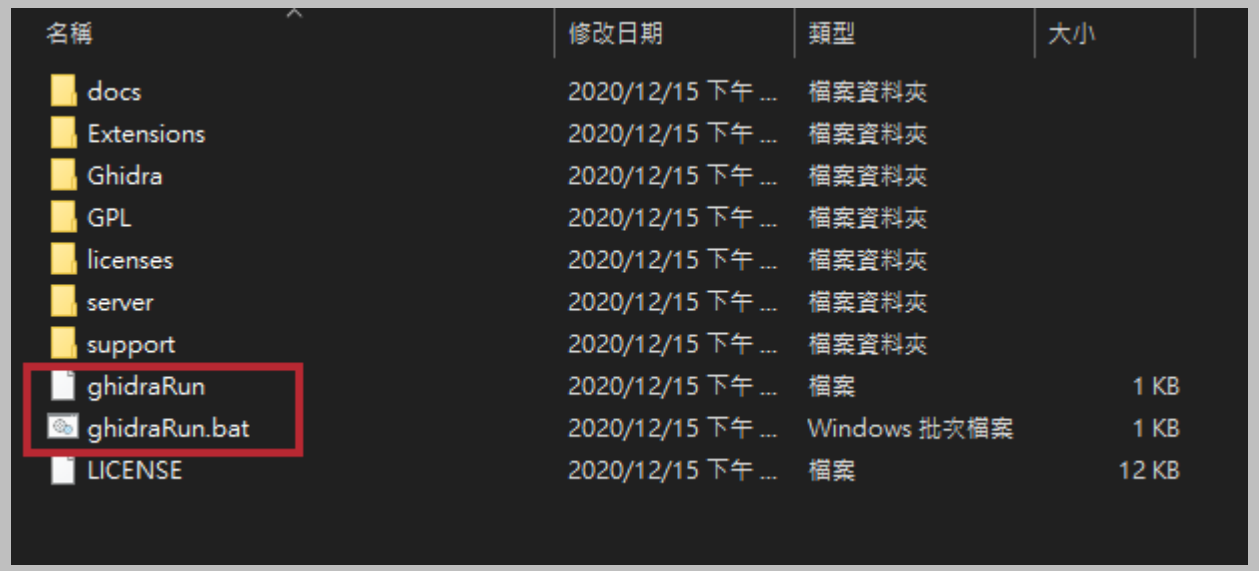
Part1. Ghidra tutorial

Installing JDK

- Since Ghidra is written in JAVA, we need to download and install JDK. https://adoptopenjdk.net/releases.html?variant=openjdk11&jvmVariant=hotspot
- Require an Oracal account.

Ghidra GUI

- Download Ghidra from https://ghidra-sre.org/ and extract the file.



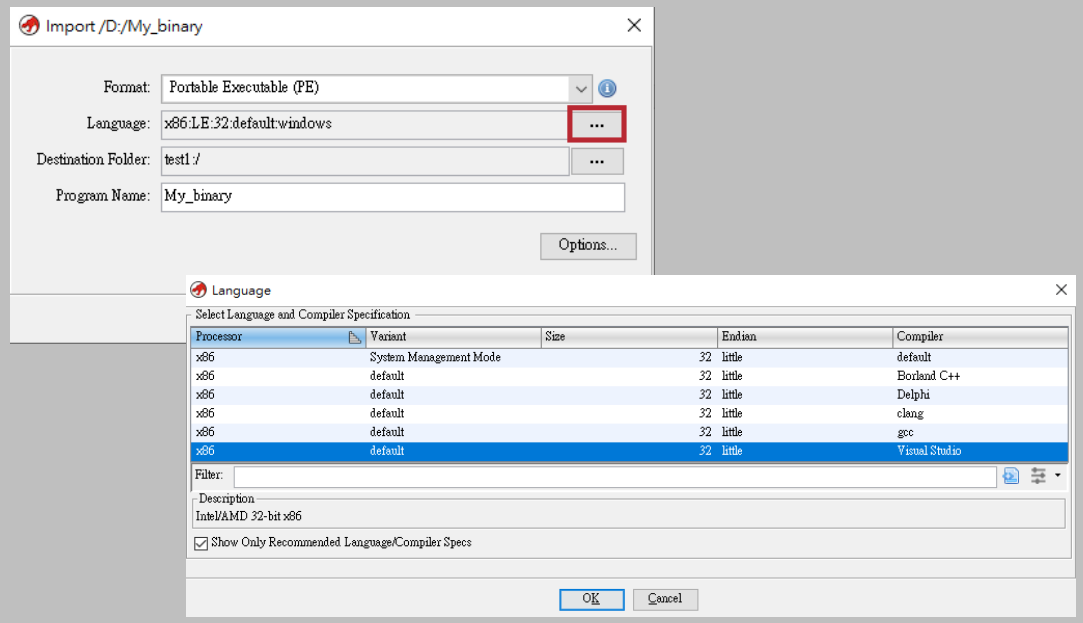
- On windows system, Run ghidraRun.bat to launch GUI.

