# Lab – Introduction to Using Ghidra

## Overview

In this lab, you will be introduced to some of the higher-level features of Ghidra. Ghidra is a software reverse engineering (SRE) framework developed by NSA's Research Directorate for NSA's cybersecurity mission. Ghidra helps analyze malicious code and malware and can give cybersecurity professionals a better understanding of potential vulnerabilities in their networks and systems.

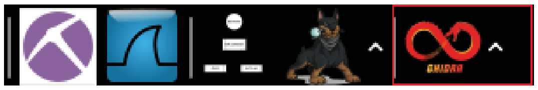
Ghidra was first released to the public by the NSA in 2019 after being referenced in the WikiLeaks' March 2017 "Vault 7" disclosure, which discussed several hacking tools used by the CIA disclosing Ghidra as a reverse-engineering tool created by the NSA.

## Lab Requirements

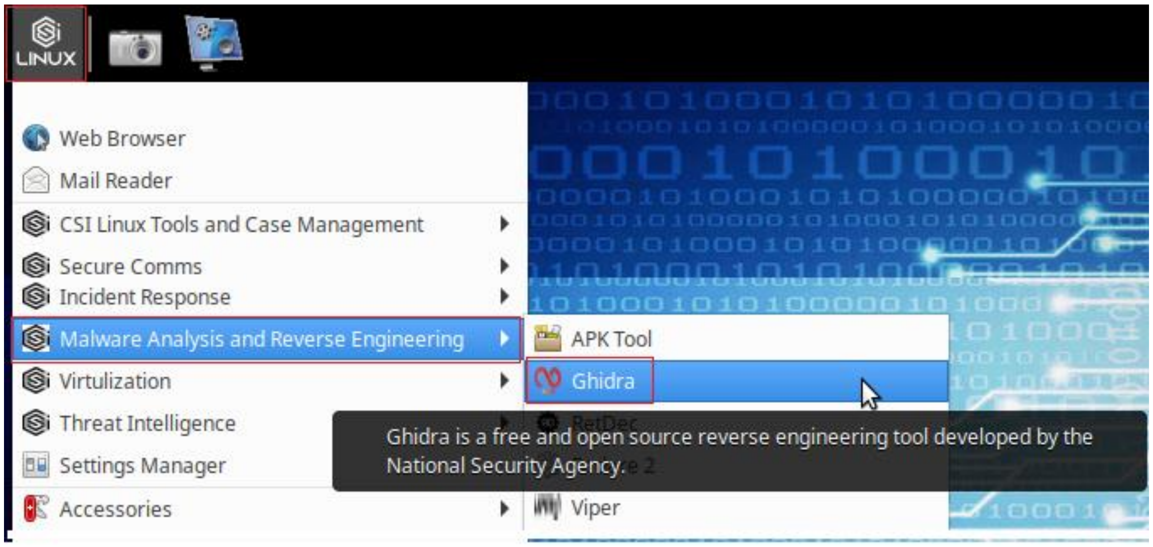
- On installation of VirtualBox with the extension pack.
- One virtual install of the latest version of Ubuntu, Kali Linux, or CSI Linux.
- One virtual install of Windows 7 or 10
- VirtualBox network adapter set to NAT network.

## Launch Ghidra

You can launch Ghidra using the Ghidra icon located in the bottom taskbar

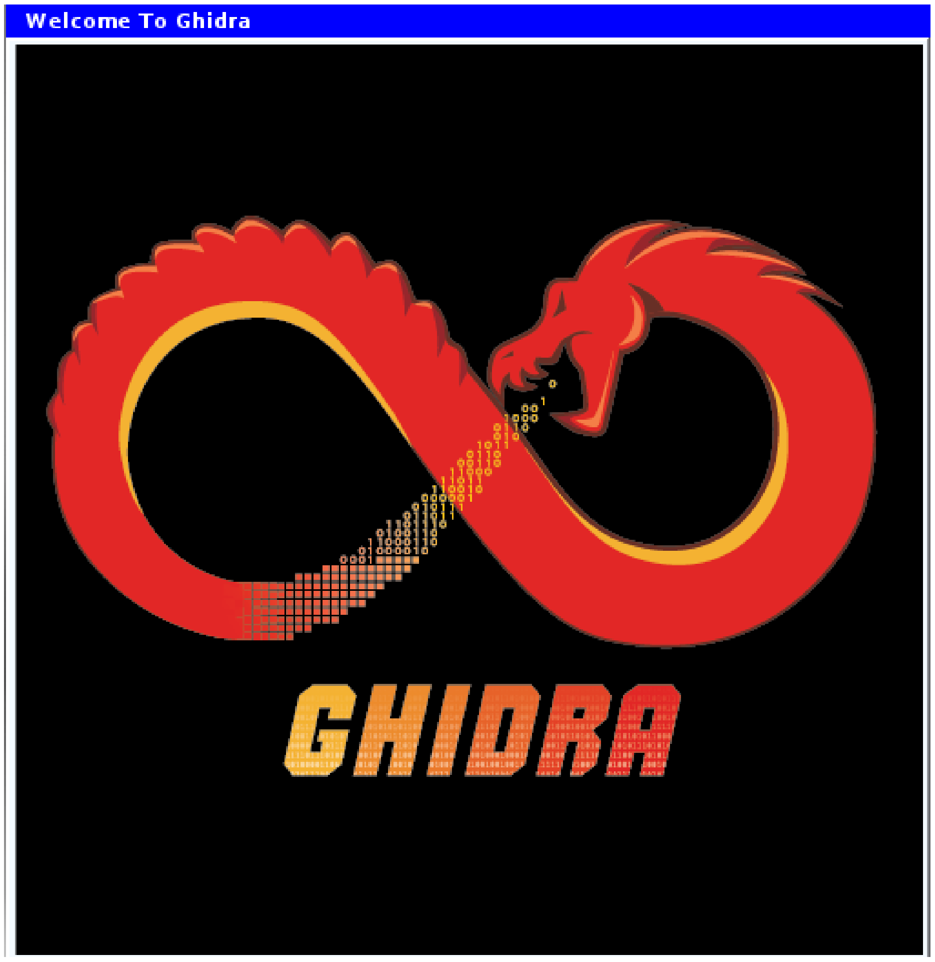


Your other option is to use the Application launcher. Scroll down to **Malware Analysis and Reverse Engineering** and launch **Ghidra** from the content menu.

