

GHIDRA

Ghidra is a software reverse engineering (SRE) framework developed by NSA's [Research Directorate](#) for NSA's [cybersecurity mission](#). It helps analyze malicious code and malware like viruses, and can give cybersecurity professionals a better understanding of potential vulnerabilities in their networks and systems.

Key features of Ghidra:

includes a suite of software analysis tools for analyzing compiled code on a variety of platforms including Windows, Mac OS, and Linux

capabilities include disassembly, assembly, decompilation, graphing and scripting, and hundreds of other features

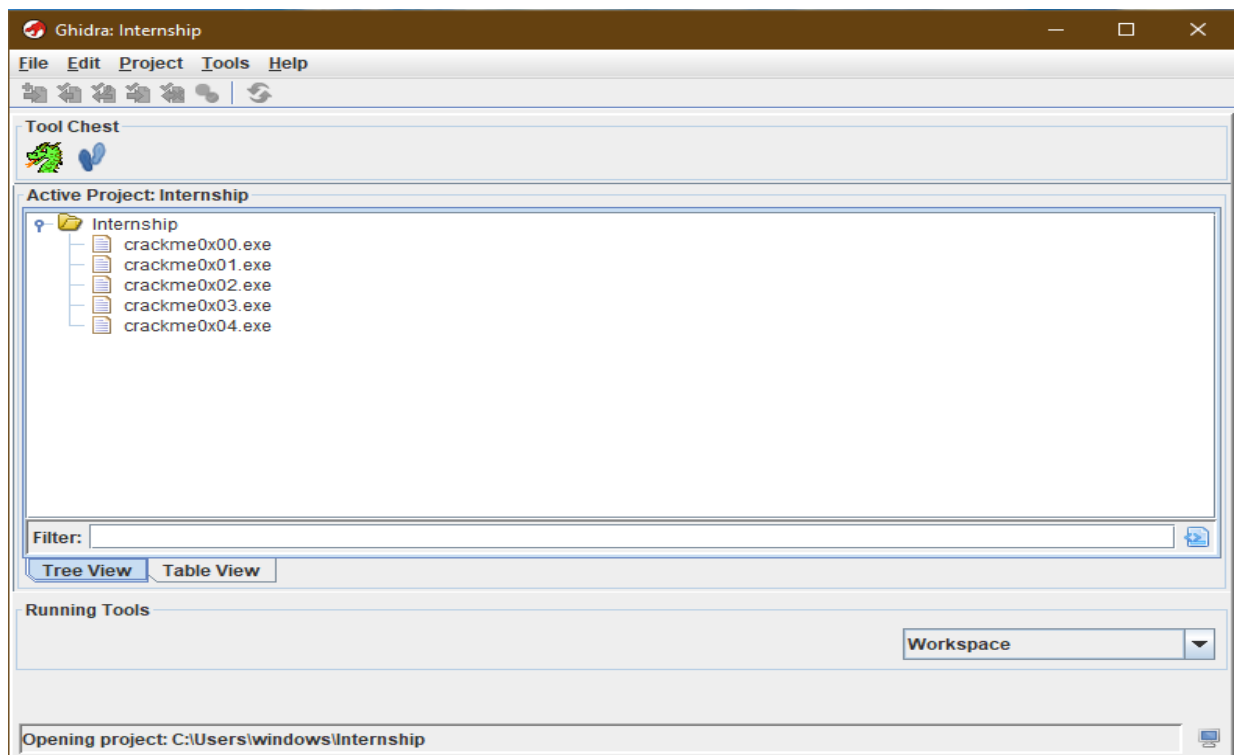
supports a wide variety of processor instruction sets and executable formats and can be run in both user-interactive and automated modes.

users may develop their own Ghidra plug-in components and/or scripts using the exposed API

Ghidra Project Window:

When Ghidra first starts, the [Ghidra Project Window](#) will appear. Ghidra is a project-oriented application and, consequently, all work must be performed in the context of a project. Therefore, the first thing to do is to [create](#) a project or [open](#) an existing project.

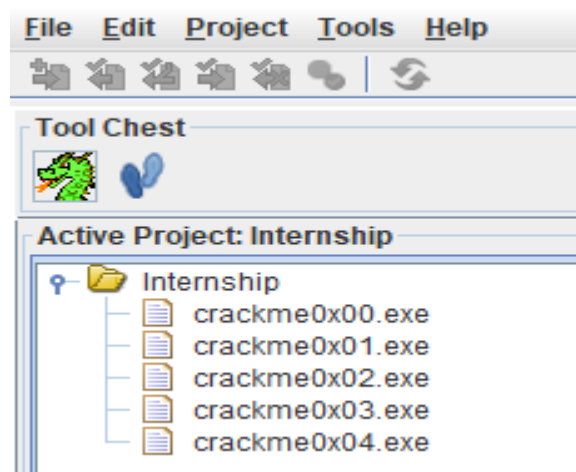
Now, you can drag your .exe files (called as a Ghidra Program) to be analysed inside this project.



