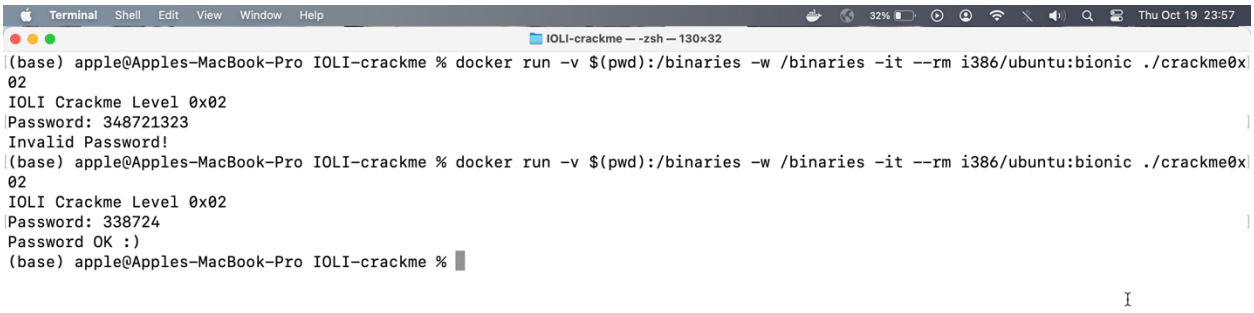
- Using Ghidra to decompile binary file crackme0x01 we can find the main function (as seen in the screenshot below)

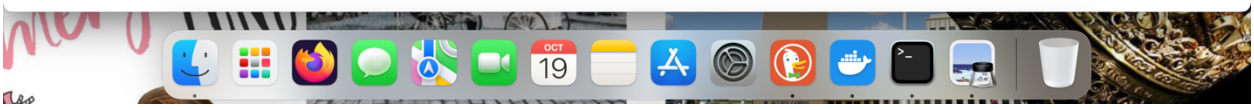


- A static analysis of the main function reveals that the file reads input from the user using scanf and stores it in a integer variable called local_8
- This time the comparison is b/w two numbers, one the value input by the user and the second one is a hexadecimal value '0x149a' which in decimal is '5274'.
- It doesn't matter if we are comparing hexadecimal and decimal numbers as the values are all converted to their binary representations. This is just a simple obfuscation on the correct password.
- Thus, if the user input is '5274', the program prints "Password OK :)"
- For any other user input, it prints "Invalid Password!"

Crackme0x02

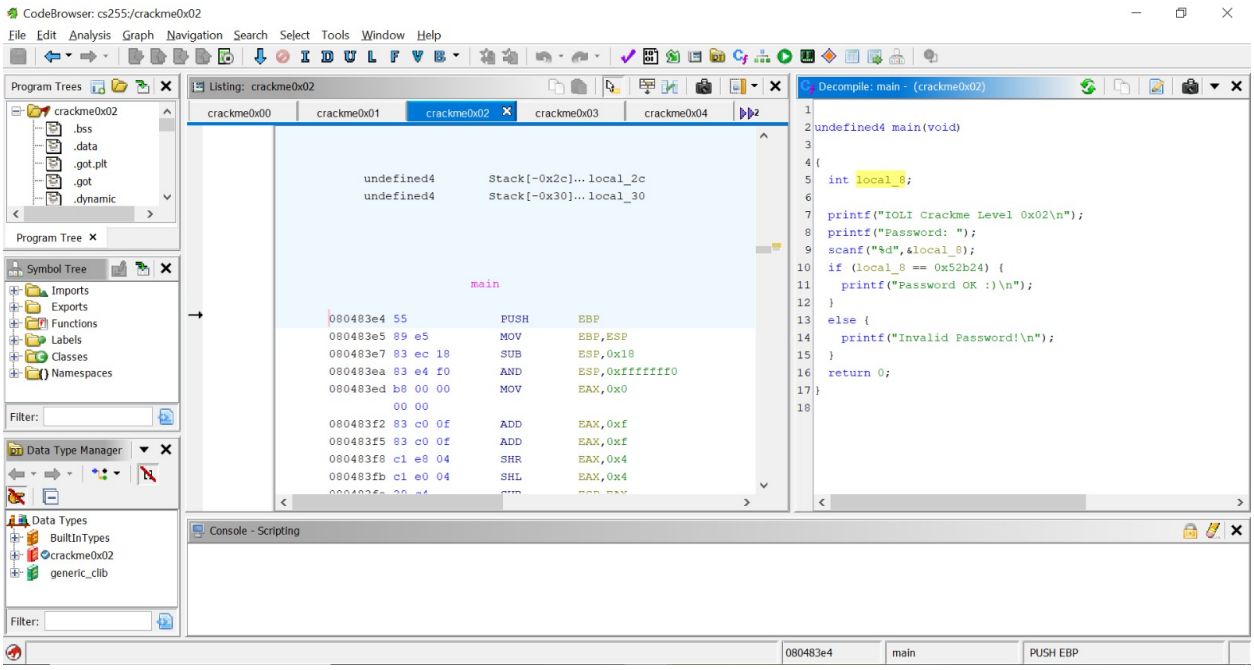
Password – *338724*





Analysis –

- Using Ghidra to decompile binary file crackme0x02 we can find the main function (as seen in the screenshot below)