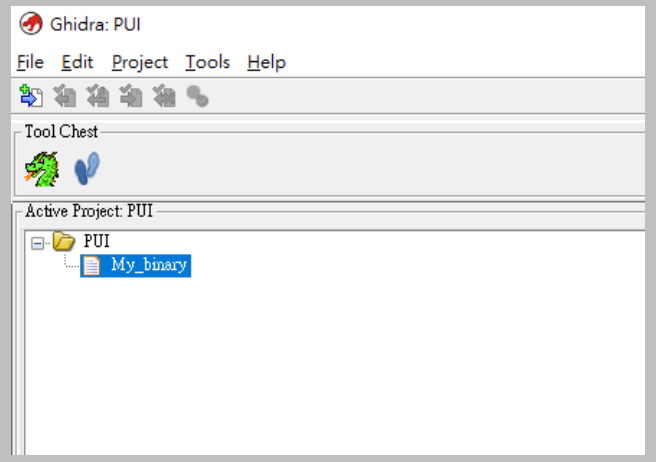
Create a new project

- [File]/[New Project]
- Non-Shared Project
- Enter project name
- [Finish]

Import binary file

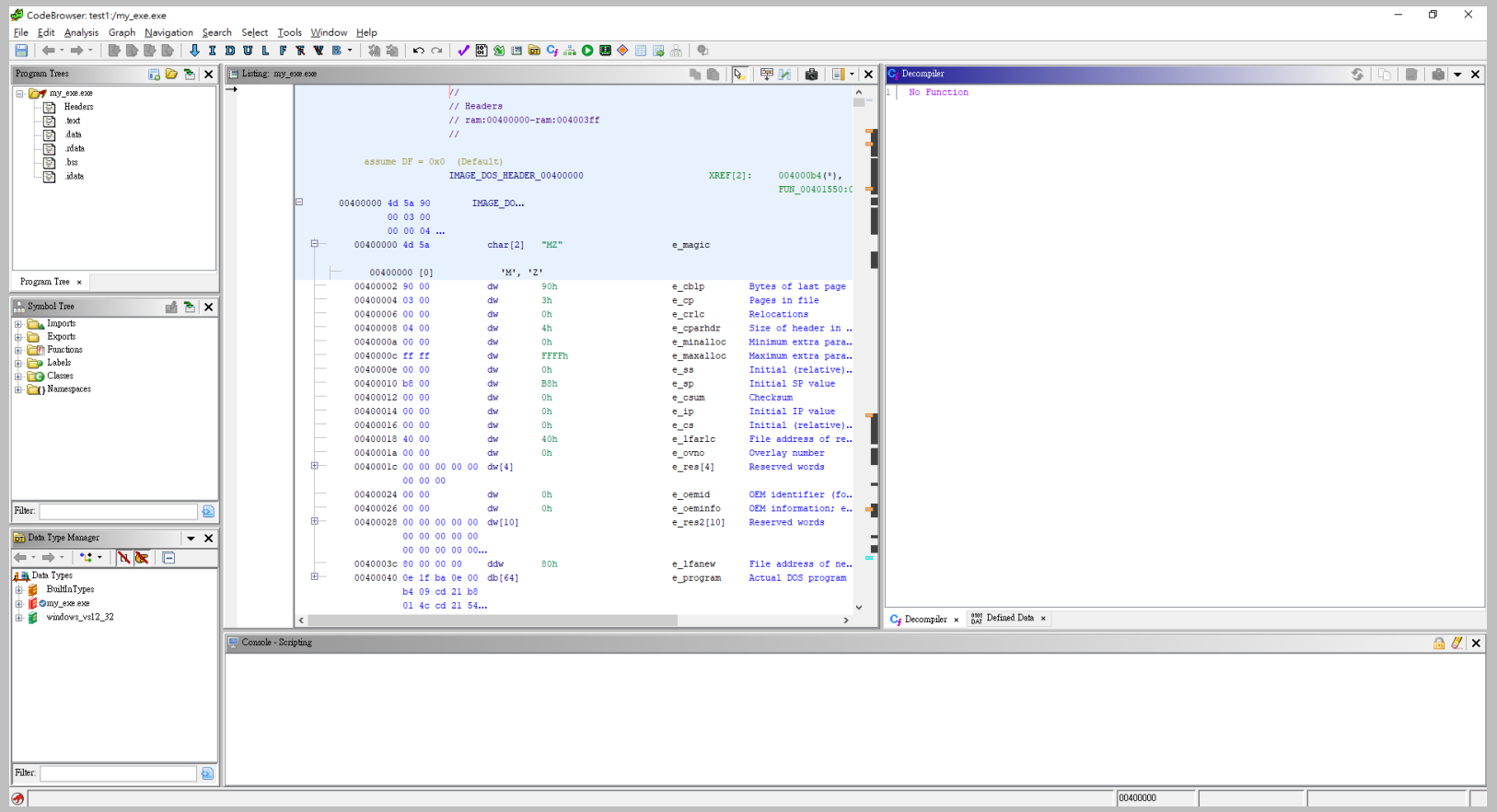
- [File]/[Import file]
- Select the file you want to analysis.
- You can choose the language of the file, or let Ghidra do it itself.
- Press [OK].
- Double click the file to analyze.
- On [Analyze] window, click [Yes] and [Analyze] .