A Ghidra program is an executable unit of software or some group of data. It can be viewed and analyzed within a Ghidra tool. A Ghidra program is stored in a project folder. An assembly language is associated with a program at the time it is created. The language is used for disassembling bytes into instructions. Each program defines its own address spaces and memory. Various program elements can be added to the program to further define it as part of the reverse engineering process. Some of the elements that can be defined in the program are labels, references, comments, functions, and data.

Ghidra Tools:

A Ghidra Tool is a collection of building blocks, called *Plugins*. You can create tools by combining different Plugins that cooperate with one another to achieve certain functionality. You can add tools to the Tool Chest or configure them to share data and resources with other tools. Ghidra provides a set of Plugins, but you may create your own Plugins to add more functionality to your tools.

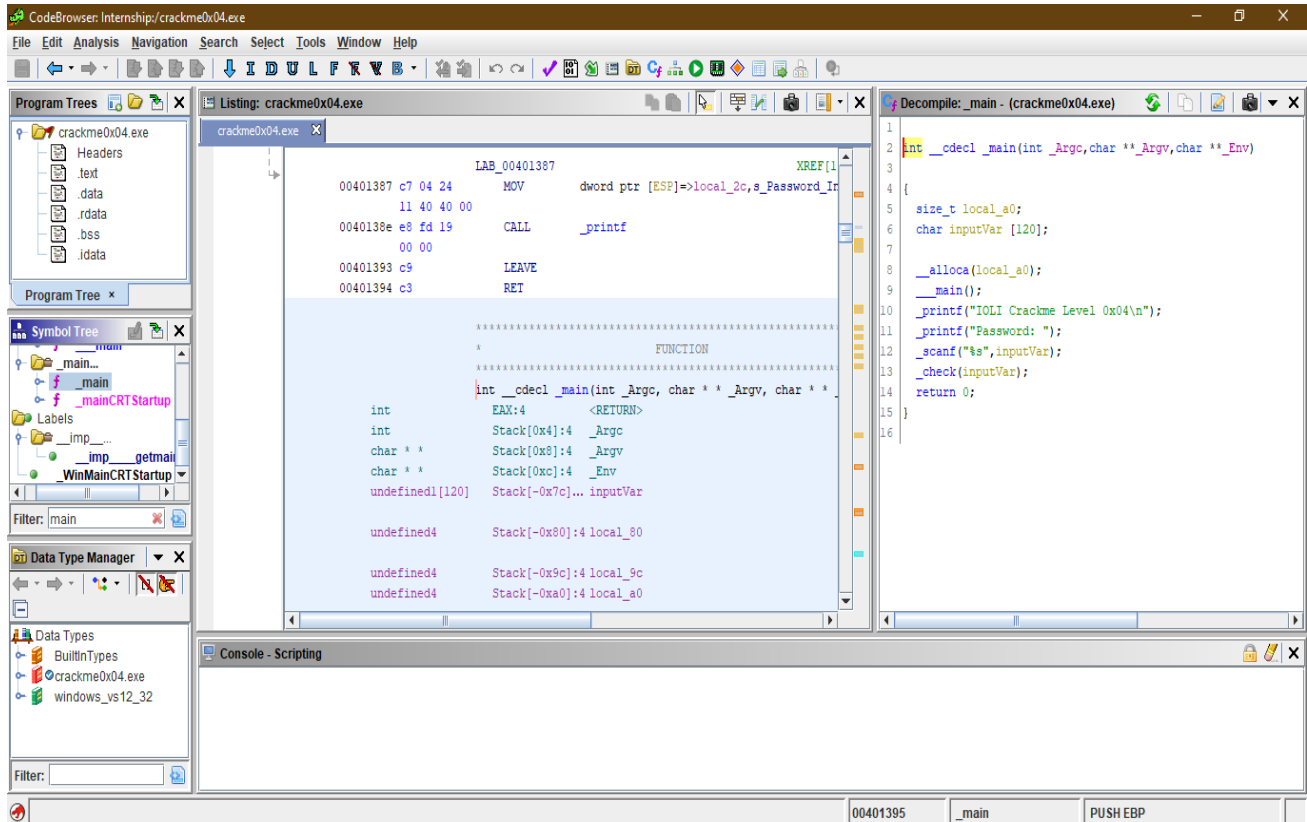


The tool Chest has 2 default tools.



: Code Browser. We'll be using this one.