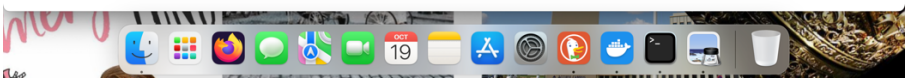


- This code is very similar to the previous one (0x01). This time again we have a comparison b/w two numbers, one the value input by the user and the second one is a hexadecimal value '0x52b24' which in decimal is '338724'.
- It doesn't matter if we are comparing hexadecimal and decimal numbers as the values are all converted to their binary representations (A simple obfuscation once again)
- Thus, if the user input is '338724', the program prints "Password OK :)"
- And just as before, for any other input, it prints "Invalid Password!"

Crackme0x03

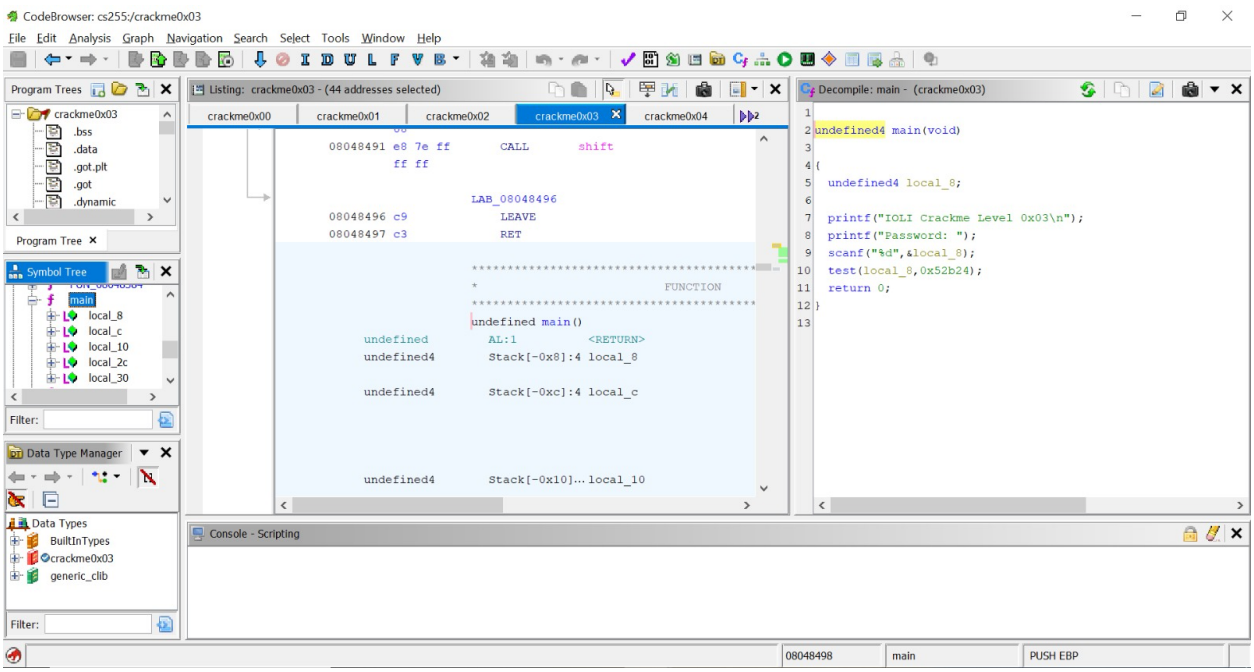
Password – *338724*

```
(base) apple@Apples-MacBook-Pro IOLI-crackme % docker run -v $(pwd):/binaries -w /binaries -it --rm i386/ubuntu:bionic ./crackme0x
03
IOLI Crackme Level 0x03
Password: fwjdbcf13
Invalid Password!
(base) apple@Apples-MacBook-Pro IOLI-crackme % docker run -v $(pwd):/binaries -w /binaries -it --rm i386/ubuntu:bionic ./crackme0x
03
IOLI Crackme Level 0x03
Password: 338724
Password OK!!! :)
(base) apple@Apples-MacBook-Pro IOLI-crackme %
```