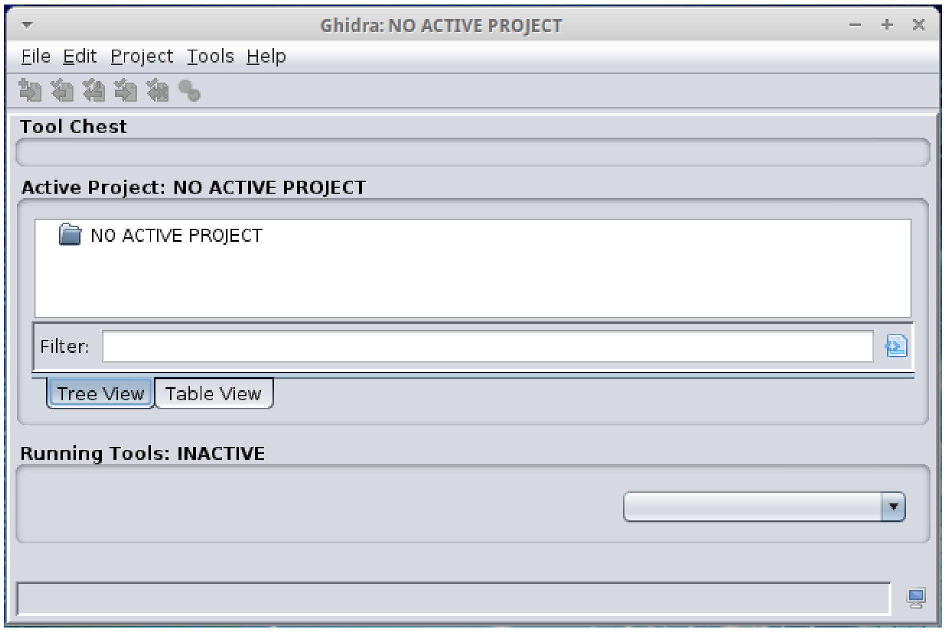


Ghidra is a free and open source reverse engineering tool developed by the National Security Agency.

Ghidra launches

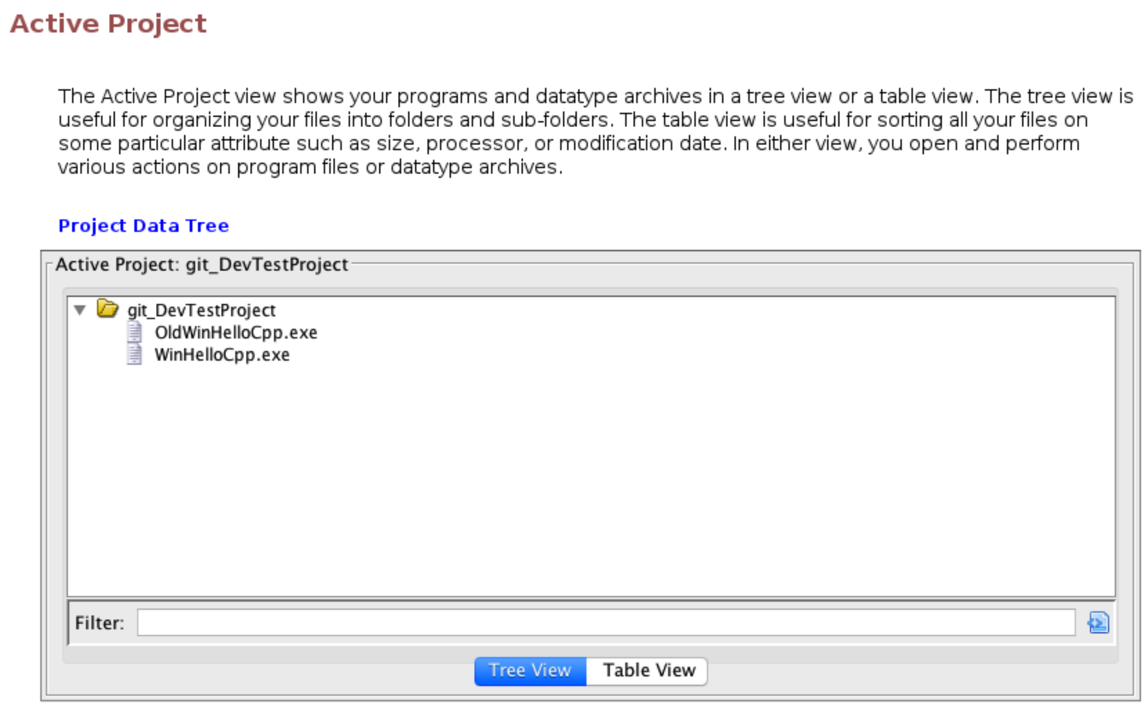


After a short pause, the Active Project screen opens.


Ghidra: NO ACTIVE PROJECT
File Edit Project Tools Help
Tool Chest
Active Project: NO ACTIVE PROJECT
NO ACTIVE PROJECT
Filter:
Tree View
Table View
Running Tools: INACTIVE

Anytime you need to access the Ghidra help documentation, press the F1 key. Regardless of the Ghidra feature or tool you are currently using, the help documentation will open to the section in the help documentation that references that feature.

Currently, we are working within the Active Project screen. When I press F1, the help documentation opens to the section with information about using the Active Project window.



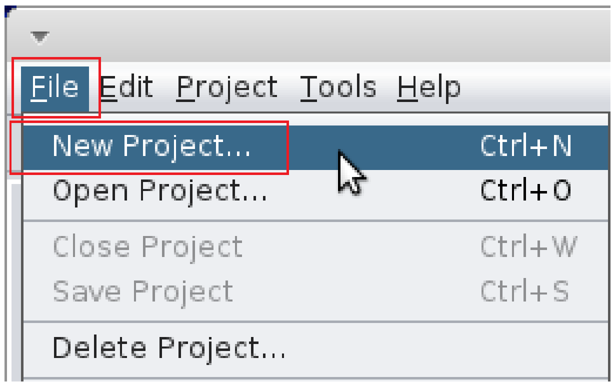
Active Project

The Active Project view shows your programs and datatype archives in a tree view or a table view. The tree view is useful for organizing your files into folders and sub-folders. The table view is useful for sorting all your files on some particular attribute such as size, processor, or modification date. In either view, you open and perform various actions on program files or datatype archives.