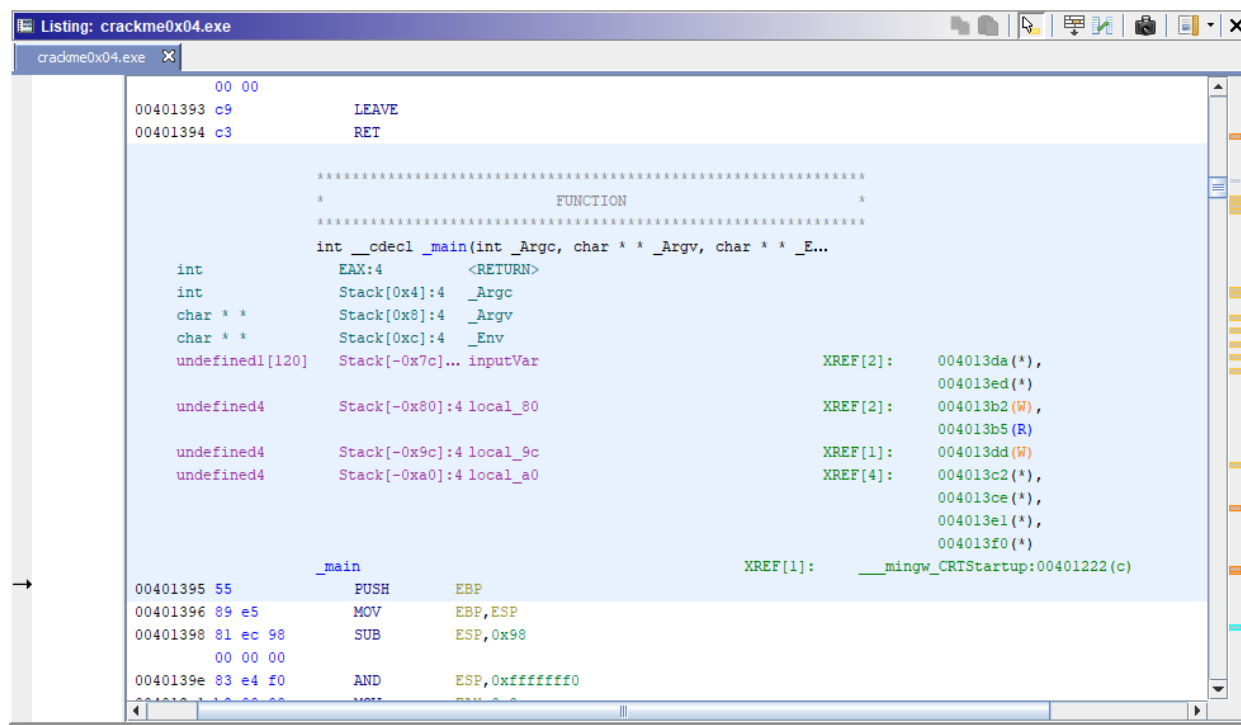
Code Browser



It has the following important components:

- Listing view (middle)
- Decompiler (right)
- Symbol Tree (left-middle)

Listing View:



The screenshot shows a window titled "Listing: crackme0x04.exe" with a tab for "crackme0x04.exe". The main area displays assembly code for the function `__cdecl _main`. The code includes instructions like `LEAVE`, `RET`, `PUSH EBP`, `MOV EBP, ESP`, `SUB ESP, 0x98`, and `AND ESP, 0xffffffff`. It also shows variable declarations for `_Argc`, `_Argv`, and `_Env`, and stack frame information. Cross-references (XREF) are listed on the right, showing pointers to various memory locations and the `__mingw_CRTStartup` function. The interface includes a toolbar at the top and a sidebar on the right for navigation.

```
00 00
00401393 c9      LEAVE
00401394 c3      RET

*****
*                      FUNCTION                      *
*****

int __cdecl _main(int _Argc, char * * _Argv, char * * _E...
EAX:4          <RETURN>
int            Stack[0x4]:4 _Argc
char * *       Stack[0x8]:4 _Argv
char * *       Stack[0xc]:4 _Env
undefined1[120] Stack[-0x7c]... inputVar
undefined4     Stack[-0x80]:4 local_80
undefined4     Stack[-0x9c]:4 local_9c
undefined4     Stack[-0xa0]:4 local_a0

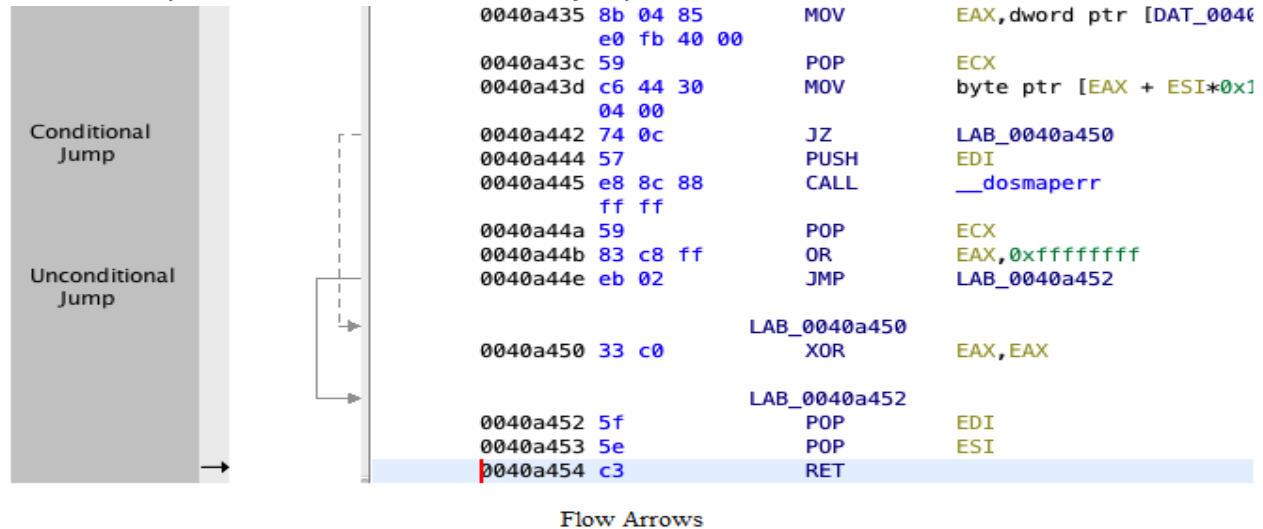
XREF[2]: 004013da(*),
          004013ed(*),
XREF[2]: 004013b2(W),
          004013b5(R),
XREF[1]: 004013dd(W),
XREF[4]: 004013c2(*),
          004013ce(*),
          004013e1(*),
          004013f0(*),
XREF[1]: __mingw_CRTStartup:00401222(c)

_main
00401395 55      PUSH     EBP
00401396 89 e5    MOV      EBP,ESP
00401398 81 ec 98    SUB      ESP,0x98
00 00 00
0040139e 83 e4 f0    AND      ESP,0xffffffff
004013a0 83 e4 f0    AND      ESP,0xffffffff
```