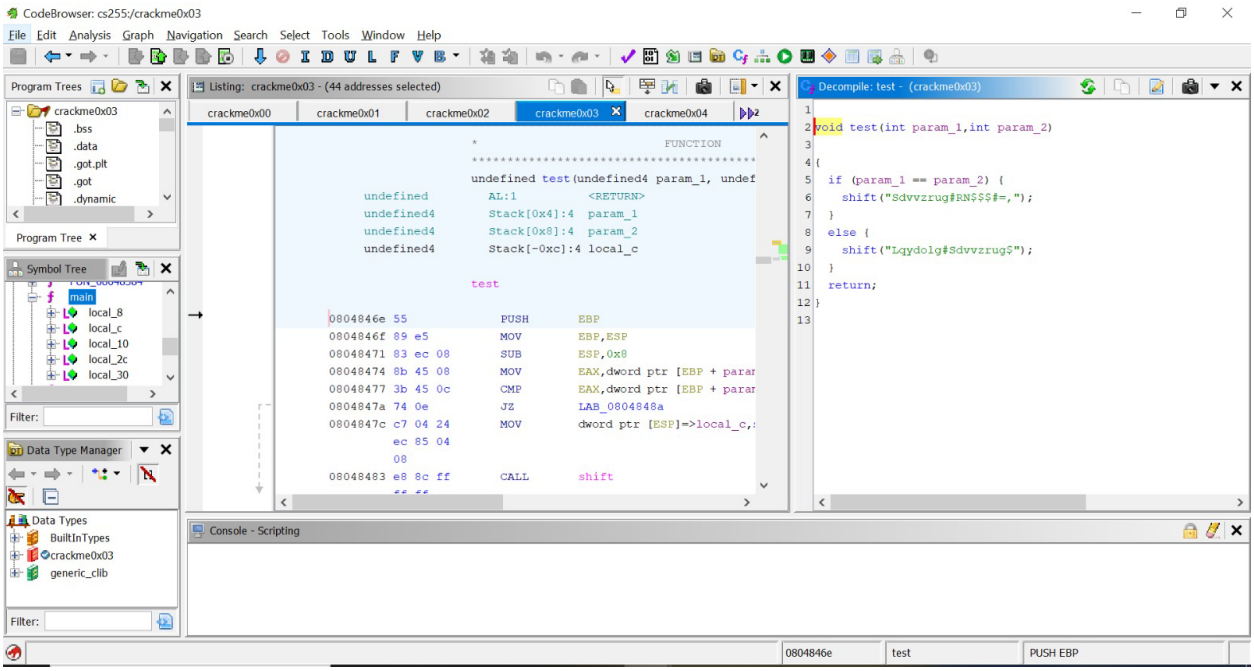


Analysis –

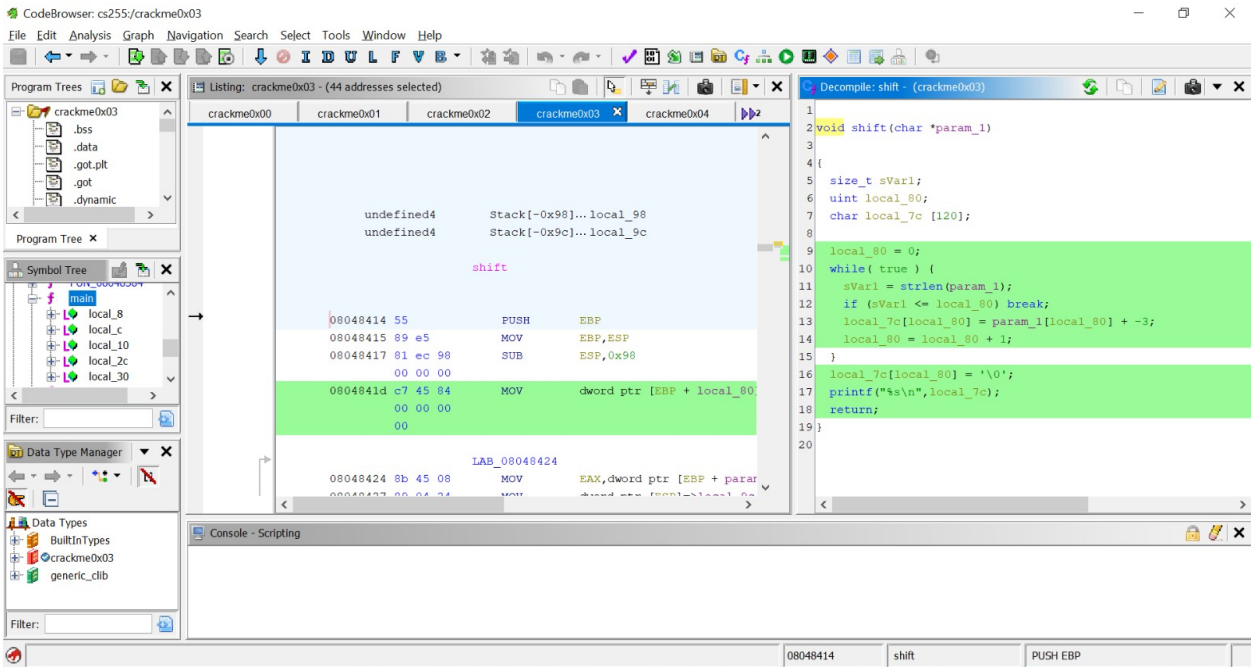
- Using Ghidra to decompile binary file crackme0x03 we can find the main function (as seen in the screenshot below)



