

Lab - Reverse Engineering crackme0x05 Using Ghidra

Overview

In the short lab, we will continue with learning reverse engineering by decompiling a simple executable labeled crackme0x05.exe.

A crackme is a small program designed to test a programmer's reverse engineering skills. Crackme files are created by other programmers as a legal way to crack software since no intellectual property is being infringed upon.

The crackme0x05.exe is a simple program that needs a password to open. Our file could be any executable that requires a password to launch.

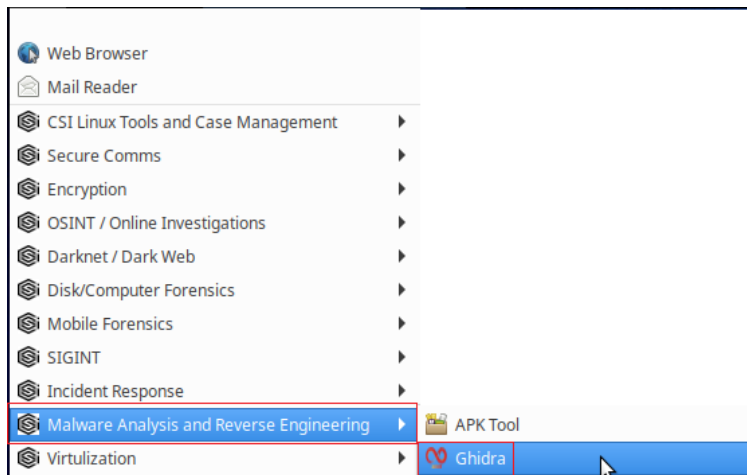
Lab Requirements

- One installation of VirtualBox with the extension pack.
- One virtual install of the latest version of CSI Linux.
- One virtual install of Windows 7
- VirtualBox network adapter set to NAT network.

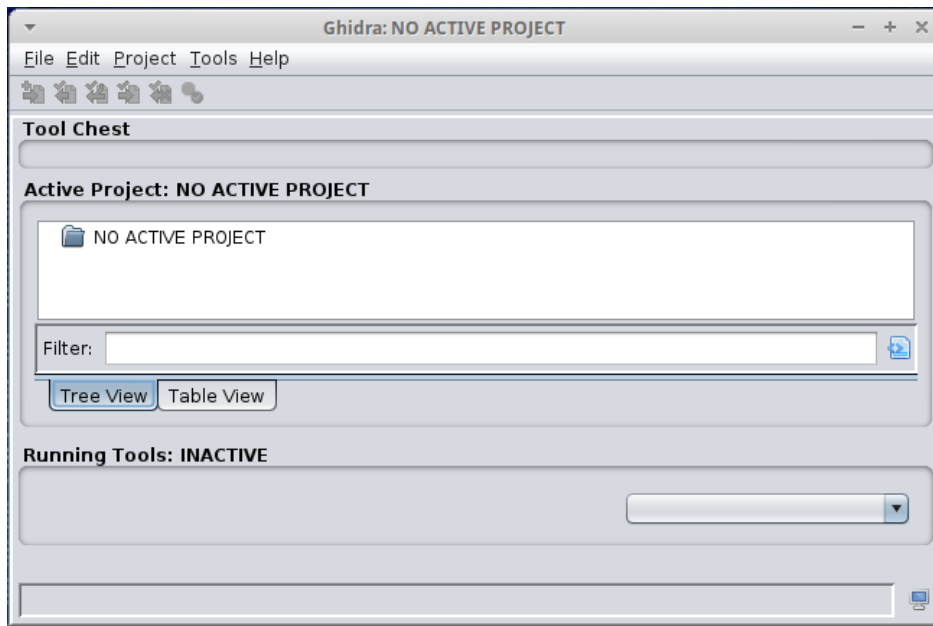
Launch Ghidra

Installing Ghidra onto CSI Linux was covered in a previous [lab](#). This lab assumes you have completed this step.

From the application menu in CSI Linux, scroll down until you come to Malware Analysis and Reverse Engineering. Then, expand the context menu, and from the option, click Ghidra to start the application.

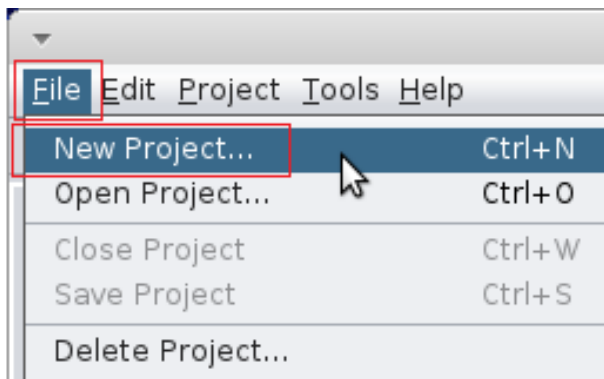


After a short pause, the Active Project screen opens.