Code is made up of various elements such as addresses, bytes, mnemonics, and operands. The Listing uses *fields* to display these elements. The overall layout of the Listing can be changed by adjusting the size and position of the fields using the [Browser Field Formatter](#).

The View: The Listing can either display an entire program or a subset of a program. The [Program Tree](#) can be used to restrict the view to a module or fragment.

Flow Arrows: The flow arrows graphically illustrate the flow of execution within a function. They appear as arrows on the left side of the Listing display indicating source and destinations for jumps. Conditional jumps are indicated by dashed lines; unconditional jumps are indicated by solid lines. Flow lines are bolded when the cursor is positioned at the source of the jump.



Mouse Hover: The Listing includes the capability of displaying popup windows when the user hovers the mouse over a particular field. This occurs whenever a plugin has additional information that it wants to display about that field. The popup window disappears when the user moves the mouse off of the window or field. Some example popup windows that can be displayed: *Reference Popups*, *Truncated Text Popups*, and *Data Type Popups*.

DECOMPILER:

The Decompiler plugin is a sophisticated transformation engine which automatically converts the binary representation of individual functions into a high-level C representation.

```

Cf Decompile: _main - (crackme0x04.exe)
1
2 int __cdecl _main(int _Argc, char **_Argv, char **_Env)
3
4 {
5     size_t local_a0;
6     char inputVar [120];
7
8     __alloca(local_a0);
9     __main();
10    _printf("IOLI Crackme Level 0x04\n");
11    _printf("Password: ");
12    _scanf("%s",inputVar);
13    _check(inputVar);
14    return 0;
15 }
16

```

Mouse Actions:

- Double-click - Navigates to the symbol that was clicked.
- Control-double-click - Navigates to the symbol that was clicked, opening the results in a new window.

Rename Variable

Any parameter or local variable can be renamed. Just place the cursor over a variable definition, or any use of the variable and choose Rename Variable from the popup menu. The name will now be saved for this function, so the next time the decompiler displays the code for the function, the same name is used

Edit Function Signature

The Edit Function Signature dialog allows you to change the function's signature, the calling convention, whether the function is inline and whether the function has no return.

HWND FUN_00401040 (HINSTANCE param_1, int param_2)

Function Name:

Calling Convention:

Function Attributes:

- ☐ Varargs
- ☐ In Line
- ☐ No Return
- ☐ Use Custom Storage

Function Variables

Index	Datatype	Name	Storage
	HWND	<RETURN>	EAX:4
1	HINSTANCE	param_1	Stack[0x4]:4
2	int	param_2	Stack[0x8]:4

Call Fixup:

OK Cancel

The function signature includes

- function name
- return type
- number of parameters
- parameter names
- parameter type
- varargs (variable arguments)

- To edit a function's signature from the Decompile window. Just place the cursor over any function name, select Edit Function Signature from the popup menu, and the dialog will appear with the function's current information.

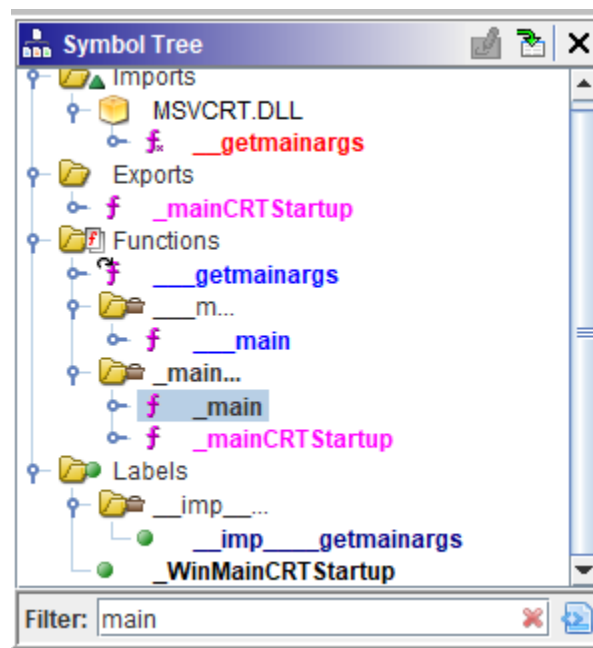
Graph AST Control Flow

Selecting Graph AST Control Flow, from the decompiler provider window toolbar, will generate an abstract syntax tree (AST) control flow graph based upon the decompiler results and render the graph within the current Graph Service.