- A static analysis of the main function reveals that the file reads input from the user using scanf and stores it in a local variable called local_8
- Now, instead of direct comparison, the program calls a test function with local_8 and the hexadecimal value ‘0x52b24’ as arguments. It finally returns 0.
- Next let’s look at the test function



- The test function compares the two integer parameters, and if they are equal, it calls the shift function with the string “Sdvvzrug#RN$$$#=,” as an argument
- If they are not equal, it still calls the shift function but with a different string – “Lqydolg#Sdvvzrug$”
- Now let’s take a look at the shift function

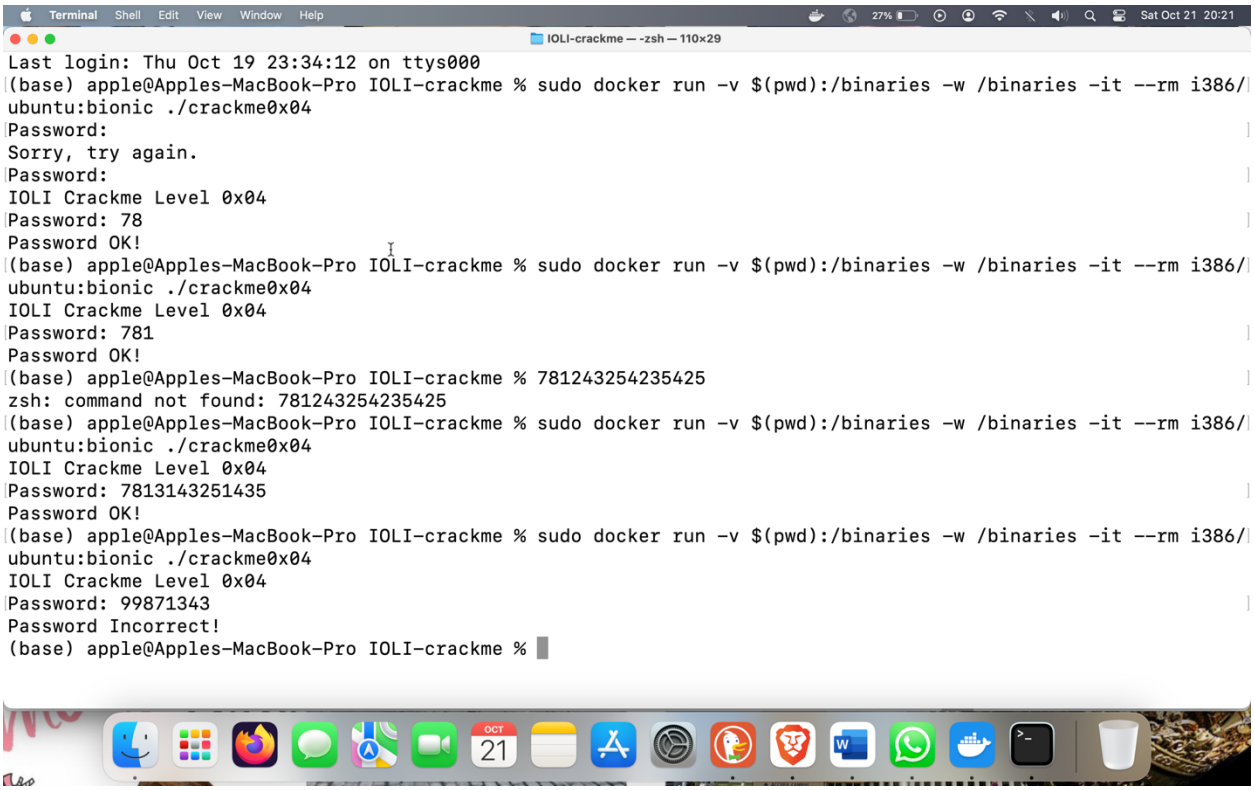


- The shift function takes a string called param_1 as an argument

- It keeps track of the size of param_1 using sVar1 and uses a local_80 as a counter for the while loop.
- The code in the while loop is basically a simple character shift. It is subtracting 3 from the ASCII values of each character in the string param_1 and updates it into the corresponding location in a new char array local_7c.
- It is essentially creating a new string one character at a time.
- Lastly it terminates the string by adding '\0' character to local_7c and then prints it.
- By performing this shift operation (subtracting 3 in ASCII value for each character) on the string "Sdvvzrug#RN$$$#=," gives us – "Password OK!!! :)"
- With string "Lqydolg#Sdvvzrug$", it gives us – "Invalid Password!"
- This just provides a simple obfuscation on the output strings printed, but the value that is actually being compared with the user input is still '0x52b24' or '338724'
- Thus, we can conclude that if the user enters the value '338724', it prints "Password OK!!! :)"
- In all other cases, it prints "Invalid Password!"