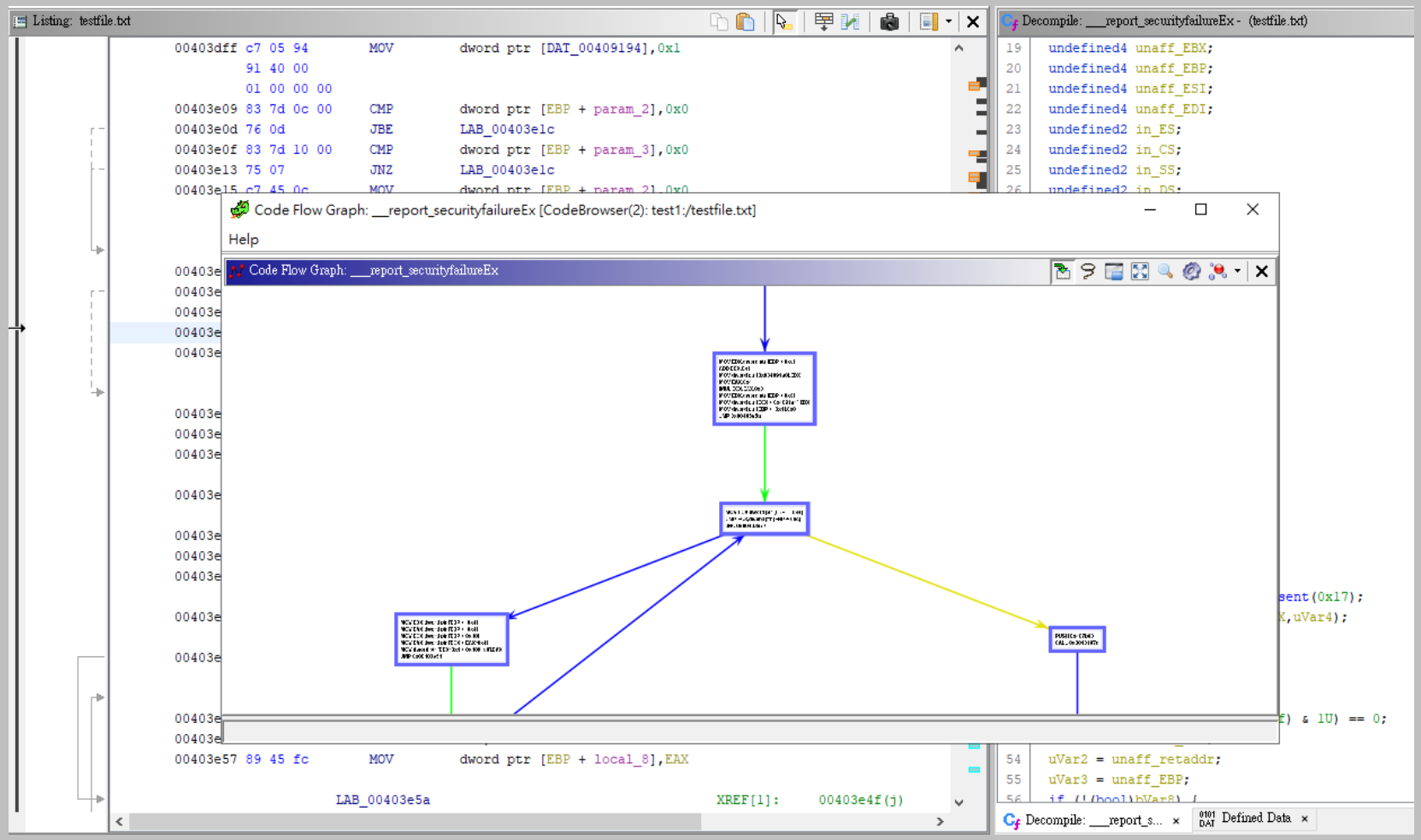
Code browser

- After the analyze is done, you can see the binary disassembly listing and other windows.
- The “MZ” indicate the binary file is a DOS executable format.



Flow graph

- [Graph]/[Block Flow] to see flow graph of current function.