If no Graph Service is available then this action will not be present.

Symbol Tree:

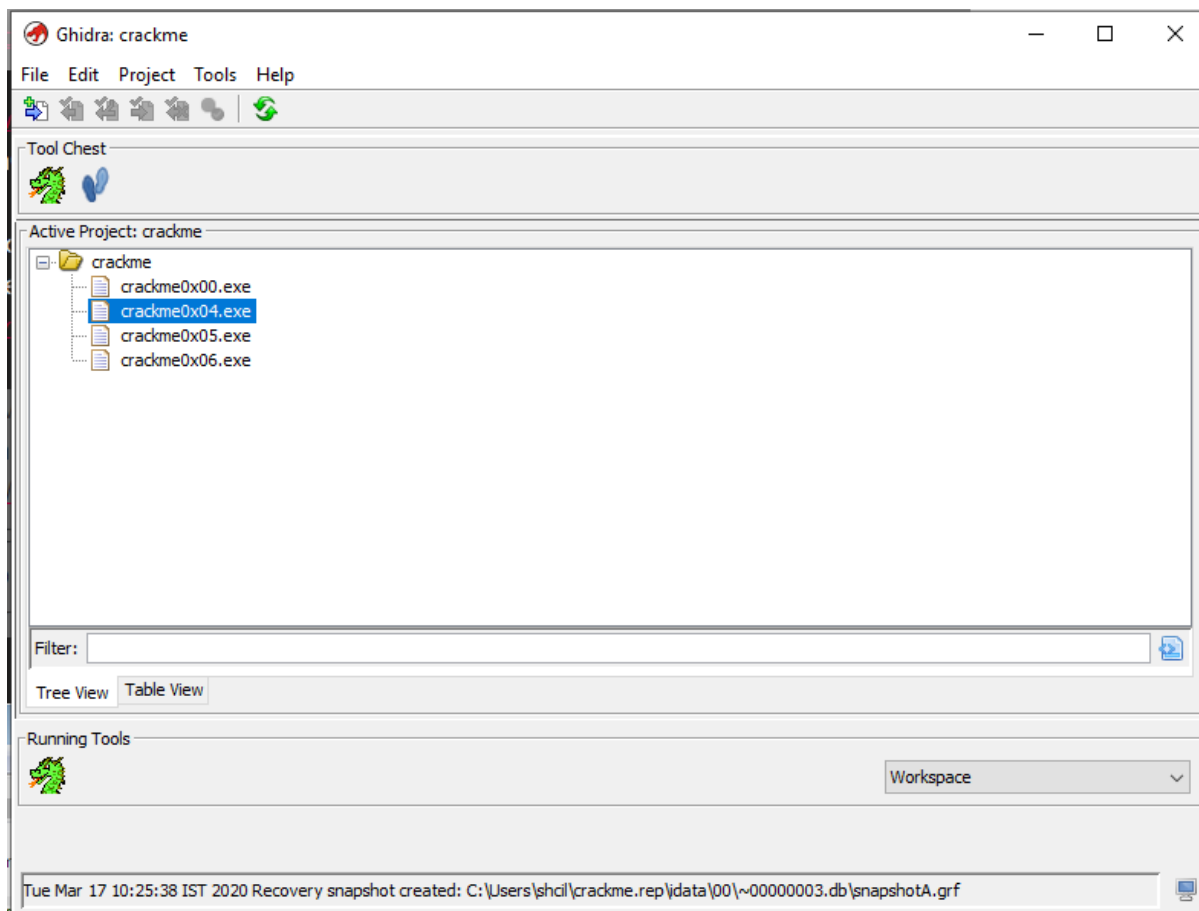
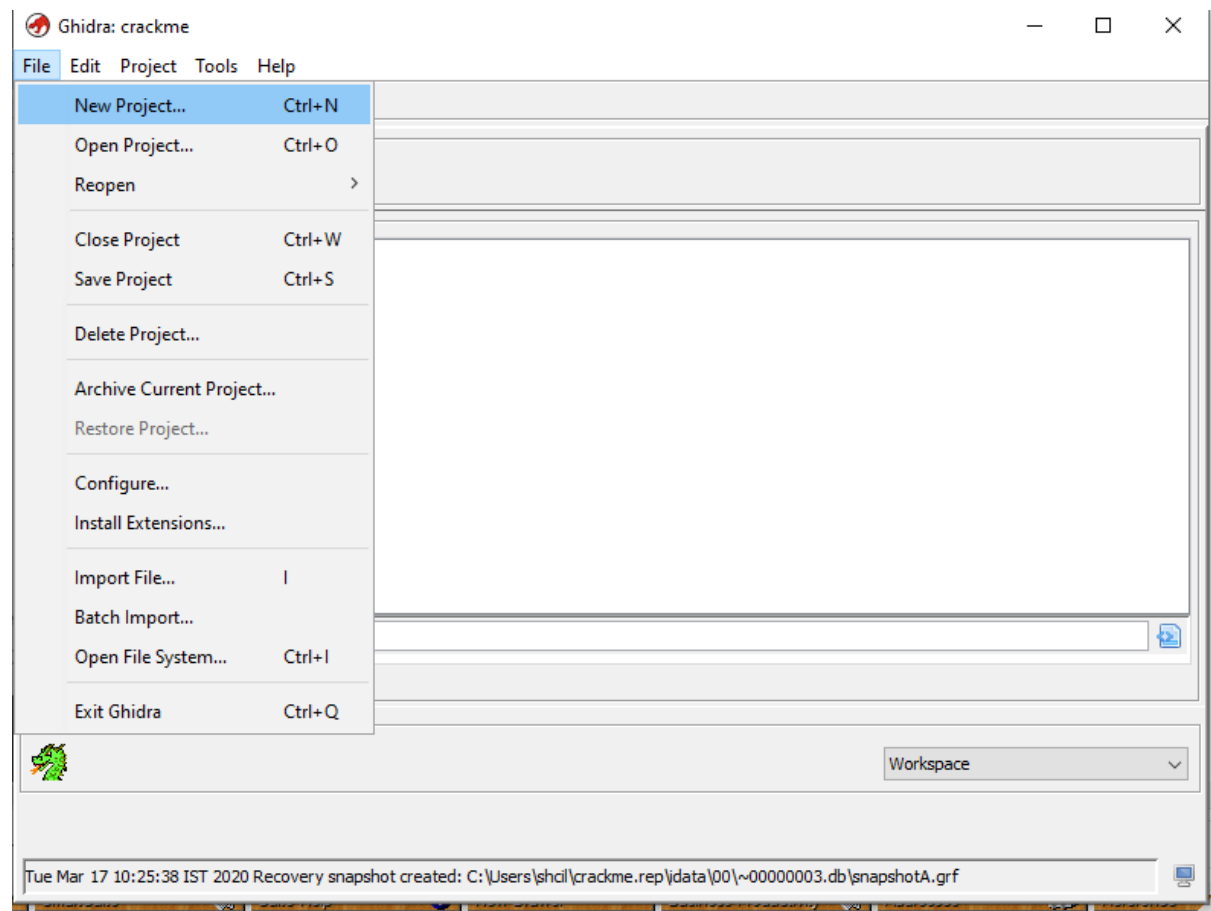
The Symbol Tree shows symbols from a program in a hierarchical view. The Symbol tree is organized by the following categories: *Externals, Function, Labels, Classes, and Namespaces*.