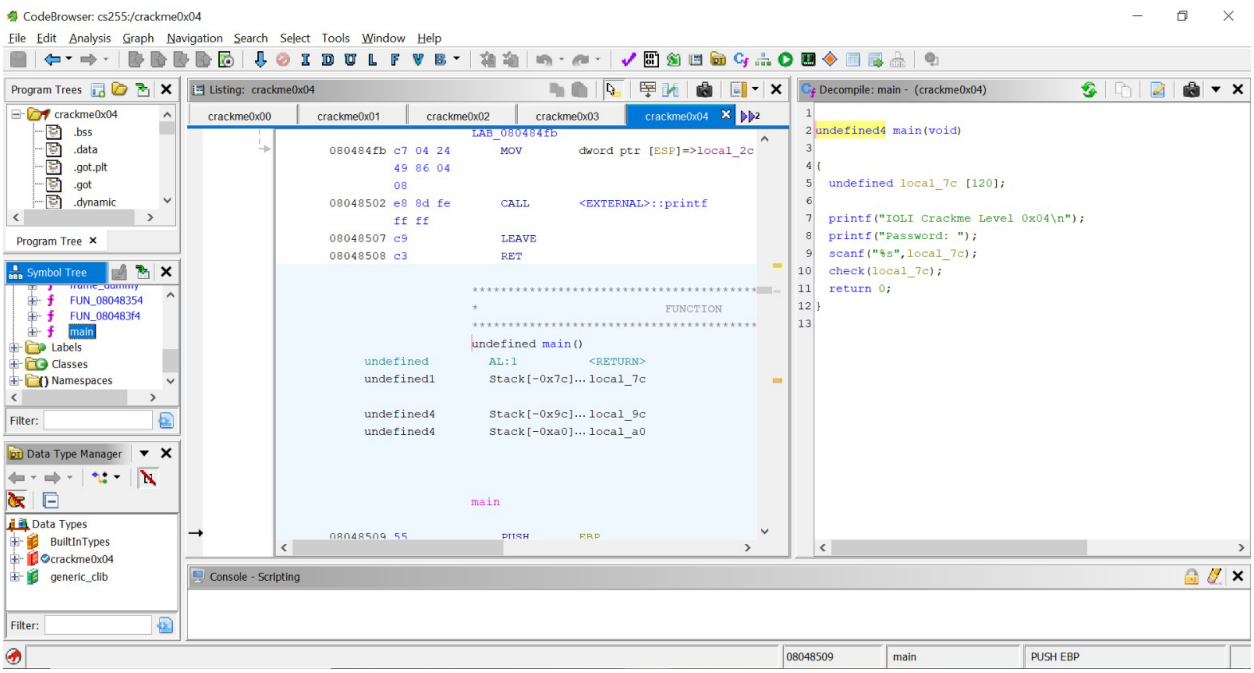
Crackme0x04

Password –
Sum of the digits in the input starting from the leftmost one should be 15 at some point (incrementally)
eg: 78, 781, 7813143251435 etc.

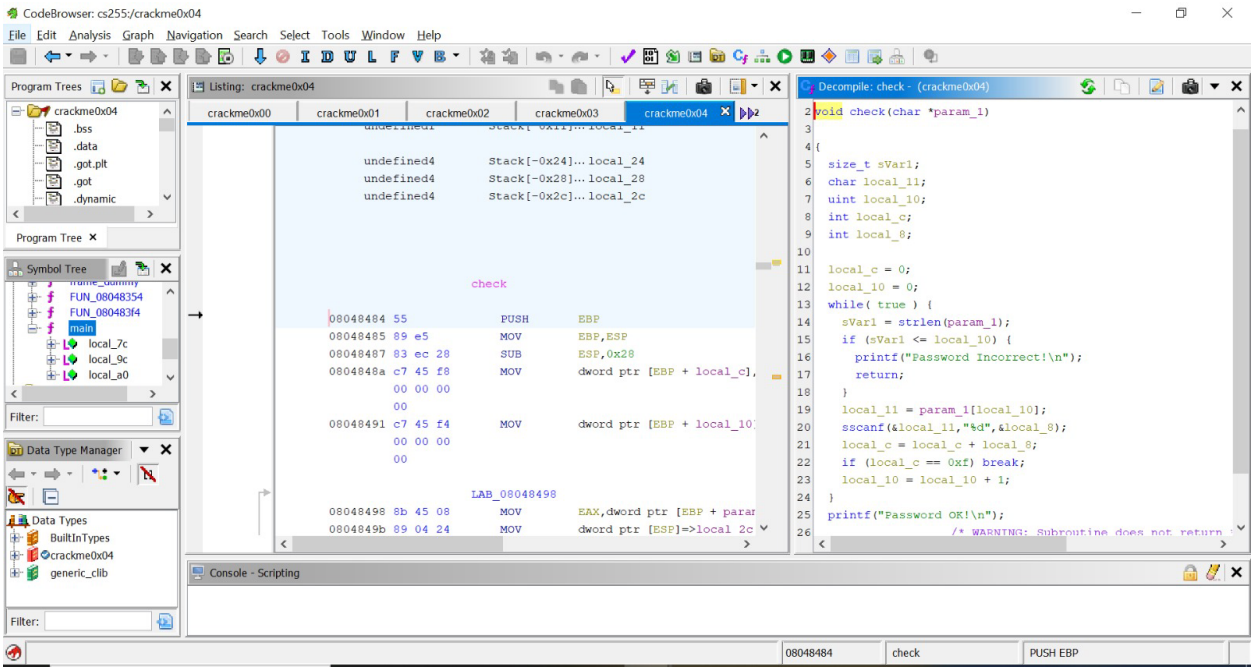


Analysis –

- Using Ghidra to decompile binary file crackme0x04 we can find the main function (as seen in the screenshot below)