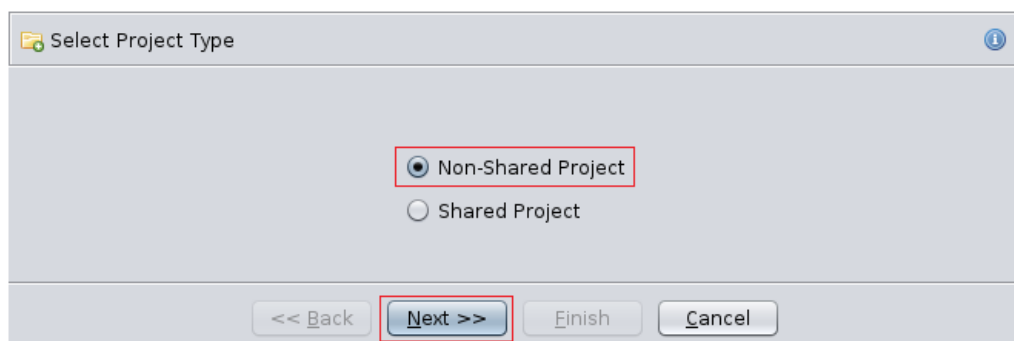


Creating a new project

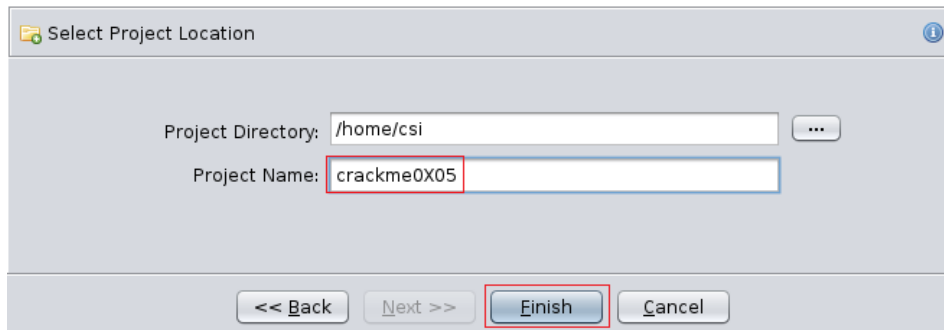
We next need to create a new project. From the Ghidra taskbar, go to File>New Project.



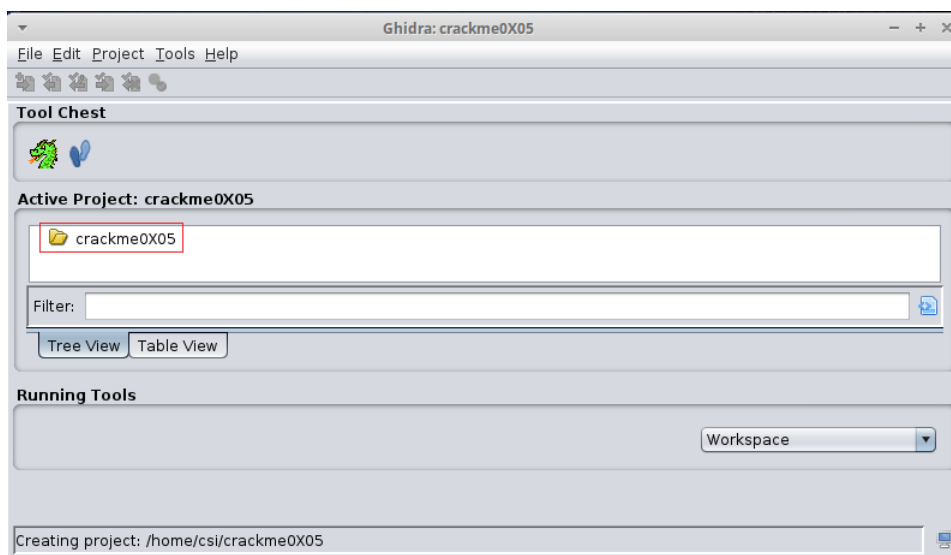
The **Select Project Type** screen opens. Ghidra can be used as a collaboration tool or as a stand-alone, non-shared project. We will be working individually on a project, so click "Non-Shared Project." Then Click Next.



On the next screen, we give our new project a user-friendly name and select a location to save our work. In this example, I have named my first project **crackme0x05**, and I will accept the default location for saving my work. You are free to name your project as you see fit and save it where you want. Click Finish.



Back at the Active Project screen, we see a folder for our new project.



Download crackme0x05.exe.

From your CSI Linux, open your default browser and copy and paste the following URL into the address bar.

<https://github.com/Maijin/radare2-workshop-2015/tree/master/IOLI-crackme/bin-win32>

We will be analyzing the crackme0x05.exe file. From the list of crackme files, click on the file crackme0x05.exe.

master radare2-workshop-2015 / IOLI-crackme / bin-win32 /

Maijin First commit	
..	
crackme0x00.exe	First commit
crackme0x01.exe	First commit
crackme0x02.exe	First commit
crackme0x03.exe	First commit
crackme0x04.exe	First commit
crackme0x05.exe	First commit
crackme0x06.exe	First commit
crackme0x07.exe	First commit
crackme0x08.exe	First commit
crackme0x09.exe	First commit

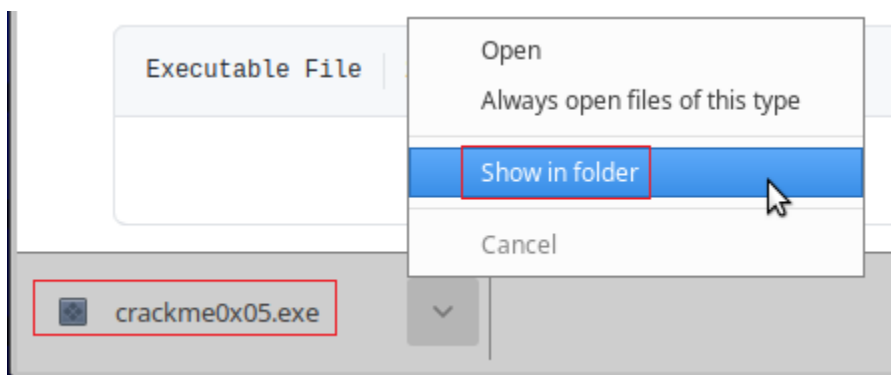
On the next page, click on the Download button.

master radare2-workshop-2015 / IOLI-crackme / bin-win32 / crackme0x05.exe Go to file ...