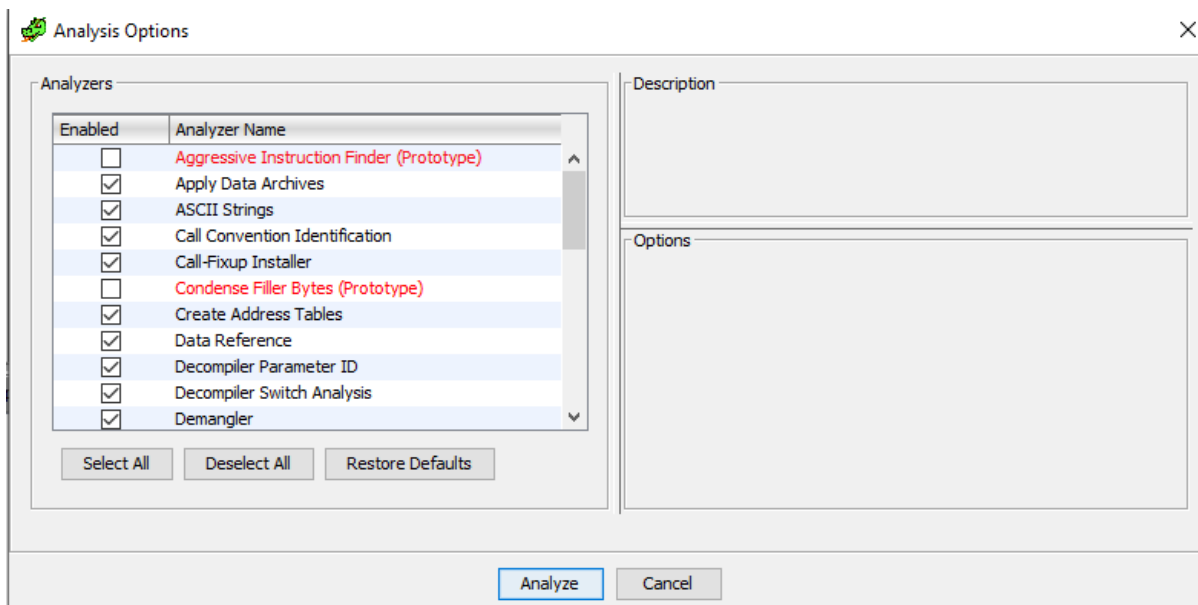
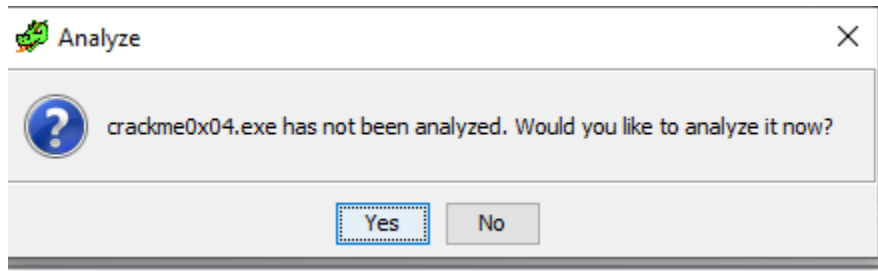
USING GHIDRA TO SOLVE CRACKME0x04.exe