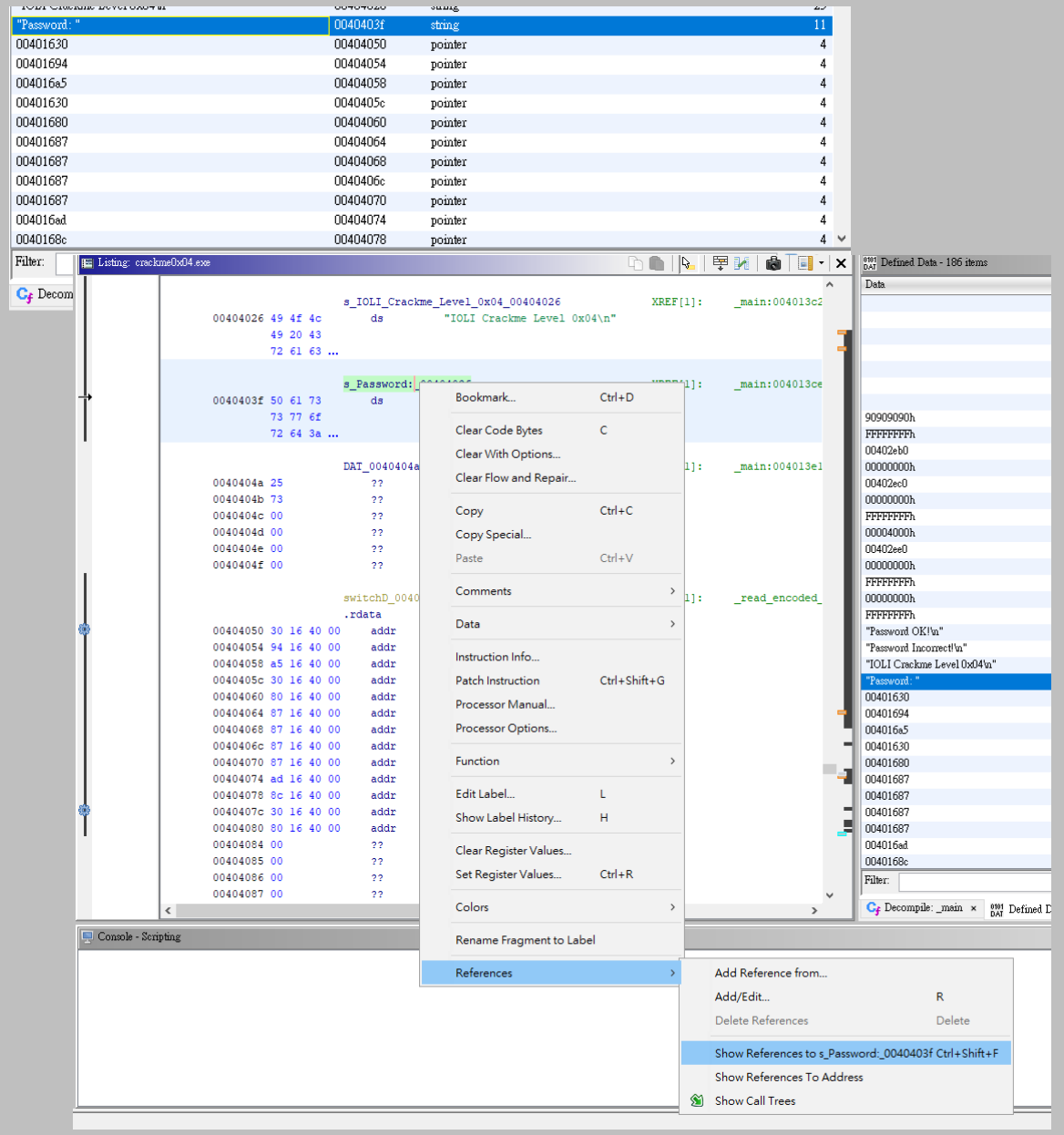


Example

- Here’s an example on cracking IOLI crackme0x04
- https://github.com/Maijin/radare2-workshop-2015/tree/master/IOLI-crackme