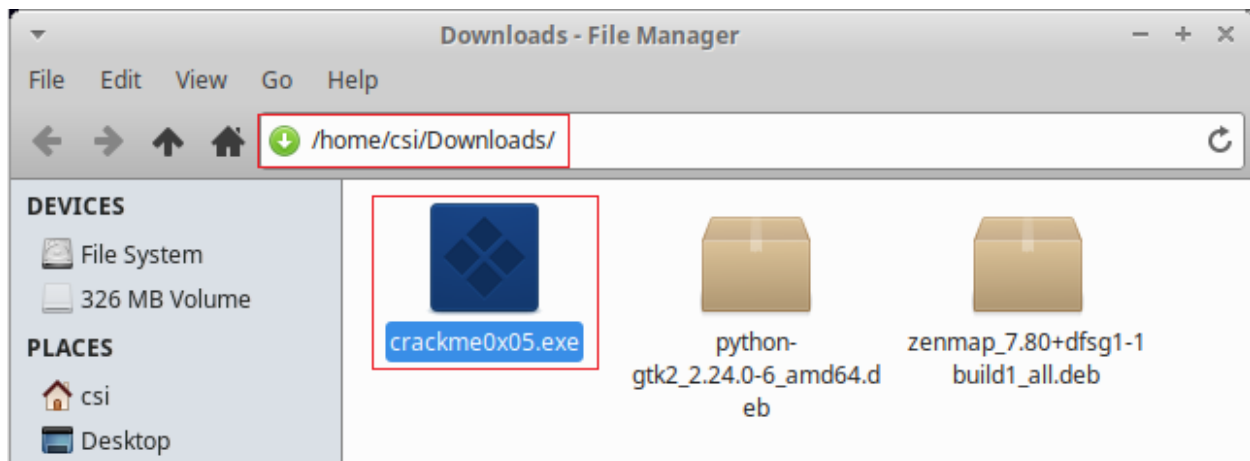
Maijin First commit

Executable File 24.1 KB Download

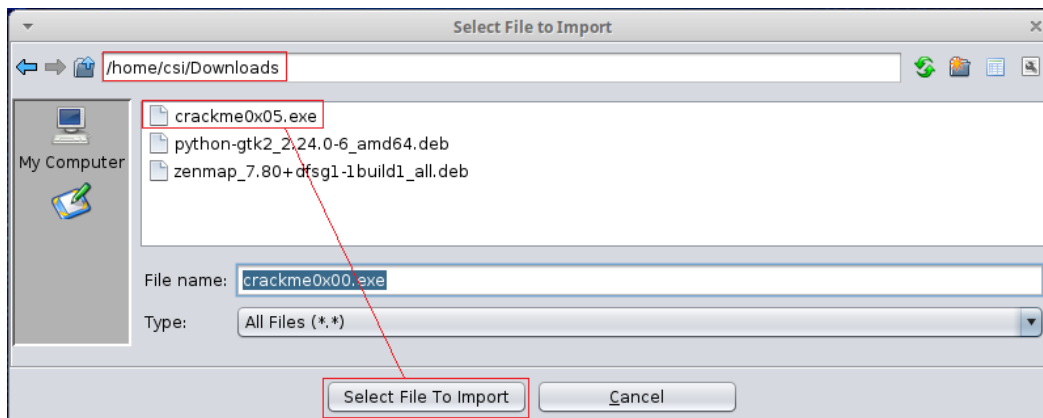
Once the file downloads, click the down arrow, and from the context menu, select **Show in folder**.



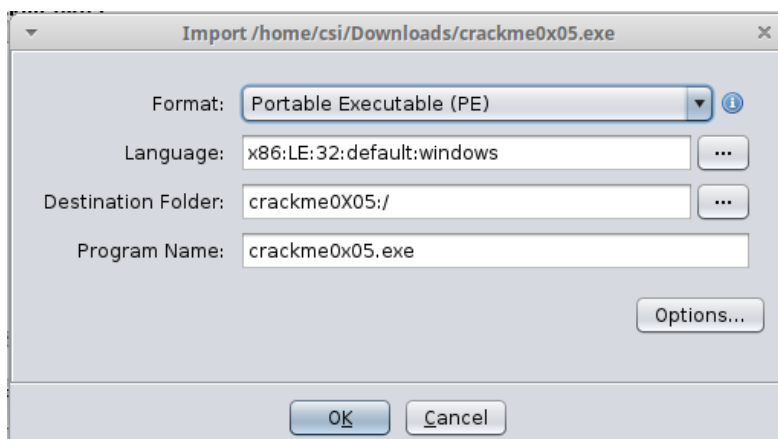
Take note that the file is in your Downloads Directory.



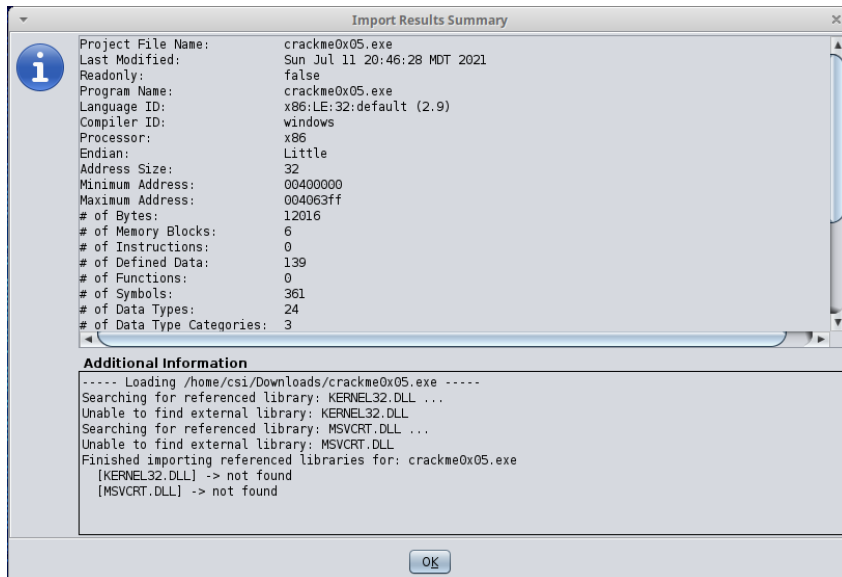
Close out your browser, the file manager, and return to Ghidra. There are two ways to import a file into Ghidra. You can click on File> Import File or open your Downloads directory and drag and drop the crackme0x05.exe into the project window.