- A static analysis of the main function reveals that the file reads input from the user using scanf and stores it in a local string variable called local_7c
- Now, instead of direct comparison with the correct password, the program calls a check function with local_7c as the argument. It finally returns 0.
- Next let's look at the check function



- The check function takes a string param_1 as an argument. It also has a bunch of local variables
- sVar1 to keep size of the string param_1.
- A char variable local_11 to read and temporarily store each character of the string param_1
- An integer local_10 which works as a counter for the while loop, which is initialized to 0
- An integer local_c which stores the sum so far and is initialized to 0.
- And an integer local_8 which stores the int value of the character local_11
- In the while loop, there is a check if the counter local_10 exceeds the size of param1, in which case, it prints “Password Incorrect!” and returns.
- If the condition is false, i.e the current char is not the last char of the string param_1, then it reads that character and stores it in local_11
- It then uses sscanf to convert that character to an integer and stores it in local_8
- Next, it increments adds the value of local_8 to the sum so far, i.e to local_c and stores it in local_c
- Next, it checks if the sum so far (local_c) is equal to 0xf (which is 15 in decimal system). If that is true, it breaks out of the loop.
- And then it prints “Password OK!” and returns
- Else, it increments the counter local_10 by 1 and goes to the next iteration.

- Since it is checking the sum of digits incrementally from the left and prints “Password OK!” as long as the sum so far reaches exactly 15 and doesn’t check the rest of the digits in the string, all valid passwords are the ones in which if the sum of digits of the input from the left equal 15 at any time.
- Invalid passwords are all in which the sum of digits from the left is not equal to 15 at any time.

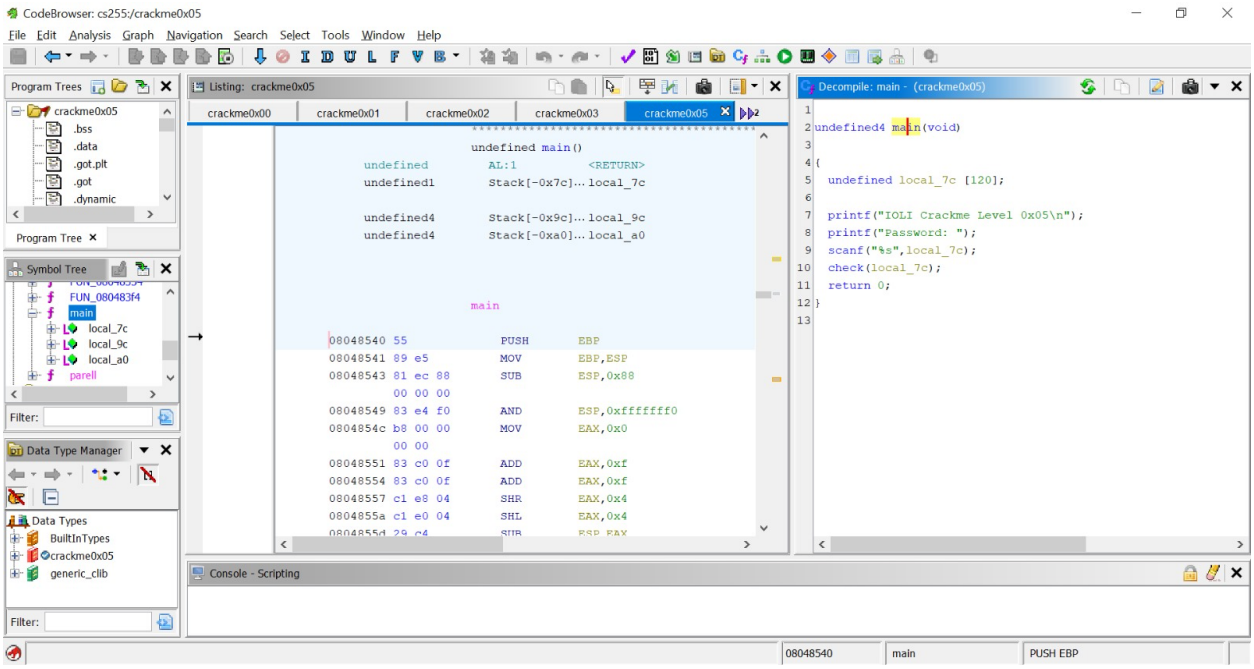
Crackme0x05

Password –
Sum of the digits in the input starting from the leftmost one should be 16 at some point (incrementally)
And
Number must be even.
eg: 88, 4444, 970, 974568, 882 etc.