Project Creation

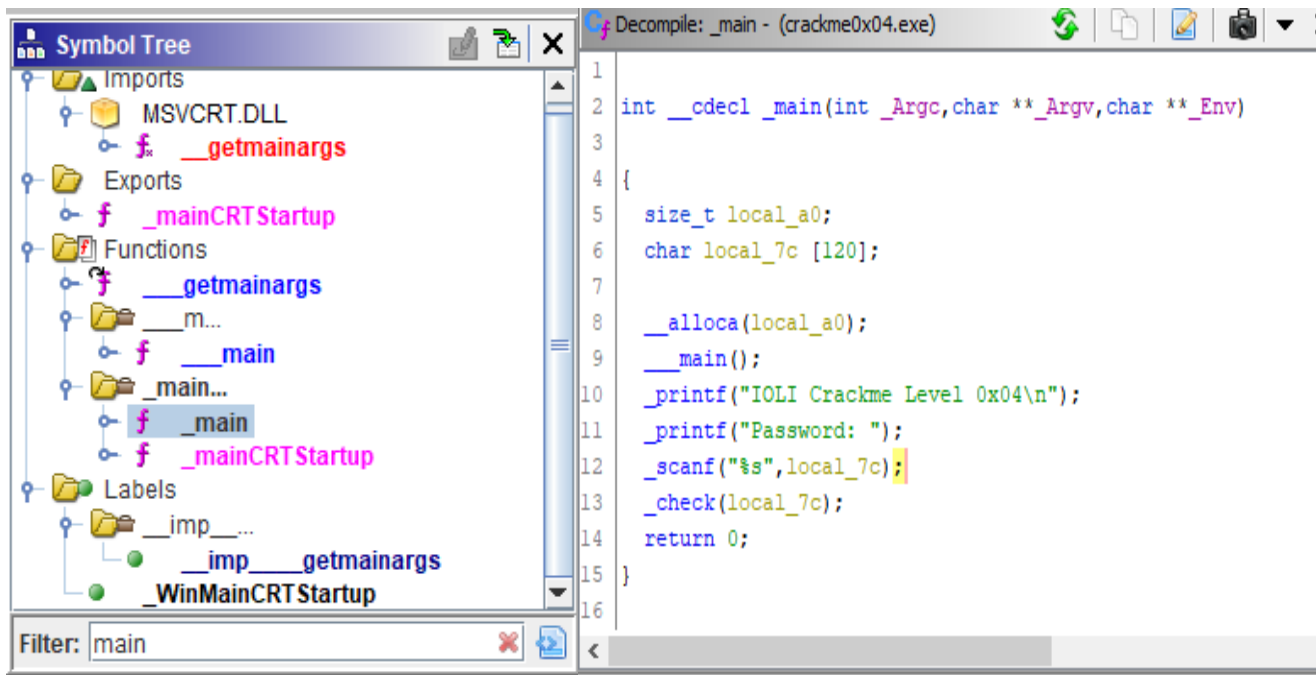
Project window enables downloading and managing binary files. The crackme0x04.exe binary file which we are going to work with is available for downloading here (<https://github.com/Maijin/Workshop2015/tree/master/IOLI-crackme/bin-win32>). Create a new project and import the downloaded crackme0x04.exe file. This can be done by dragging and dropping the binary file into the project window or by selecting File -> Import File in project window options. After this has been done you will see the downloaded crackme0x04.exe file.



Next, double click crackme0x04.exe in the project window to open code browser. A message box with the question “Do you want to analyze the binary file?” will be displayed. Select “Yes” and you will be shown various types of analysis available. Default settings are suitable for this project, so select “Analyze” and wait until Ghidra has completed the task. As soon as this has been done, code browser and main windows will be displayed.