When you begin the import process, Ghidra will respond with the following information. Click OK.



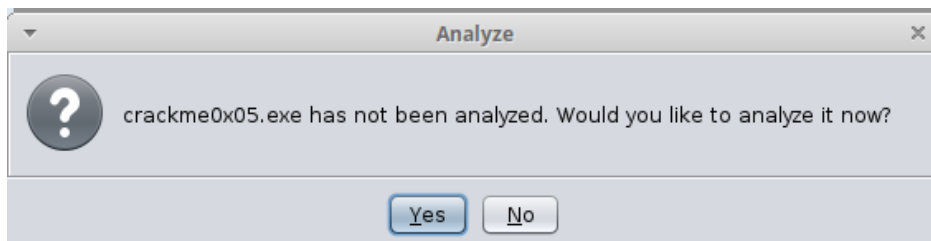
Once the file is imported, another screen loads with a summary of the import. Click OK to close this screen.



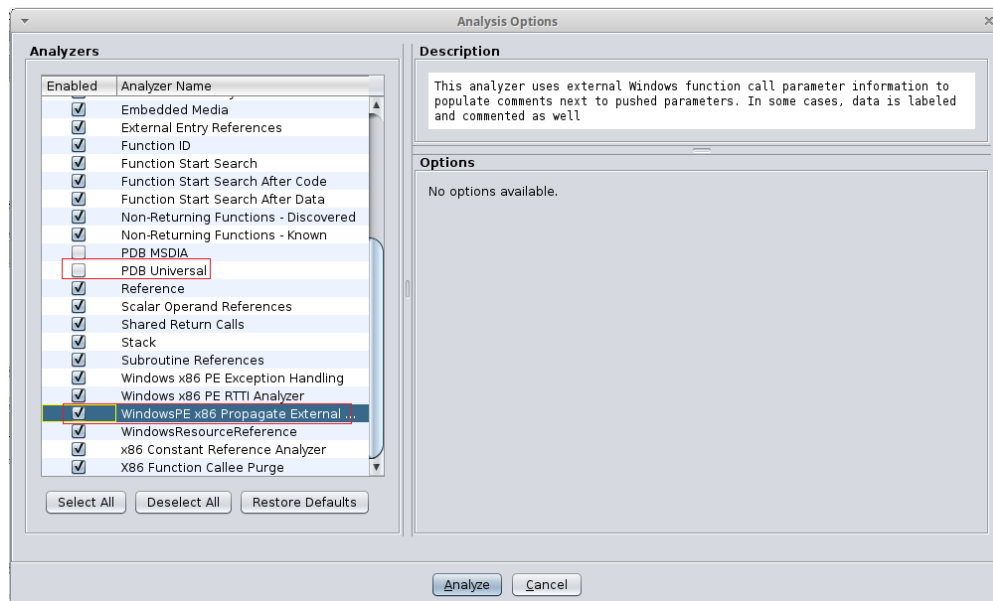
Begin the file Analysis

X2 click the imported crackme0x05.exe file inside your Ghidra Active Projects window to begin the file analysis.

When asked if you would like to analyze now, press the yes button.