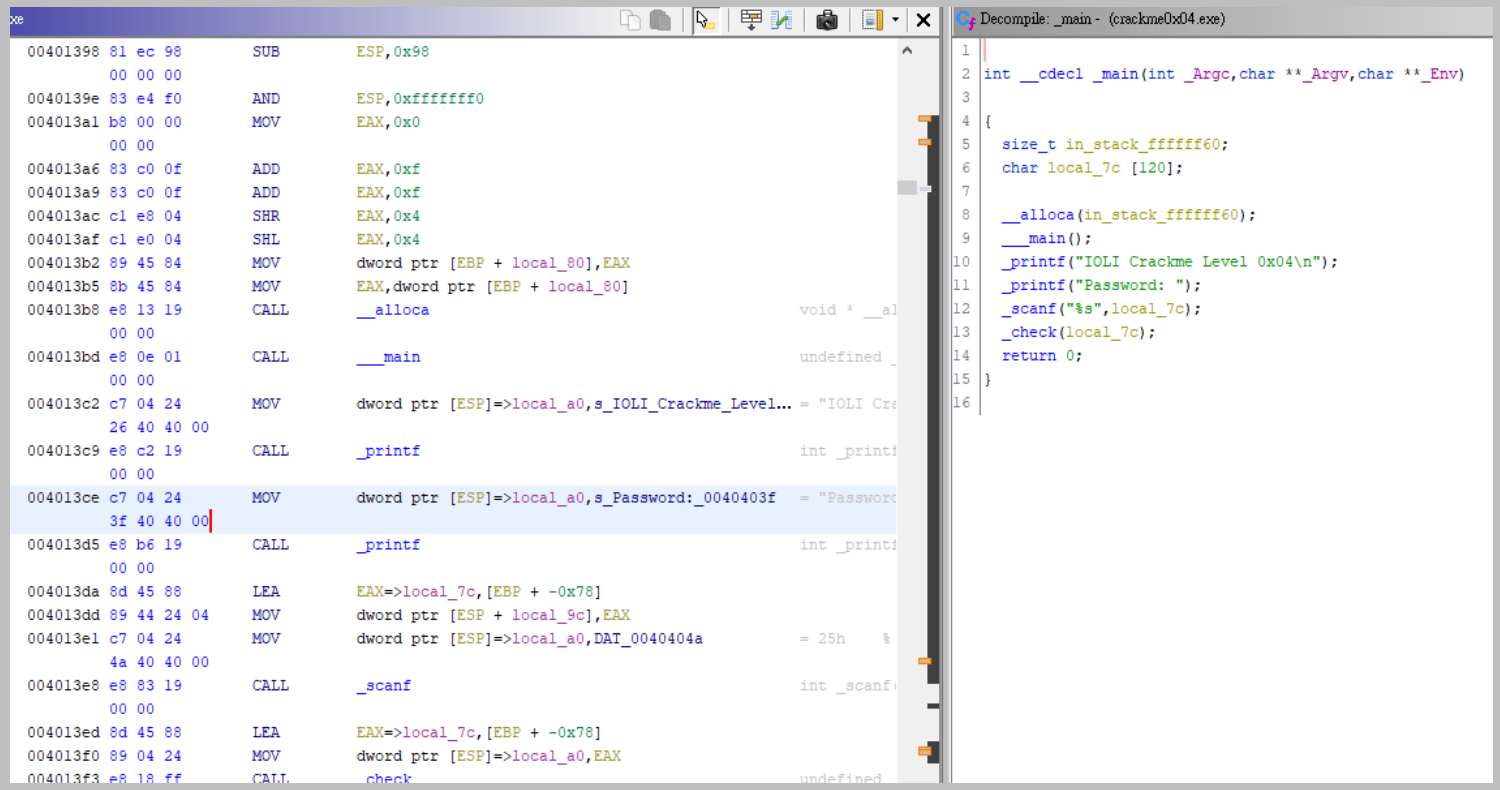
Example

- On Defined Data window, we found a string “Password: ” which is interesting, double click to find the location of the string.
- Richt click/[References]/[Show References]



Example

- We can find the instruction that read the string, and the Decompile window showing the function in C.
- The program reads string input by scanf(), then call _check().
- Double click _ckeck() function, let's see what it does.



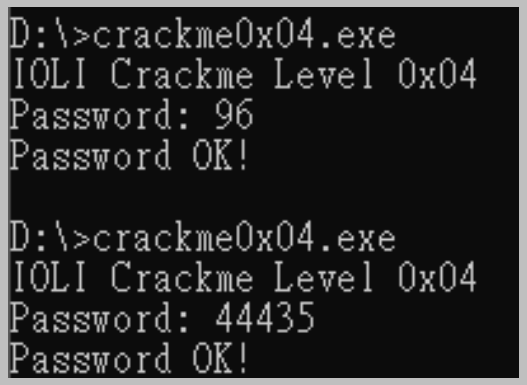
_check()

- in each loop, strlen() function get the length of our input, and call sscanf() function to read each character, convert it to decimal and sum it together. If the sum equals to 0xf(15 in decimal), it break the while loop and print the victory message.

```
void __cdecl _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);
}
```

Example

- Simply input strings which the sum of first N bit is 15 to pass the test.



Part2. OLLVM

- Ollvm wiki: https://github.com/obfuscator-llvm/obfuscator/wiki

Install OLLVM

- Run the following command

```
$ git clone -b llvm-4.0 https://github.com/obfuscator-llvm/obfuscator.git
$ mkdir build
$ cd build
$ cmake -DCMAKE_BUILD_TYPE=Release -DLLVM_INCLUDE_TESTS=OFF ../obfuscator/
$ make
```

Usage

- There are three options you can use while compiling your code.
  - -mllvm –sub for instructions substitution
  - -mllvm –bcf for bogus control flow
  - -mllvm –fla for control flow flattening
- Just pass the flag to compiler like:

  $ ./build/bin/clang source.c -o test -mllvm -sub -mllvm -fla

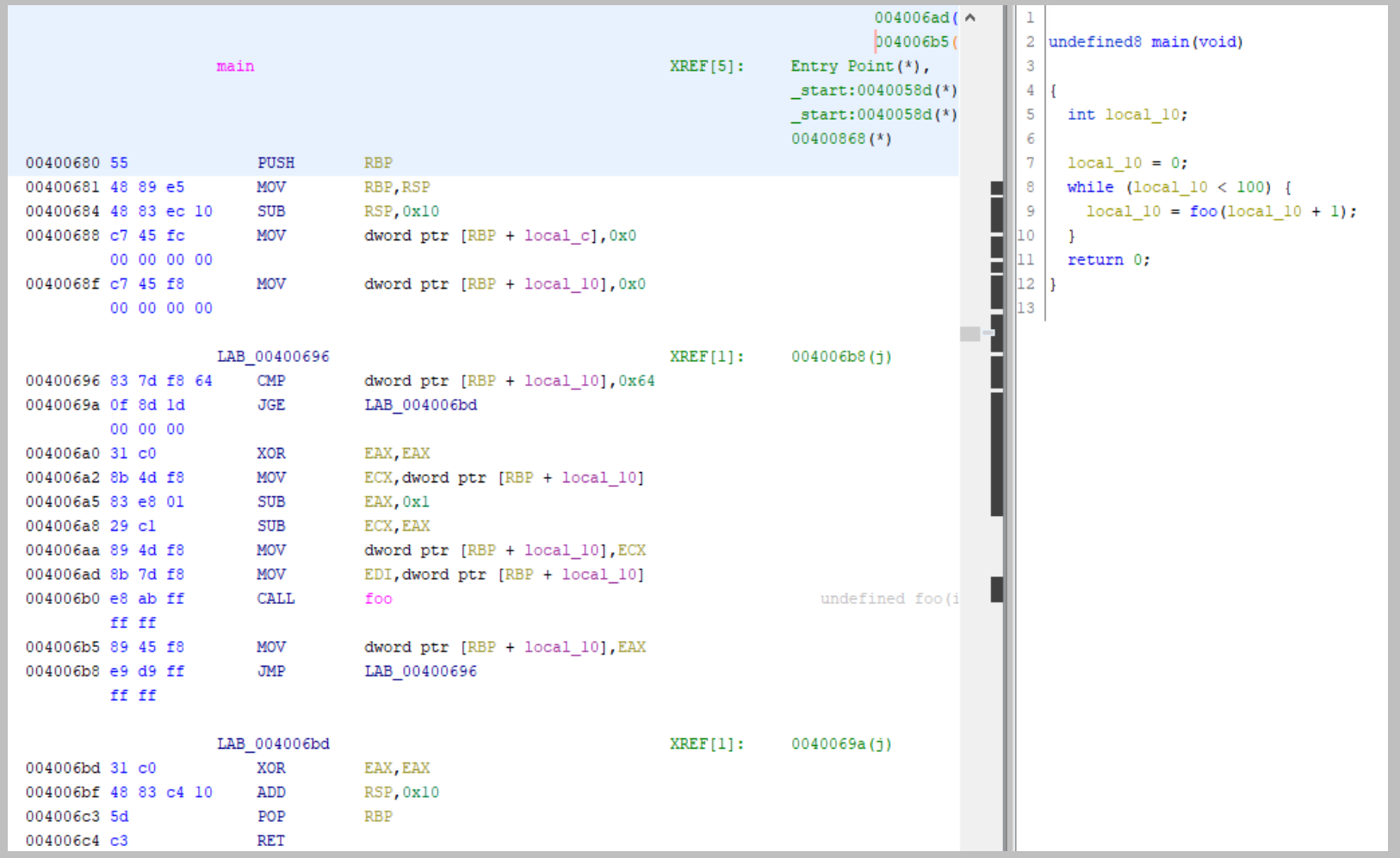
Example

- Now let's take a look at how “–sub” flag affect our code.
- Take this simple program for example

```
#include <iostream>

using namespace std;

int foo(int cnt)
{
    return cnt + 1;
}

int main()
{
    int cnt = 0;
    while(cnt < 100)
    {
        cnt += 1;
        cnt = foo(cnt);
    }

    return 0;
}
```

Decompile

- Here's the decompile result, it seem that there's nothing wrong with it, let's take a closer look at the assembly code.



```
                                                                    004006ad(
                                                                    004006b5(
                main                                    XREF[5]:    Entry Point(*),
                                                                    _start:0040058d(*)
                                                                    _start:0040058d(*)
                                                                    00400868(*)
00400680 55              PUSH        RBP
00400681 48 89 e5        MOV         RBP,RSP
00400684 48 83 ec 10     SUB         RSP,0x10
00400688 c7 45 fc        MOV         dword ptr [RBP + local_c],0x0
         00 00 00 00