After importing the binary, select the “Symbol Tree” window and expand the “Functions” folder to get to the `_main` function or use the “Filter” input box to search for `_main`. Once you click on it, you should be inside the main function and you will see the decompilation output to the right.



Rename the variables to meaningful names and added comments to make things more readable

```
int __cdecl _main(int _Argc, char **_Argv, char **_Env)
{
    size_t local_a0;
    char inputVar [120];

    __alloca(local_a0);
    __main();
    _printf("IOLI Crackme Level 0x04\n");
    _printf("Password: ");
    _scanf("%s", inputVar);
    _check(inputVar);
    return 0;
}
```

We can see that a function named `_check` is being called with a character array called `local_7c`, which is our user input from the command line.

Double click on `_check` in the decompilation window and let's take a closer look at this function.

Check():

```
void __cdecl _check(char *Input)
{
    size_t sVar1;
    char CharOfInput;
    uint IndexOfInput;
    int TotalOfInput;
    int CharInt;

    TotalOfInput = 0;
    IndexOfInput = 0;
    while( true ) {
        sVar1 = _strlen(Input);
        if (sVar1 <= IndexOfInput) {
            _printf("Password Incorrect!\n");
            return;
        }
        CharOfInput = Input[IndexOfInput];
        _sscanf(&CharOfInput, "%d", &CharInt);
        TotalOfInput = TotalOfInput + CharInt;
        if (TotalOfInput == 0xf) break;
        IndexOfInput = IndexOfInput + 1;
    }
    _printf("Password OK!\n");
}
```

IndexOfInput is iterated in this while loop.

In each iteration, each char of input is gained by Input[IndexOfInput] and stored in Char.

The sscanf() function allows us to read formatted data from a string rather than standard input or keyboard. Its syntax is as follows:

Syntax: int sscanf(const char *str, const char * control_string [arg_1, arg_2, ...]);

The first argument is a pointer to the string from where we want to read the data. The rest of the arguments of sscanf() is same as that of scanf(). It returns the number of items read from the string and -1 if an error is encountered.

Link for eg: <https://overiq.com/c-programming-101/the-sscanf-function-in-c/>

Sscanf() statement stores Char (which is in char format) in CharInt (which is in integer format).

TotalOfInput is calculated by adding CharInt in each iteration.

If TotalOfInput=15 then password is correct.

The IndexOfInput is incremented. Then its checked if index is more than the length of Input string.

If yes, password is wrong.

Therefore, password requires a string of numbers where the addition of digits is 15 at some point.

```

PS C:\Users\windows\Downloads> .//crackme0x04.exe
IOLI Crackme Level 0x04
Password: 5433
Password OK!
PS C:\Users\windows\Downloads> .//crackme0x04.exe
IOLI Crackme Level 0x04
Password: 96
Password OK!
PS C:\Users\windows\Downloads> .//crackme0x04.exe
IOLI Crackme Level 0x04
Password: 12345
Password OK!
PS C:\Users\windows\Downloads> █

```

Some More Solved Examples:

Crackme0x00:

```

_printf("IOLI Crackme Level 0x00\n");
_printf("Password: ");
_scanf("%s",inputVar);
passCheck = _strcmp(inputVar,"250382");
if (passCheck == 0) {
    _printf("Password OK :)\n");
}
else {
    _printf("Invalid Password!\n");
}

```

Password is a string here : 250382 which is compared with input.

```

PS C:\Users\windows\Downloads> .\\crackme0x00.exe
IOLI Crackme Level 0x00
Password: 250382
Password OK :)
PS C:\Users\windows\Downloads> █

```

Crackme0x01:

```

_printf("IOLI Crackme Level 0x01\n");
_printf("Password: ");
_scanf("%d",&inputvar);
if (inputvar == 0x149a) {
    _printf("Password OK :)\n");
}
else {
    _printf("Invalid Password!\n");
}
return 0;

```

Decimal value of 0x149a=5274

```
PS C:\Users\windows\Downloads> .\crackme0x01.exe
IOLI Crackme Level 0x01
Password: 5274
Password OK :)
PS C:\Users\windows\Downloads>
```

Crackme0x02:

```
_printf("IOLI Crackme Level 0x02\n");
_printf("Password: ");
_scanf("%d",&inputVar);
if (inputVar == 0x52b24) {
    _printf("Password OK :)\n");
}
else {
    _printf("Invalid Password!\n");
}
```

Decimal value of 0x52b24=338724

```
Invalid Password.
PS C:\Users\windows\Downloads> .\crackme0x02.exe
IOLI Crackme Level 0x02
Password: 338724
Password OK :)
PS C:\Users\windows\Downloads>
```

Crackme0x03:

```
_printf("IOLI Crackme Level 0x03\n");
_printf("Password: ");
_scanf("%d",&inputVar);
_test(inputVar,0x52b24);
return 0;
```

```
PS C:\Users\windows\Downloads> .\crackme0x03.exe
IOLI Crackme Level 0x03
Password: 338724
Password OK!!! :)
PS C:\Users\windows\Downloads>
```