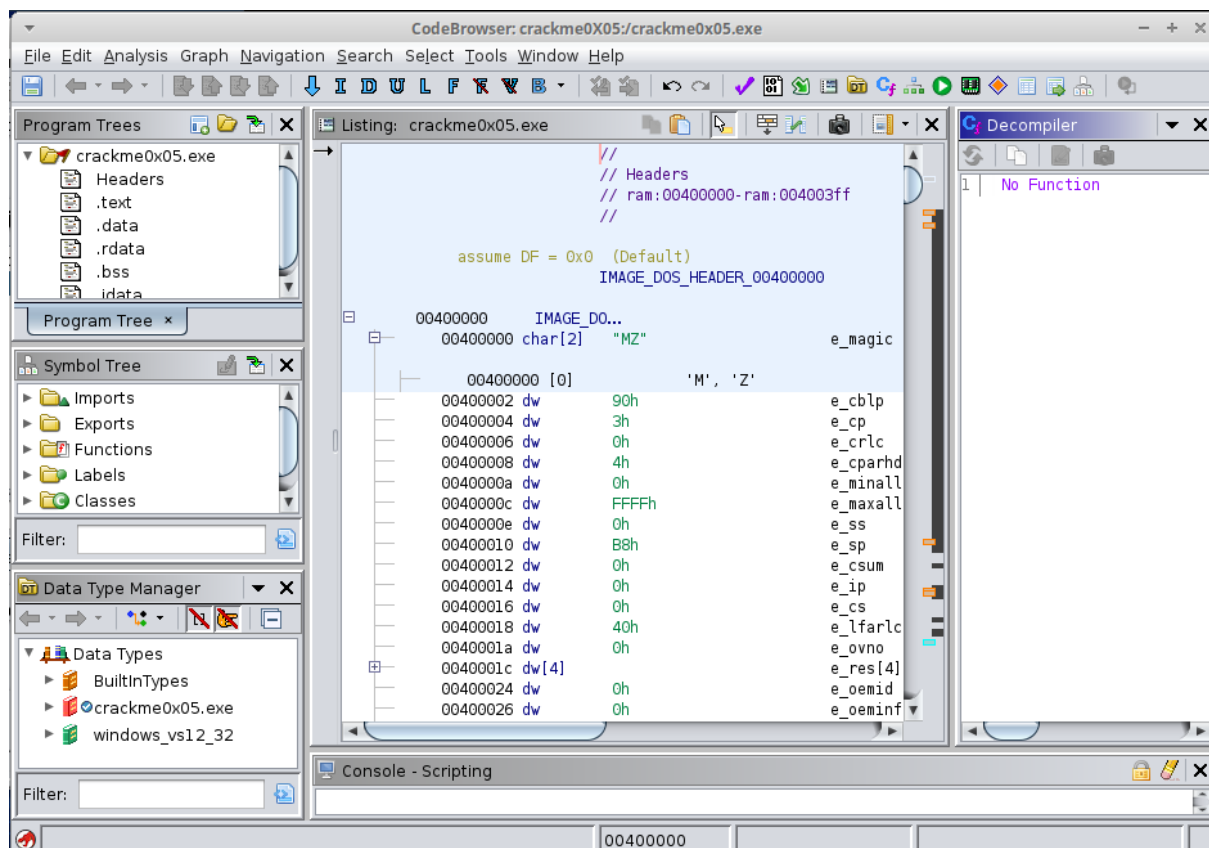


The Analysis Options screen opens. Uncheck the box for the PDB Universal scanner; we will not need it. Check the box to enable the WindowsPE x86 Propagate External scanner. Press the analyze button.



Once the analysis is complete, we are presented with the Ghidra Code Browser default view, showing us the disabled code.

The file assembly code is broken down into the different windows shown in the code browser.



Where do we start?

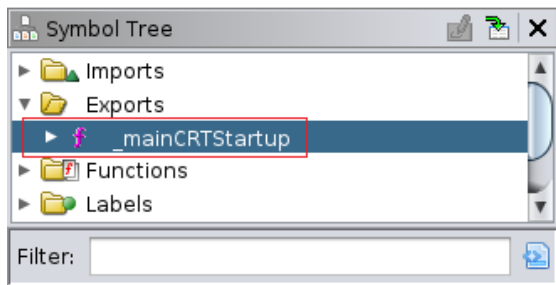
If you were a programmer, you would know the answer to this question. *Every C program has a primary (main) function that must be named main.*

The **main function** serves as the starting point for program execution. It usually controls program execution by directing the calls to other **functions** in the program. A program usually stops executing at the end of **main**, although it can terminate at different points in the program for a variety of reasons.

In our analysis of this executable, we are trying to determine how the program works.

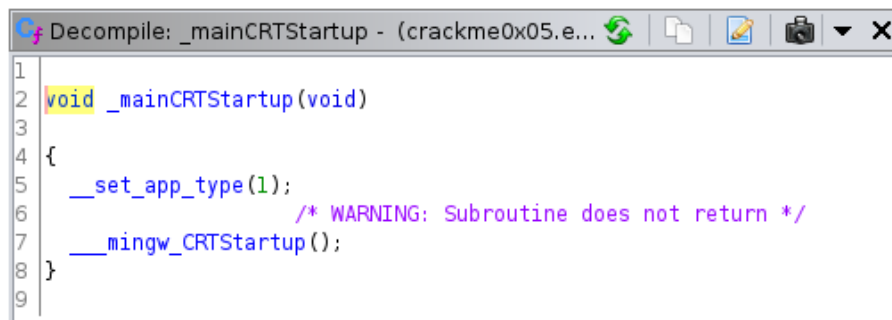
Looking for the main function is where we should begin.

We can expand the "exports" sub-tab in the "symbol tree" view on the right. When we do, we see an option for "_mainCRTStartup":



When we double click on this, it populates the "decompile" window with the following:

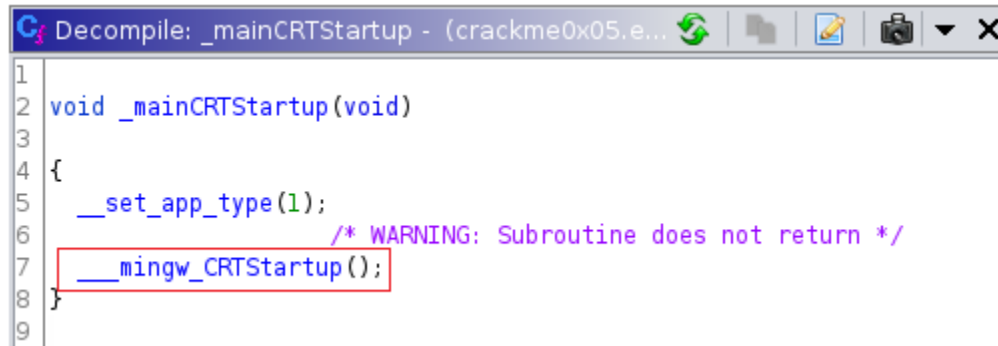
When we double click on this function, it populates the "decompile" window with the following:



```
1
2 void _mainCRTStartup(void)
3
4 {
5     __set_app_type(1);
6     /* WARNING: Subroutine does not return */
7     __mingw_CRTStartup();
8 }
9
```

The decompiler is one of the most powerful features of Ghidra. It automatically converts binary instructions into high-level C-like code.