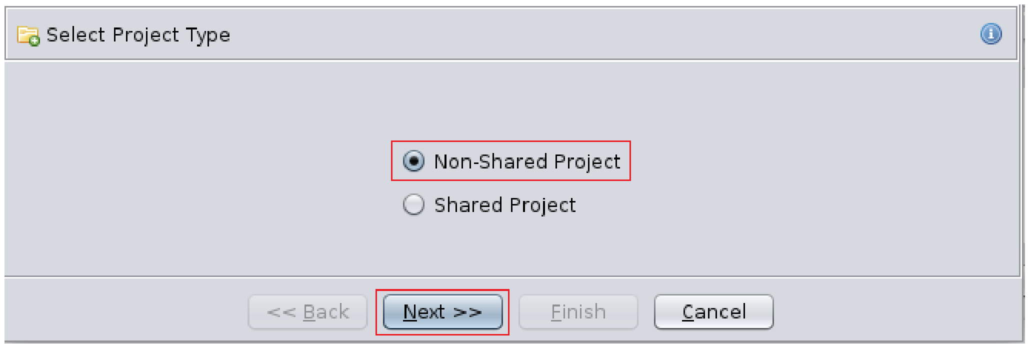
Project Data Tree

## Creating a new project

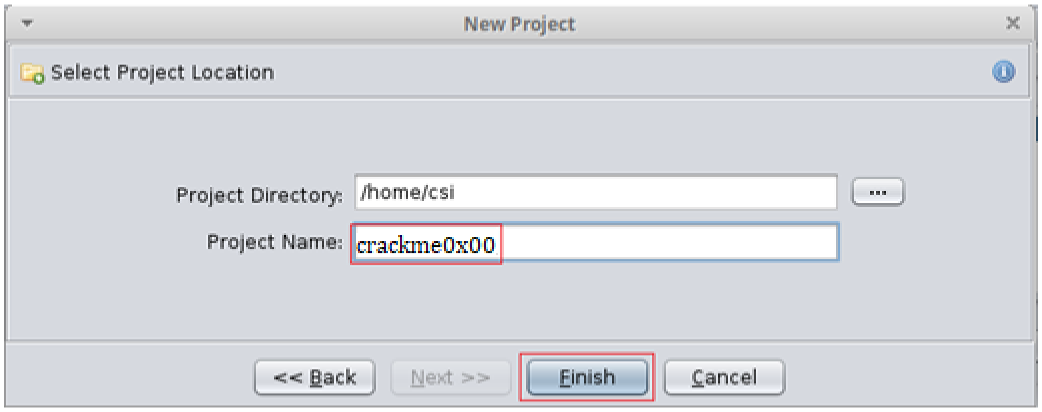
We next need to create a new project. From the Ghidra taskbar, go to File>New Project.



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, **crackme0x00,** 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.



Project Directory: /home/csi

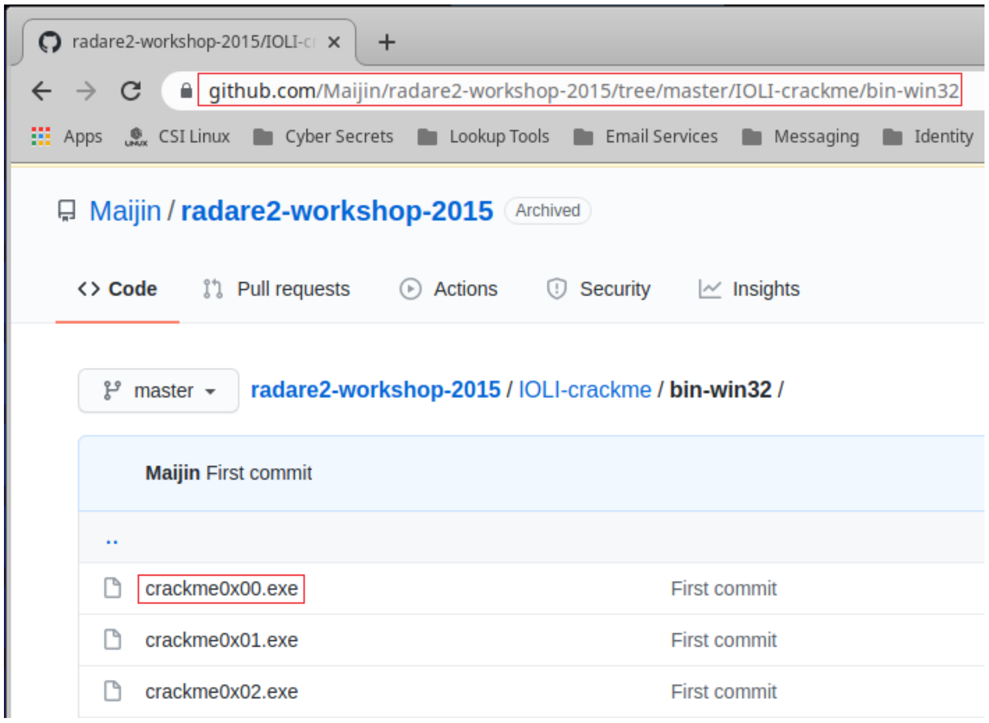
Project Name: crackme0x00

We next need a file to analyze. We can get some rudimentary files to crack using the following GitHub repository.

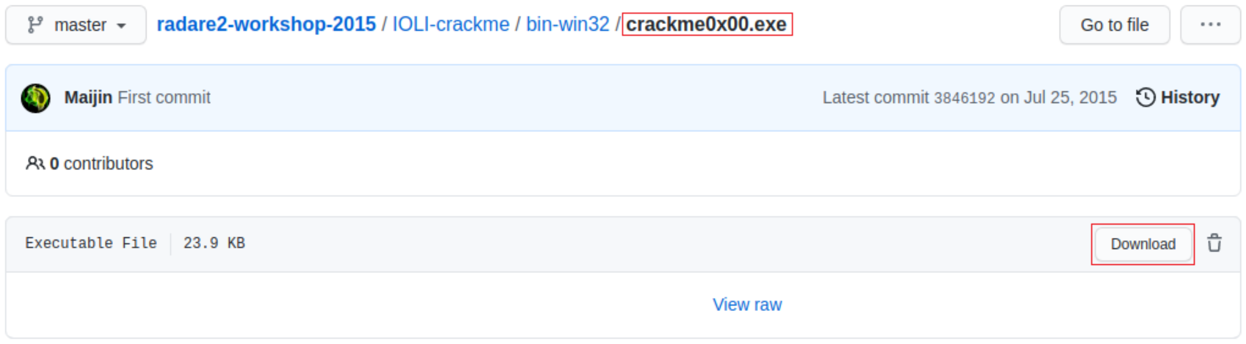
https://github.com/Maijin/radare2-workshop-2015/tree/master/IOLI-crackme/bin-win32

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

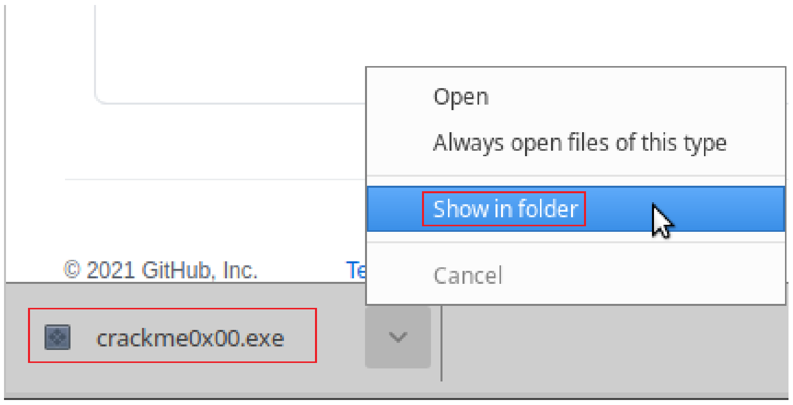
We will be analyzing the **crackme0x00.exe** file. From the list of crackme files, click on the crackme0x00.exe file.