0040068f c7 45 f8        MOV         dword ptr [RBP + local_10],0x0
         00 00 00 00

             LAB_00400696                               XREF[1]:    004006b8(j)
00400696 83 7d f8 64     CMP         dword ptr [RBP + local_10],0x64
0040069a 0f 8d 1d        JGE         LAB_004006bd
         00 00 00
004006a0 31 c0           XOR         EAX,EAX
004006a2 8b 4d f8        MOV         ECX,dword ptr [RBP + local_10]
004006a5 83 e8 01        SUB         EAX,0x1
004006a8 29 c1           SUB         ECX,EAX
004006aa 89 4d f8        MOV         dword ptr [RBP + local_10],ECX
004006ad 8b 7d f8        MOV         EDI,dword ptr [RBP + local_10]
004006b0 e8 ab ff        CALL        foo                                 undefined foo(i
         ff ff
004006b5 89 45 f8        MOV         dword ptr [RBP + local_10],EAX
004006b8 e9 d9 ff        JMP         LAB_00400696
         ff ff

             LAB_004006bd                               XREF[1]:    0040069a(j)
004006bd 31 c0           XOR         EAX,EAX
004006bf 48 83 c4 10     ADD         RSP,0x10
004006c3 5d              POP         RBP
004006c4 c3              RET
```

```
1
2  undefined8 main(void)
3
4  {
5    int local_10;
6
7    local_10 = 0;
8    while (local_10 < 100) {
9      local_10 = foo(local_10 + 1);
10   }
11   return 0;
12 }
13
```