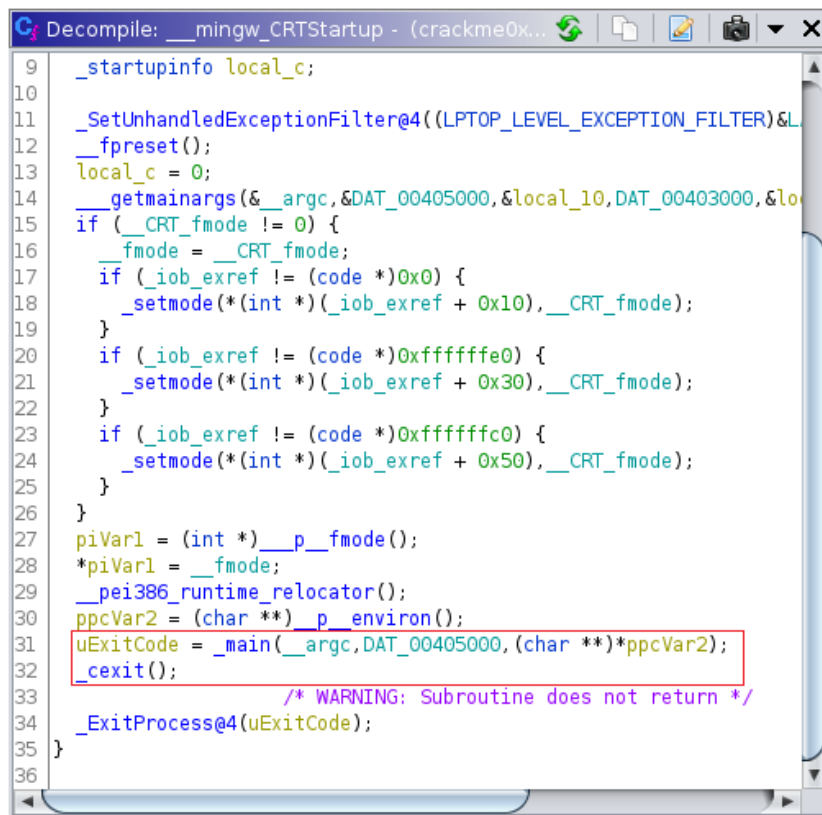
Inside the Decompile window, if we double click "__mingw_CRTStartup"



The image shows a decompiler window titled "Decompile: _mainCRTStartup - (crackme0x05.e...". The code is as follows:

```
1
2 void _mainCRTStartup(void)
3 {
4     __set_app_type(1);
5     /* WARNING: Subroutine does not return */
6     __mingw_CRTStartup();
7 }
8
9
```

we are brought to the following screen, where we can see there is a call to the main function:



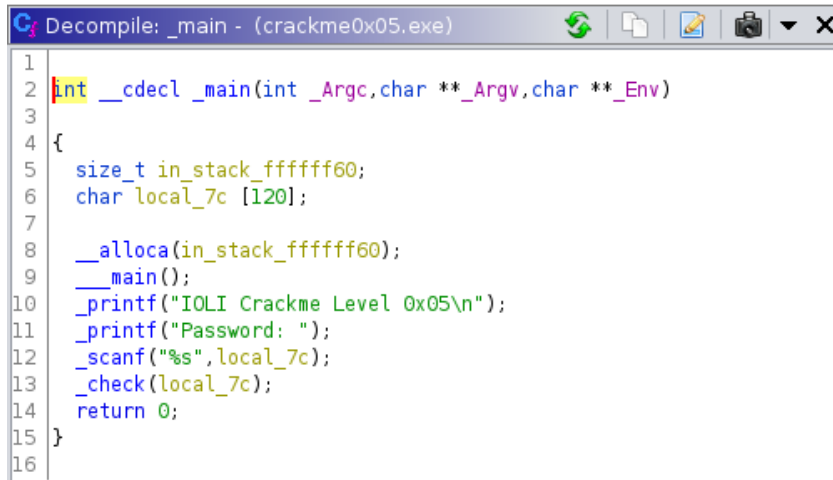
The image shows a decompiler window titled "Decompile: __mingw_CRTStartup - (crackme0x...". The code is as follows:

```
9  _startupinfo local_c;
10
11  _SetUnhandledExceptionFilter@4((LPTOP_LEVEL_EXCEPTION_FILTER)&L
12  __fpreset();
13  local_c = 0;
14  __getmainargs(&__argc,&DAT_00405000,&local_10,DAT_00403000,&lo
15  if (__CRT_fmode != 0) {
16      __fmode = __CRT_fmode;
17      if (_iob_exref != (code *)0x0) {
18          _setmode(*(int *)(_iob_exref + 0x10),__CRT_fmode);
19      }
20      if (_iob_exref != (code *)0xffffffff0) {
21          _setmode(*(int *)(_iob_exref + 0x30),__CRT_fmode);
22      }
23      if (_iob_exref != (code *)0xffffffffc0) {
24          _setmode(*(int *)(_iob_exref + 0x50),__CRT_fmode);
25      }
26  }
27  piVar1 = (int *)__p_fmode();
28  *piVar1 = __fmode;
29  __pei386_runtime_relocator();
30  ppcVar2 = (char *)__p_environ();
31  uExitCode = _main(__argc,DAT_00405000,(char *)__ppcVar2);
32  _cexit();
33  /* WARNING: Subroutine does not return */
34  _ExitProcess@4(uExitCode);
35 }
36
```

We can double click on the "_main" function to view the decompilers recreation of the binary files main function from the original source code:

```
uExitCode = _main(__argc,DAT_00405000,(char *)__ppcVar2);
_cexit();
```

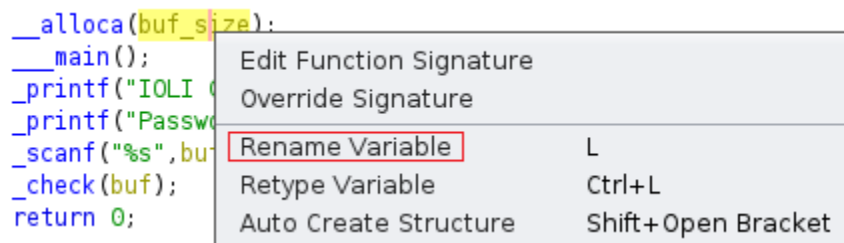
Result



```
1 2 int __cdecl _main(int _Argc, char **_Argv, char **_Env)
3
4 {
5     size_t in_stack_ffffff60;
6     char local_7c [120];
7
8     __alloca(in_stack_ffffff60);
9     __main();
10    _printf("IOLI Crackme Level 0x05\n");
11    _printf("Password: ");
12    _scanf("%s", local_7c);
13    _check(local_7c);
14    return 0;
15 }
16
```

We can start making sense of the program. We can see that we take in a command-line argument (password), read it into a buffer, and pass this buffer to the "_check" program.