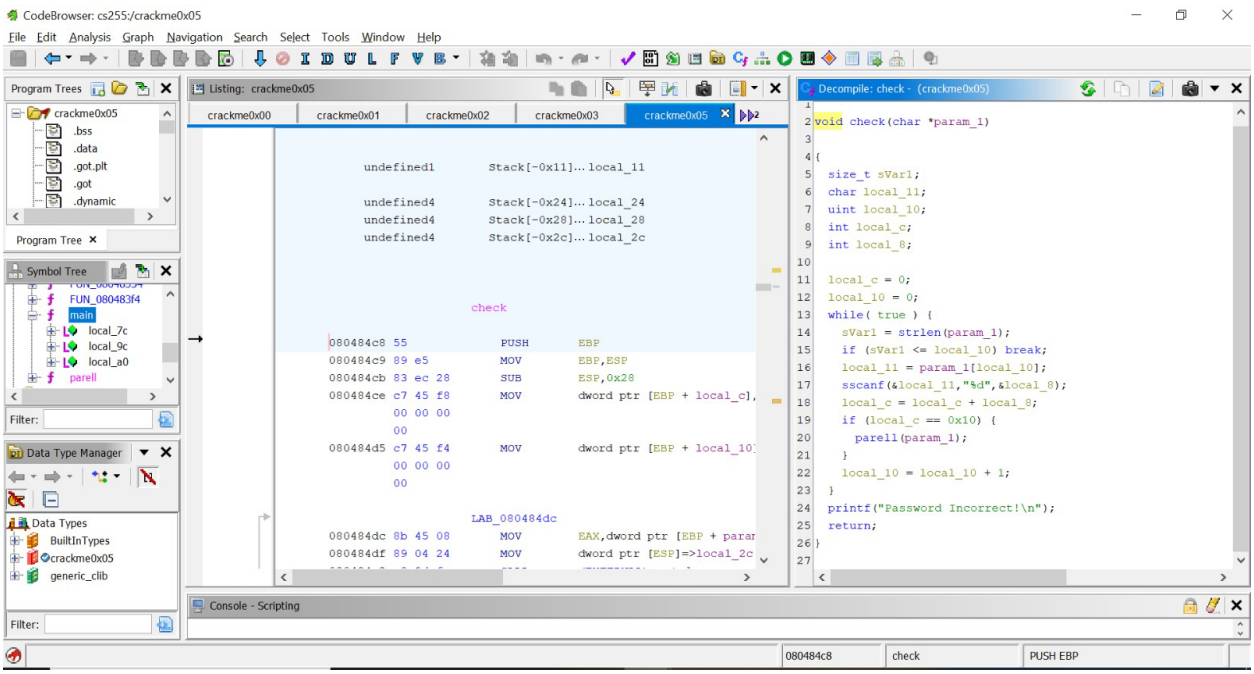
```
(base) apple@Apples-MacBook-Pro IOLI-crackme % sudo docker run -v $(pwd):/binaries -w /binaries -it --rm i386/ubuntu:bi
onic ./crackme0x05
IOLI Crackme Level 0x05
Password: 97
Password Incorrect!
(base) apple@Apples-MacBook-Pro IOLI-crackme % sudo docker run -v $(pwd):/binaries -w /binaries -it --rm i386/ubuntu:bi
onic ./crackme0x05
IOLI Crackme Level 0x05
Password: 970
Password OK!
(base) apple@Apples-MacBook-Pro IOLI-crackme % sudo docker run -v $(pwd):/binaries -w /binaries -it --rm i386/ubuntu:bi
onic ./crackme0x05
IOLI Crackme Level 0x05
Password: 974568
Password OK!
(base) apple@Apples-MacBook-Pro IOLI-crackme % sudo docker run -v $(pwd):/binaries -w /binaries -it --rm i386/ubuntu:bi
onic ./crackme0x05
IOLI Crackme Level 0x05
Password: 88
Password OK!
(base) apple@Apples-MacBook-Pro IOLI-crackme % sudo docker run -v $(pwd):/binaries -w /binaries -it --rm i386/ubuntu:bi
onic ./crackme0x05
IOLI Crackme Level 0x05
Password: 882
Password OK!
(base) apple@Apples-MacBook-Pro IOLI-crackme % sudo docker run -v $(pwd):/binaries -w /binaries -it --rm i386/ubuntu:bi
onic ./crackme0x05
IOLI Crackme Level 0x05
Password: 79
Password Incorrect!
(base) apple@Apples-MacBook-Pro IOLI-crackme % sudo docker run -v $(pwd):/binaries -w /binaries -it --rm i386/ubuntu:bi
onic ./crackme0x05
IOLI Crackme Level 0x05
Password: 792
Password OK!
```