Without -sub flag vs With –sub flag

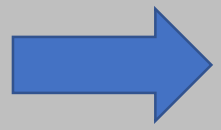
```
                    main                                            XREF[5]:     Entry Point(
                                                                                 _start:00400
                                                                                 _start:00400
                                                                                 00400868(*)
00400680 55              PUSH        RBP
00400681 48 89 e5        MOV         RBP,RSP
00400684 48 83 ec 10     SUB         RSP,0x10
00400688 c7 45 fc        MOV         dword ptr [RBP + local_c],0x0
         00 00 00 00
0040068f c7 45 f8        MOV         dword ptr [RBP + local_10],0x0
         00 00 00 00

                    LAB_00400696                                    XREF[1]:     004006b4(j)
00400696 83 7d f8 64     CMP         dword ptr [RBP + local_10],0x64
0040069a 0f 8d 19        JGE         LAB_004006b9
         00 00 00
004006a0 8b 45 f8        MOV         EAX,dword ptr [RBP + local_10]
004006a3 83 c0 01        ADD         EAX,0x1
004006a6 89 45 f8        MOV         dword ptr [RBP + local_10],EAX
004006a9 8b 7d f8        MOV         EDI,dword ptr [RBP + local_10]
004006ac e8 af ff        CALL        foo                                          undefined
         ff ff
004006b1 89 45 f8        MOV         dword ptr [RBP + local_10],EAX
004006b4 e9 dd ff        JMP         LAB_00400696
         ff ff

                    LAB_004006b9                                    XREF[1]:     0040069a(j)
004006b9 31 c0           XOR         EAX,EAX
004006bb 48 83 c4 10     ADD         RSP,0x10
004006bf 5d              POP         RBP
004006c0 c3              RET
```

```
                    main                                            XREF[5]:     Entry Point(
                                                                                 _start:00400
                                                                                 _start:00400
                                                                                 00400868(*)
00400680 55              PUSH        RBP
00400681 48 89 e5        MOV         RBP,RSP
00400684 48 83 ec 10     SUB         RSP,0x10
00400688 c7 45 fc        MOV         dword ptr [RBP + local_c],0x0
         00 00 00 00
0040068f c7 45 f8        MOV         dword ptr [RBP + local_10],0x0
         00 00 00 00

                    LAB_00400696                                    XREF[1]:     004006b8(j)
00400696 83 7d f8 64     CMP         dword ptr [RBP + local_10],0x64
0040069a 0f 8d 1d        JGE         LAB_004006bd
         00 00 00
004006a0 31 c0           XOR         EAX,EAX
004006a2 8b 4d f8        MOV         ECX,dword ptr [RBP + local_10]
004006a5 83 e8 01        SUB         EAX,0x1
004006a8 29 c1           SUB         ECX,EAX
004006aa 89 4d f8        MOV         dword ptr [RBP + local_10],ECX
004006ad 8b 7d f8        MOV         EDI,dword ptr [RBP + local_10]
004006b0 e8 ab ff        CALL        foo                                          undefined
         ff ff
004006b5 89 45 f8        MOV         dword ptr [RBP + local_10],EAX
004006b8 e9 d9 ff        JMP         LAB_00400696
         ff ff

                    LAB_004006bd                                    XREF[1]:     0040069a(j)
004006bd 31 c0           XOR         EAX,EAX
004006bf 48 83 c4 10     ADD         RSP,0x10
004006c3 5d              POP         RBP
004006c4 c3              RET
```

Example

- We can observe that there's a slightly difference between two assembly code, the add instruction became two sub instruction.
  a = b – (-c)

```
EAX = local_10
EAX = EAX + 1
local_10 = EAX
```



```
EAX = 0
ECX = local_10
EAX = EAX - 1
ECX = ECX - EAX
Local_10 = ECX
```

- Although we can't see the difference after decompile, but the obfuscator do did its job.