Note that we can rename variables in the decompiler by clicking on them from the context menu and selecting **Rename Variable**.

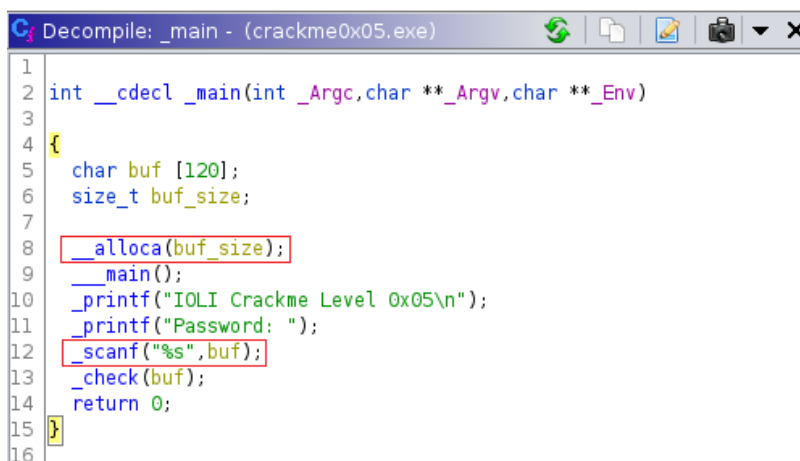


```
__alloca(buf_size);
__main();
_printf("IOLI Crackme Level 0x05\n");
_printf("Password: ");
_scanf("%s", buf_size);
_check(buf_size);
return 0;
```

Edit Function Signature	
Override Signature	
Rename Variable	L
Retype Variable	Ctrl+L
Auto Create Structure	Shift+ Open Bracket

This brings up a text box where we can enter the new name for our variable. We can rename "in_stack_ffffff60" to buf_size and "local_7c" to buf:

After the rename.



```
1 2 int __cdecl _main(int _Argc, char **_Argv, char **_Env)
3
4 {
5     char buf [120];
6     size_t buf_size;
7
8     __alloca(buf_size);
9     __main();
10    _printf("IOLI Crackme Level 0x05\n");
11    _printf("Password: ");
12    _scanf("%s", buf);
13    _check(buf);
14    return 0;
15 }
16
```

Next, we double-click the "_check(buf)" function to see what it is doing. This produces the following view: Again, we can rename our local variables to make more sense:

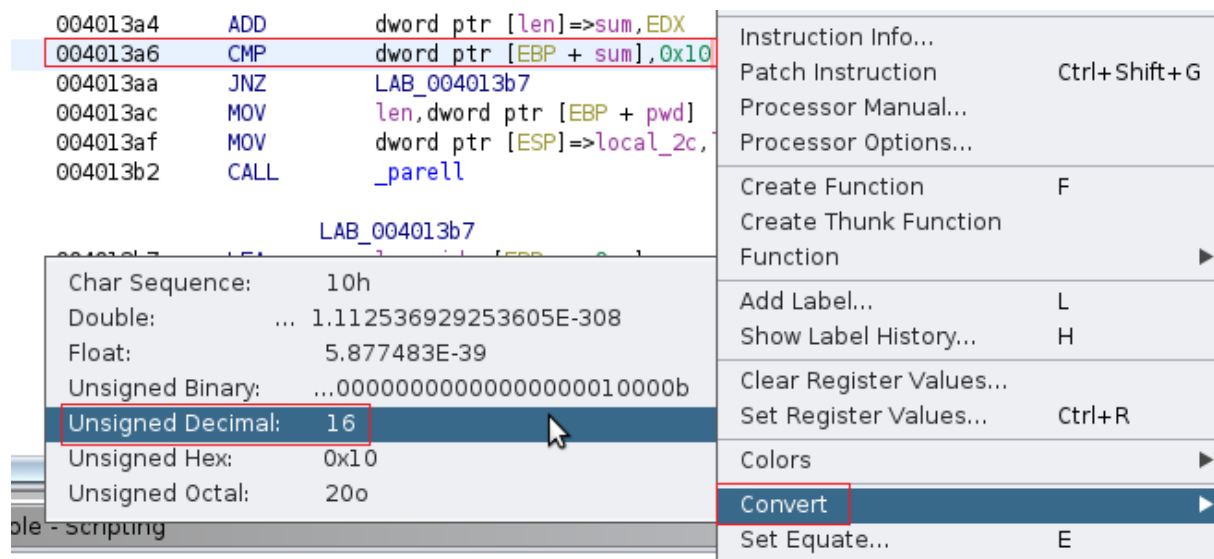
Before

```
1 void __cdecl _check(char *param_1)
2 {
3     size_t sVar1;
4     char local_11;
5     uint local_10;
6     int local_c;
7     int local_8;
8
9     local_c = 0;
10    local_10 = 0;
11    while( true ) {
12        sVar1 = _strlen(param_1);
13        if (sVar1 <= local_10) break;
14        local_11 = param_1[local_10];
15        _sscanf(&local_11, "%d", &local_8);
16        local_c = local_c + local_8;
17        if (local_c == 0x10) {
18            _parell(param_1);
19        }
20        local_10 = local_10 + 1;
21    }
22    _printf("Password Incorrect!\n");
23    return;
24 }
25
26
27
```