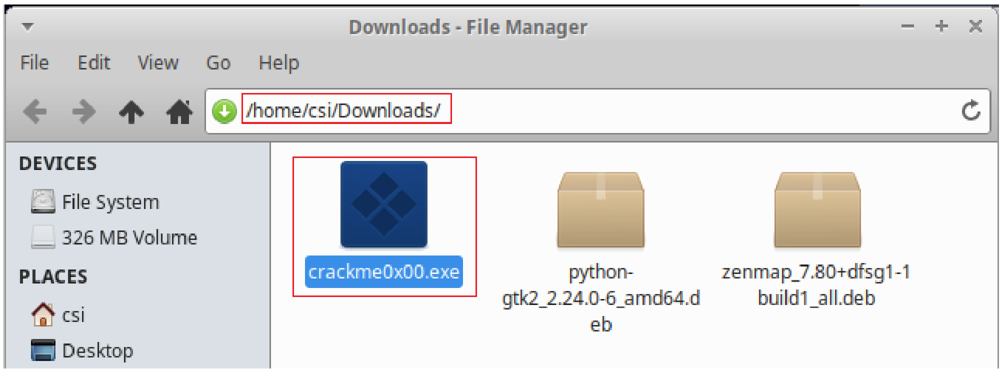
On the next page, click on the Download button.


master
radare2-workshop-2015 / IOLI-crackme / bin-win32 / crackme0x00.exe
Go to file
...
Maijin First commit
Latest commit 3846192 on Jul 25, 2015
History
0 contributors
Executable File
23.9 KB
Download
View raw

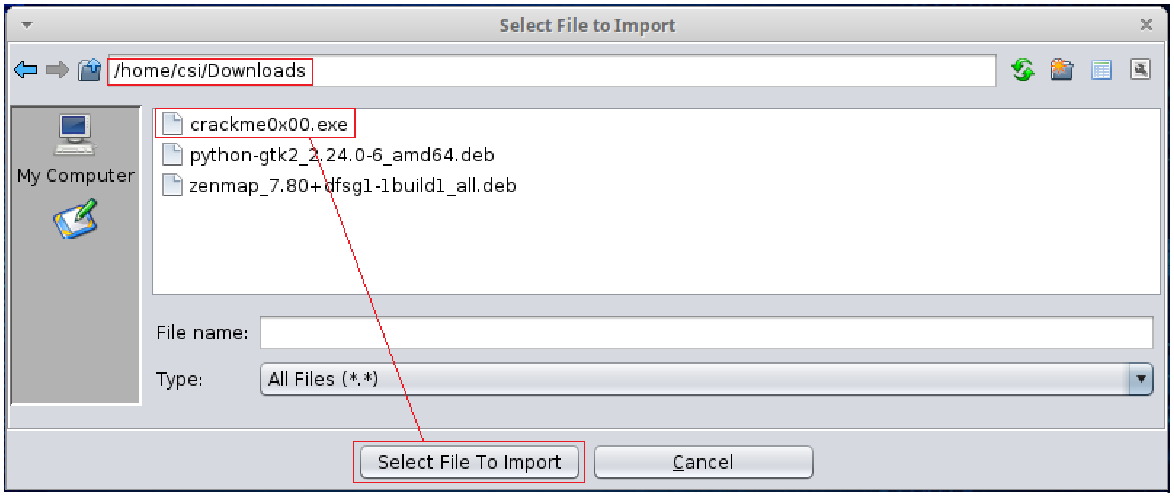
Once the file downloads, click the down arrow, and from the context menu, select **Show in folder**.



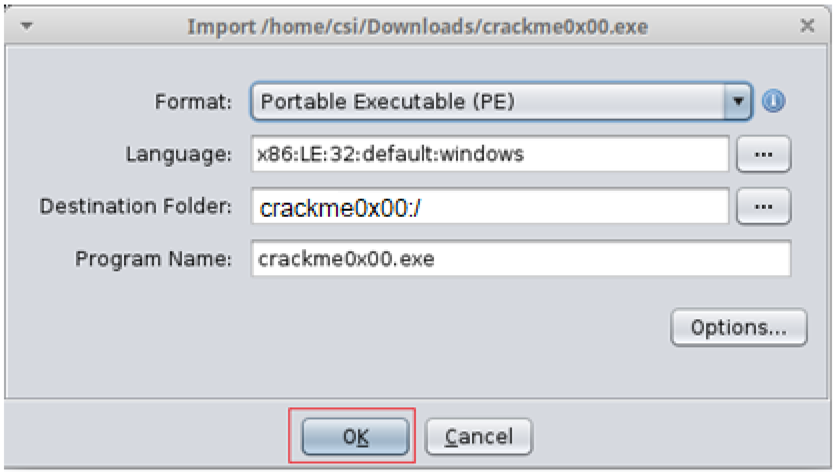
Take note that the file is in your Downloads Directory.



Close out your browser and the File Manager. Return to Ghidra. There are two ways to import our downloaded file into Ghidra. You can click on File> Import File or open your Downloads directory and drag and drop the crackme0x00.exe into the project window.