Analysis –

- Using Ghidra to decompile binary file crackme0x05 we can find the main function (as seen in the screenshot below)

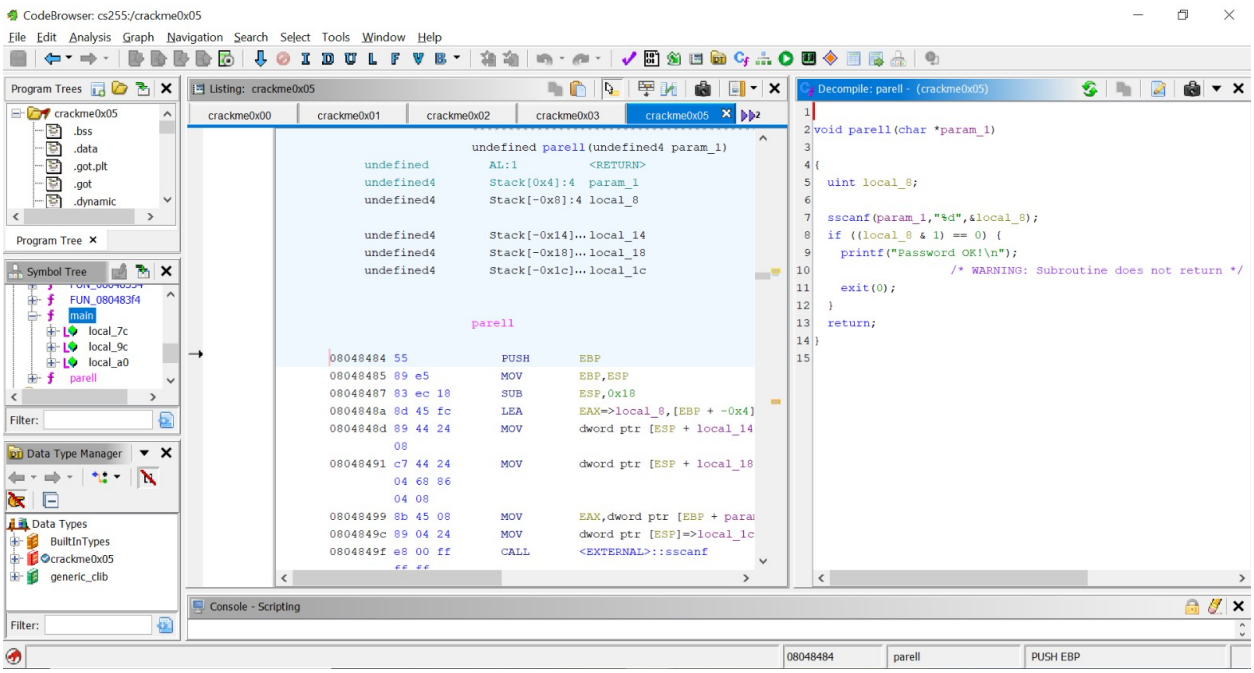


- A static analysis of the main function reveals that the file reads input from the user using scanf and stores it in a local string variable called local_7c
- Now, instead of direct comparison with the correct password, the program calls a check function with local_7c as the argument. It finally returns 0.
- Next let's look at the check function



- The check function takes a string param_1 as an argument. It also has a bunch of local variables similar to the previous one

- sVar1 to keep size of the string param_1.
- A char variable local_11 to read and temporarily store each character of the string param_1
- An integer local_10 which works as a counter for the while loop, which is initialized to 0
- An integer local_c which stores the sum so far and is initialized to 0.
- And an integer local_8 which stores the int value of the character local_11
- Next we have an indefinite while loop, within which we check if length of string param_1 is less than or equal to local_10. In case it is true, we break out of the loop
- Next, we read a character from the string param_1 (at the index local_10 which is a counter) and store it in local_11
- We then convert the character stored in local_11 into an integer using sscanf and store it in local_8
- We then add local_8 to the sum so far, i.e. local_c and store it in local_c
- We now check, if the sum so far or local_c is equal to the value '0x10', which is 16 in decimal system, if it is true we call the parell function while passing the string param_1
- We then increment the value of the counter local_10 by 1
- If the loop exits without the sum of digits from the left ever reaching 16 or 0x10, we print "Password Incorrect!"
- Now let's take a look at the parell function -



- The parell function takes a string argument called param_1
- It also has a local integer variable called local_8
- It first converts the string param_1 into an integer and stores it in local_8 using sscanf
- Now it checks if the number local_8 is even using bitwise AND operation
- If it is even, it prints "Password OK!" and exits the program.

- Since it is checking the sum of digits incrementally from the left and prints “Password OK!” as long as the sum so far reaches exactly 16 and the number is even and doesn’t check the rest of the digits in the string
- all valid passwords are the ones in which if the sum of digits of the input from the left equal 16 at any time and that the input is even.
- Invalid passwords are all in which the sum of digits from the left is not equal to 16 at any time and cases where the sum is 16, but the number is not even.