After

```
1 void __cdecl _check(char *pwd)
2 {
3     size_t len;
4     char cur_char;
5     uint idx;
6     int sum;
7     int next_value_to_add;
8
9     sum = 0;
10    idx = 0;
11    while( true ) {
12        len = _strlen(pwd);
13        if (len <= idx) break;
14        cur_char = pwd[idx];
15        _sscanf(&cur_char, "%d", &next_value_to_add);
16        sum = sum + next_value_to_add;
17        if (sum == 0x10) {
18            _parell(pwd);
19        }
20        idx = idx + 1;
21    }
22    _printf("Password Incorrect!\n");
23    return;
24 }
25
26
27
```

The "_check" function takes in a password that the user has entered and starts an infinite loop. The goal of the loop is to iterate through each character in the string and store the next character in "cur_char" (line 16). This character then gets converted to an integer by the "_sscanf" call (line 17). The sum of all these integers is stored in the "sum" variable (line 18). On line 19, the function checks if the value of the sum is equal to "0x10". This is 16 in decimal, and we can get the decompiler to represent this as a decimal number by right-clicking on the value in the Listing window, and clicking convert-> unsigned decimal.



It now shows as a decimal number in the decompiler:

```

1  void __cdecl _check(char *pwd)
2  {
3  {
4  {
5  size_t len;
6  char cur_char;
7  uint idx;
8  int sum;
9  int next_value_to_add;
10
11  sum = 0;
12  idx = 0;
13  while( true ) {
14      len = _strlen(pwd);
15      if (len <= idx) break;
16      cur_char = pwd[idx];
17      _sscanf(&cur_char, "%d", &next_value_to_add);
18      sum = sum + next_value_to_add;
19      if (sum == 16) {
20          _parell(pwd);
21      }
22      idx = idx + 1;
23  }
24  _printf("Password Incorrect!\n");
25  return;
26  }
27  }

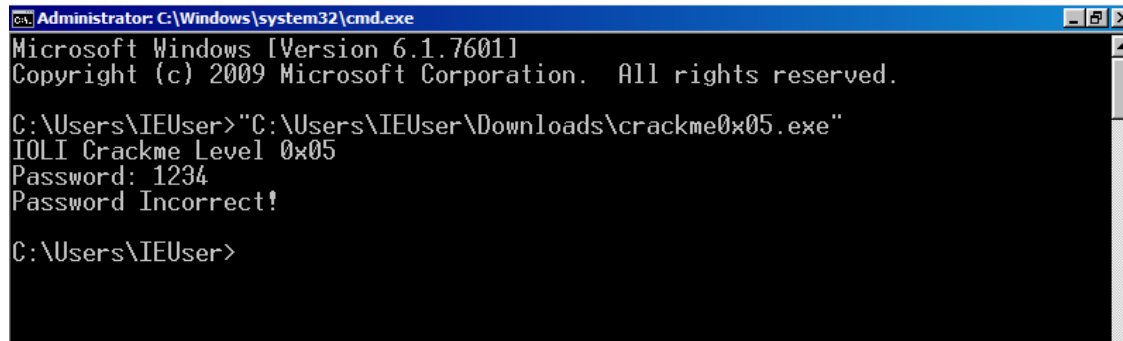
```

This program's function is to verify the password to see whether the characters entered by the user add up or equal the sum to "16" after being converted into integers.

We can verify this by bringing up Command Prompt and running the program:

Examining crackme0x05.exe

You have the option of downloading **crackme0x05.exe** and running the program to examine its operation. In this example, I have downloaded the crackme0x05.exe onto a virtual install of Windows 7. I type in 1234 for the password. The password is incorrect.

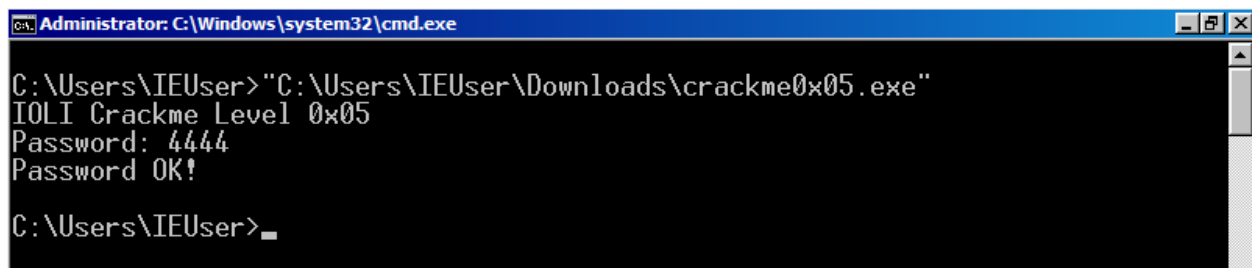


```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\IEUser>"C:\Users\IEUser\Downloads\crackme0x05.exe"
IOLI Crackme Level 0x05
Password: 1234
Password Incorrect!

C:\Users\IEUser>
```

I next type in a combination of numbers that equal the sum of 16. I type in 4444. Bingo!



```
Administrator: C:\Windows\system32\cmd.exe

C:\Users\IEUser>"C:\Users\IEUser\Downloads\crackme0x05.exe"
IOLI Crackme Level 0x05
Password: 4444
Password OK!

C:\Users\IEUser>_
```

Summary –

In this lab, we reversed engineered a simple executable, **crackme0x05.exe**. By examining the main function of the program. We quickly saw the main function checks the password for the correct sum. We also saw how we could rename variables to make them easier to find and track in the assembly code. Going forward will continue to work with other crackme files to improve upon our reverse engineering skills.