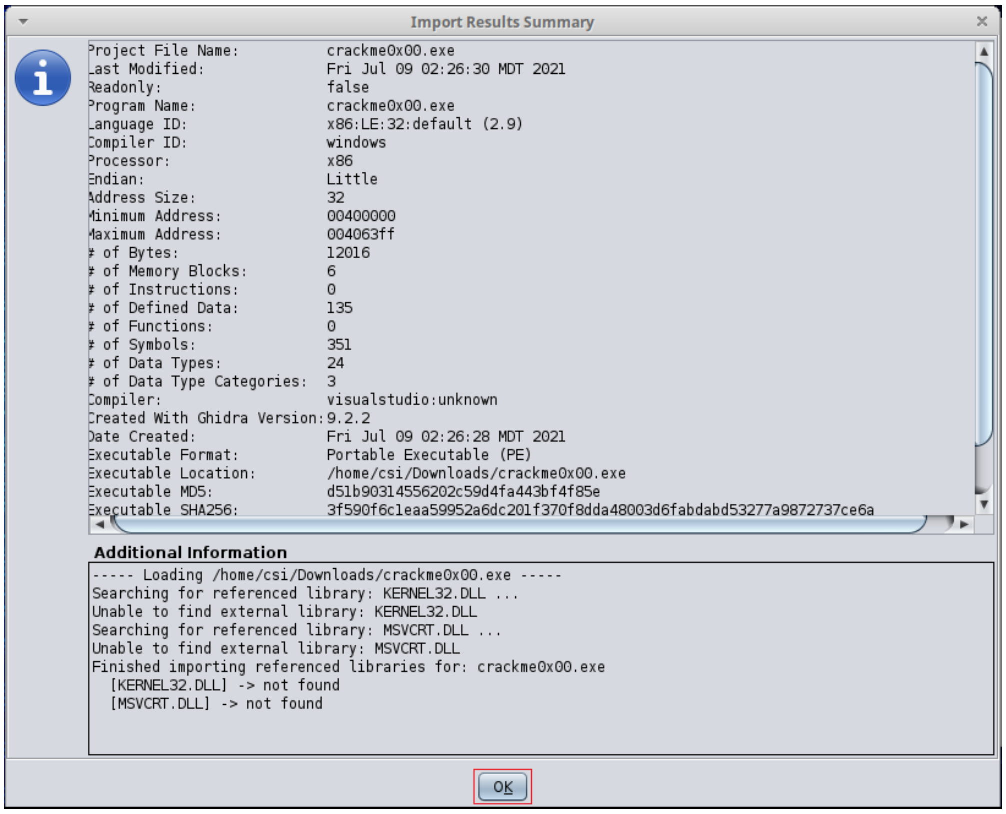


When you first import your file, Ghidra will respond with the information about the file. Click OK.



Format: Portable Executable (PE)
Language: x86:LE:32:default:windows
Destination Folder: crackme0x00:/
Program Name: crackme0x00.exe

Ghidra then displays a screen like the one shown below with an import summary of the file. Click OK.

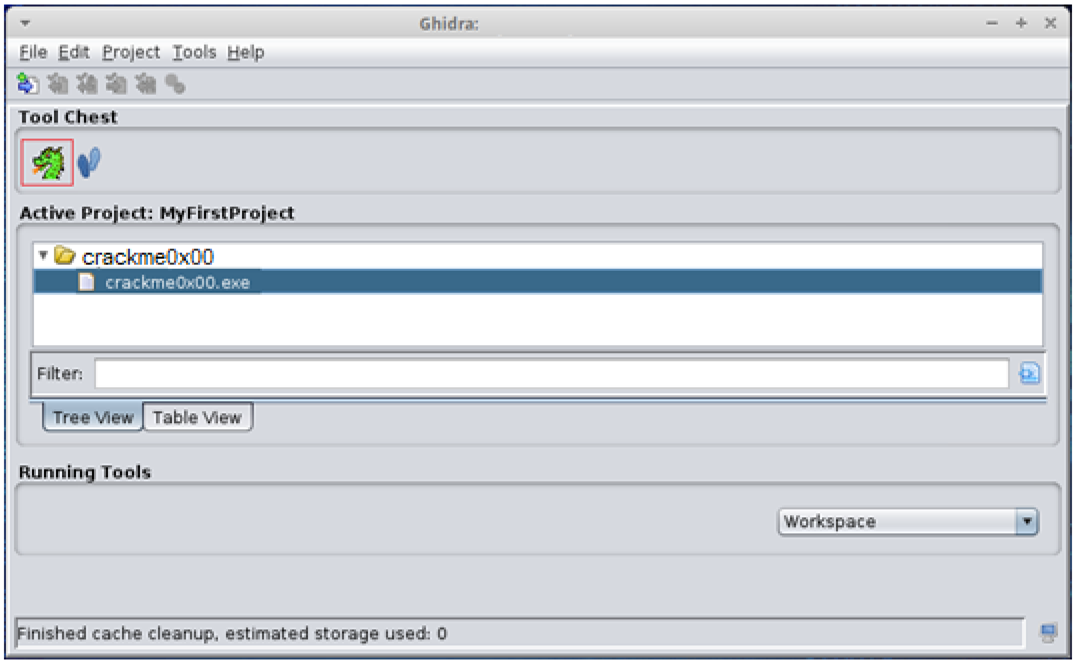


```
Project File Name:            crackme0x00.exe
Last Modified:                Fri Jul 09 02:26:30 MDT 2021
Readonly:                     false
Program Name:                 crackme0x00.exe
Language ID:                  x86:LE:32:default (2.9)
Compiler ID:                  windows
Processor:                    x86
Endian:                       Little
Address Size:                 32
Minimum Address:              00400000
Maximum Address:              004063ff
# of Bytes:                   12016
# of Memory Blocks:           6
# of Instructions:            0
# of Defined Data:            135
# of Functions:               0
# of Symbols:                 351
# of Data Types:              24
# of Data Type Categories:    3
Compiler:                     visualstudio:unknown
Created With Ghidra Version:9.2.2
Date Created:                 Fri Jul 09 02:26:28 MDT 2021
Executable Format:            Portable Executable (PE)
Executable Location:          /home/csi/Downloads/crackme0x00.exe
Executable MD5:               d51b90314556202c59d4fa443bf4f85e
Executable SHA256:            3f590f6c1eaa59952a6dc201f370f8dda48003d6fabdabd53277a9872737ce6a
```

Additional Information

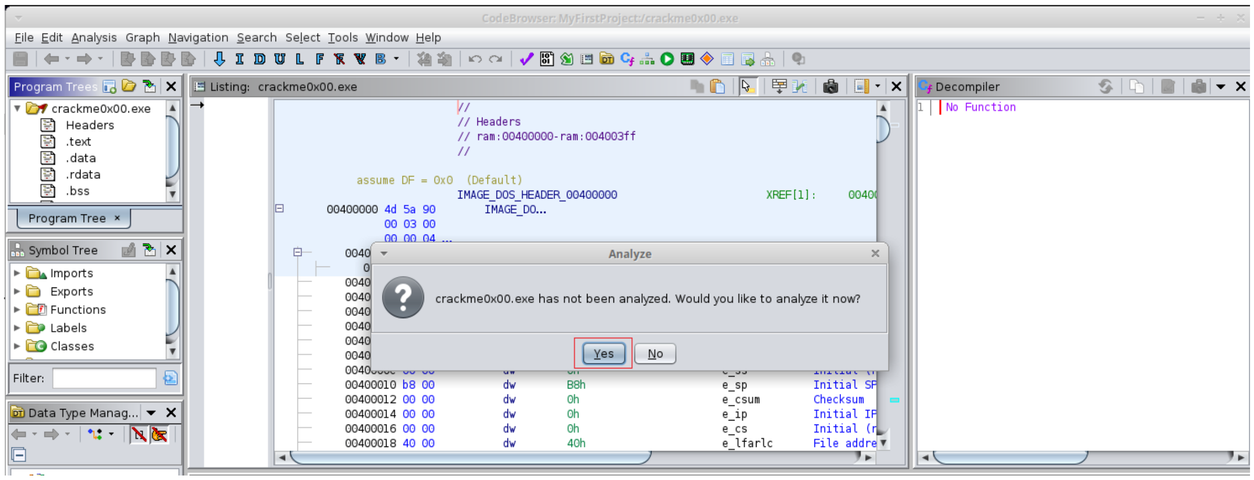
```
----- Loading /home/csi/Downloads/crackme0x00.exe -----
Searching for referenced library: KERNEL32.DLL ...
Unable to find external library: KERNEL32.DLL
Searching for referenced library: MSVCRT.DLL ...
Unable to find external library: MSVCRT.DLL
Finished importing referenced libraries for: crackme0x00.exe
  [KERNEL32.DLL] -> not found
  [MSVCRT.DLL] -> not found
```

Back at the Active Project window, you can either double-click on the file or use the green Ghidra dragon to begin the analysis.


Ghidra:
File Edit Project Tools Help
Tool Chest
Active Project: MyFirstProject
crackme0x00
crackme0x00.exe
Filter:
Tree View
Table View
Running Tools
Workspace
Finished cache cleanup, estimated storage used: 0

Ghidra first displays the file’s assembler language in the center Listing window and asks if you want to analyze the file. Click “Yes.”

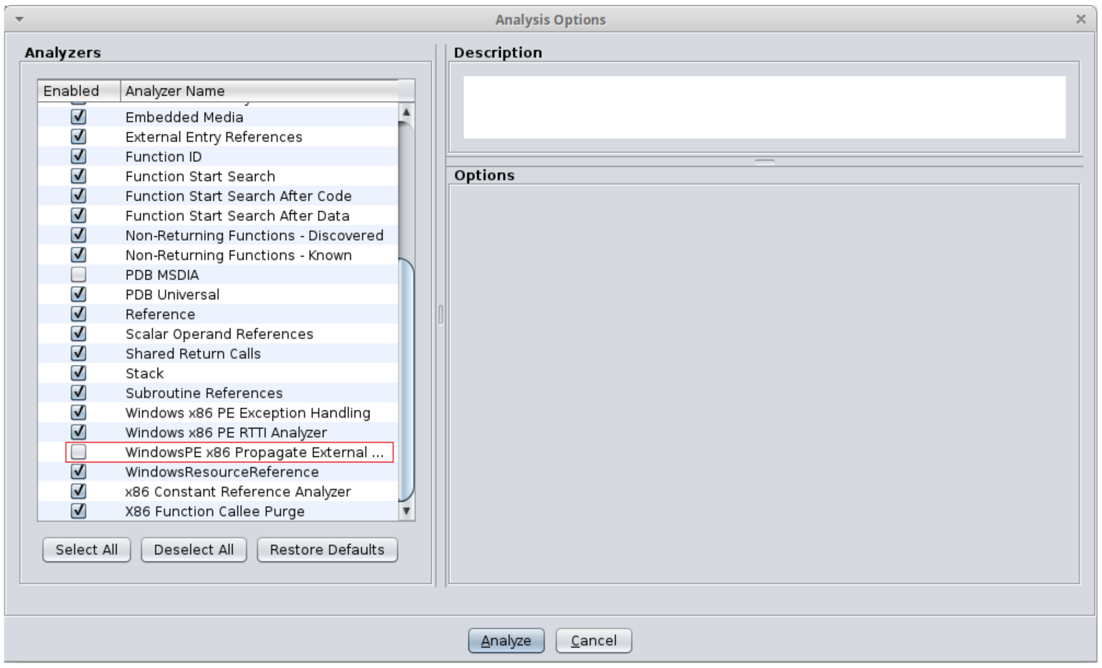
If the Ghidra fails to show any data in the Listing window, close out the Listing window and back the Active Project window, x2 click the file one more time and relaunch. This is a well-known feature of Ghidra.



```
                         //
                         // Headers
                         // ram:00400000-ram:004003ff
                         //
      assume DF = 0x0  (Default)
                         IMAGE_DOS_HEADER_00400000                XREF[1]:     0040
    00400000 4d 5a 90       IMAGE_DO...
             00 03 00
             00 00 04 ...
    00400010 b8 00      dw      B8h          e_sp       Initial SP
    00400012 00 00      dw      0h           e_csum     Checksum
    00400014 00 00      dw      0h           e_ip       Initial IP
    00400016 00 00      dw      0h           e_cs       Initial (r
    00400018 40 00      dw      40h          e_lfarlc   File addre
```

crackme0x00.exe has not been analyzed. Would you like to analyze it now?

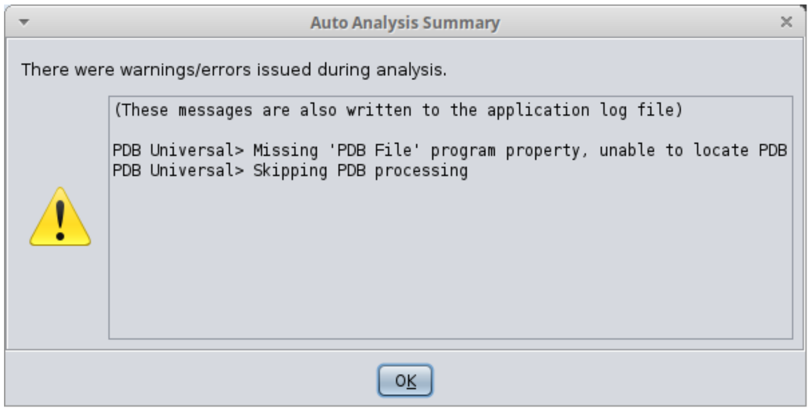
On the next screen, you have your analyzing options:



| Enabled | Analyzer Name |
|---|---|
| ✓ | Embedded Media |
| ✓ | External Entry References |
| ✓ | Function ID |
| ✓ | Function Start Search |
| ✓ | Function Start Search After Code |
| ✓ | Function Start Search After Data |
| ✓ | Non-Returning Functions - Discovered |
| ✓ | Non-Returning Functions - Known |
| | PDB MSDIA |
| ✓ | PDB Universal |
| ✓ | Reference |
| ✓ | Scalar Operand References |
| ✓ | Shared Return Calls |
| ✓ | Stack |
| ✓ | Subroutine References |
| ✓ | Windows x86 PE Exception Handling |
| ✓ | Windows x86 PE RTTI Analyzer |
| | WindowsPE x86 Propagate External ... |
| ✓ | WindowsResourceReference |
| ✓ | x86 Constant Reference Analyzer |
| ✓ | X86 Function Callee Purge |

Select the **“WindowsPE x86 Propogate External Parameters”** option. This analyzer populates push instructions with comments which might help us make sense of the binary file more easily. We then select “Analyze,” and Ghidra starts analyzing our executable.

Upon completion, we are presented with the following error message regarding the “PDB”:



There were warnings/errors issued during analysis.

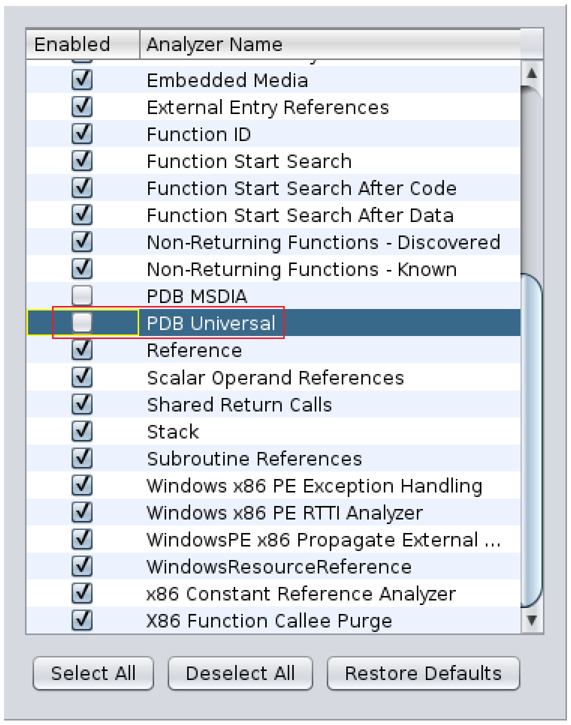
```
(These messages are also written to the application log file)

PDB Universal> Missing 'PDB File' program property, unable to locate PDB
PDB Universal> Skipping PDB processing
```

Click OK.

A Program Database (PDB) file contains information intended to help the debugger and user debug the file. PDB files, sometimes referred to as "symbol files," were developed by Microsoft and are "typically created from source files during compilation" (docs.microsoft.com). They store a list of "symbols in a module with their addresses and possibly the name of the file and the line on which the symbol was declared" (docs.microsoft.com).

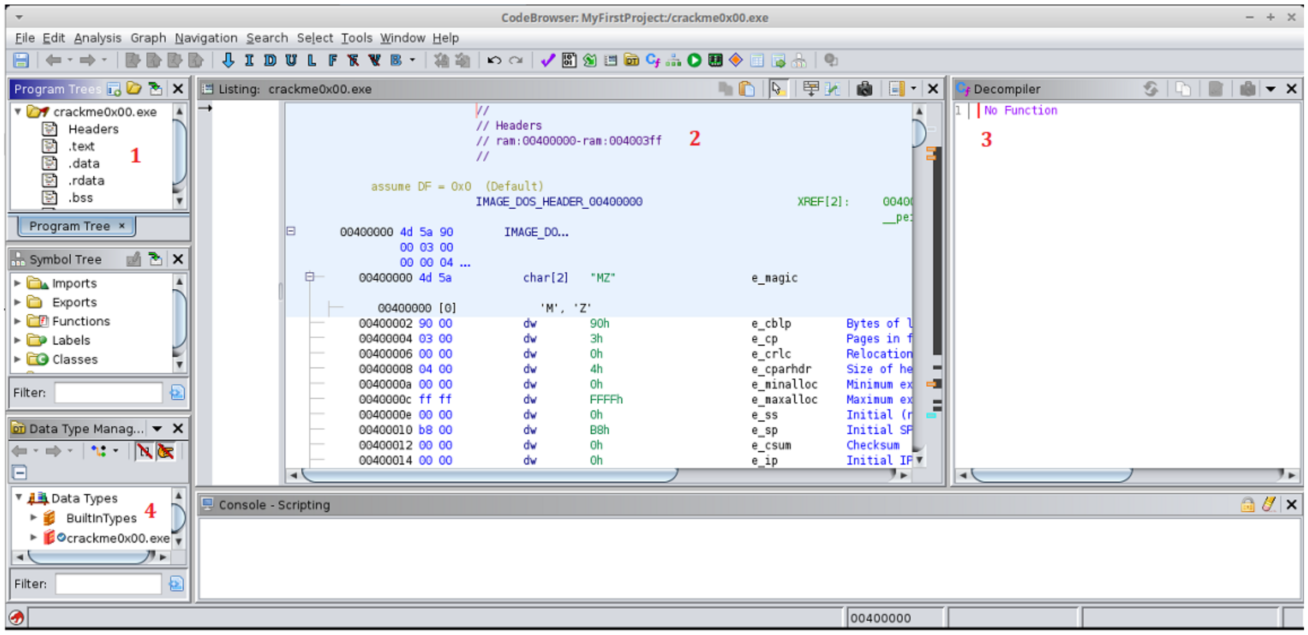
Since we specified that our PE file was compiled using Visual Studio in our import file step, Ghidra automatically searches for a PDB file associated with our executable when we run our "Auto Analysis."

To handle this error, we can uncheck the PDB box before we run our Auto Analysis:



| Enabled | Analyzer Name |
|---|---|
| ✓ | Embedded Media |
| ✓ | External Entry References |
| ✓ | Function ID |
| ✓ | Function Start Search |
| ✓ | Function Start Search After Code |
| ✓ | Function Start Search After Data |
| ✓ | Non-Returning Functions - Discovered |
| ✓ | Non-Returning Functions - Known |
|  | PDB MSDIA |
|  | PDB Universal |
| ✓ | Reference |
| ✓ | Scalar Operand References |
| ✓ | Shared Return Calls |
| ✓ | Stack |
| ✓ | Subroutine References |
| ✓ | Windows x86 PE Exception Handling |
| ✓ | Windows x86 PE RTTI Analyzer |
| ✓ | WindowsPE x86 Propagate External ... |
| ✓ | WindowsResourceReference |
| ✓ | x86 Constant Reference Analyzer |
| ✓ | X86 Function Callee Purge |

Ghidra will now analyze your file and display the information as seen in the five windows below.



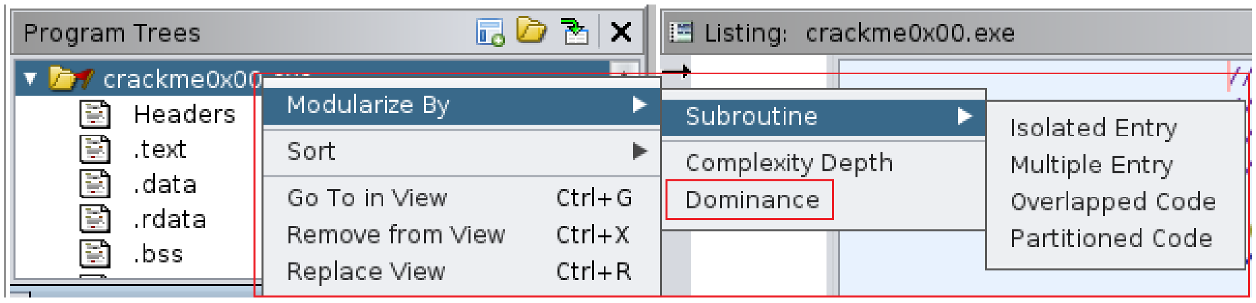
```
                                //
                                // Headers
                                // ram:00400000-ram:004003ff
                                //
                       assume DF = 0x0  (Default)
                                IMAGE_DOS_HEADER_00400000                       XREF[2]:     0040
                                                                                             __pe
       00400000 4d 5a 90            IMAGE_DO...
                00 03 00
                00 00 04 ...
          00400000 4d 5a            char[2]    "MZ"                  e_magic
             00400000 [0]              'M', 'Z'
          00400002 90 00            dw         90h                   e_cblp        Bytes of l
          00400004 03 00            dw         3h                    e_cp          Pages in f
          00400006 00 00            dw         0h                    e_crlc        Relocation
          00400008 04 00            dw         4h                    e_cparhdr     Size of he
          0040000a 00 00            dw         0h                    e_minalloc    Minimum ex
          0040000c ff ff            dw         FFFFh                 e_maxalloc    Maximum ex
          0040000e 00 00            dw         0h                    e_ss          Initial (r
          00400010 b8 00            dw         B8h                   e_sp          Initial SP
          00400012 00 00            dw         0h                    e_csum        Checksum
          00400014 00 00            dw         0h                    e_ip          Initial IP
```

## The left Windowpane

## The Program Trees Manager

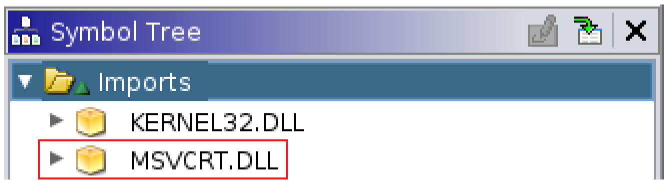
The program Trees Manager is used to organize programs into a tree-like structure. In addition, the Program Tree Manager allows you to create, delete, rename, and close program tree views.

Within “Program Trees,” you can right-click on the “crackme0x00” folder to organize the sections of disassembly code in different ways. You can do this by selecting “Modularize By” and choosing “Subroutine,” “Complexity Depth,” or “Dominance.” You can also make new folders and drag/drop sections according to your organizational preferences. For example, under Subroutine, choose Dominance.



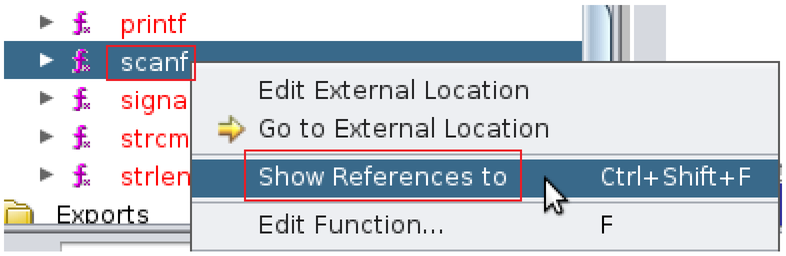
## The Symbol Tree Window

Directly below the "Program Trees" is the "Symbol Tree." Within this window, you are shown all the different symbols used with the program itself. The different symbols are shown as folders labeled imports, exports, functions, labels, classes, and namespaces. Try expanding the "Imports" section to see the various DLLs and functions used by the target.

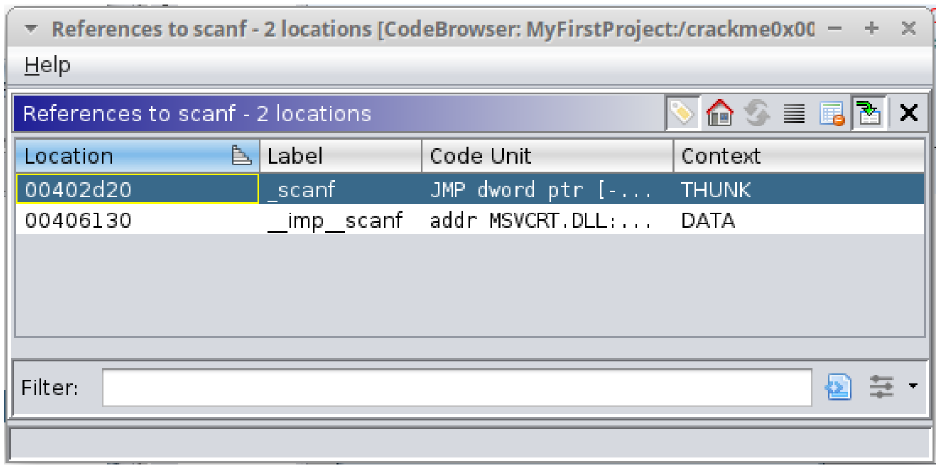


Open MSVCRT.DLL under imports. Scroll down until you come to function, scanf.

Right-click on the function and select "Show references to"



Another window pops up showing what the scanf function references.


References to scanf - 2 locations [CodeBrowser: MyFirstProject:/crackme0x00
Help
References to scanf - 2 locations
Filter:

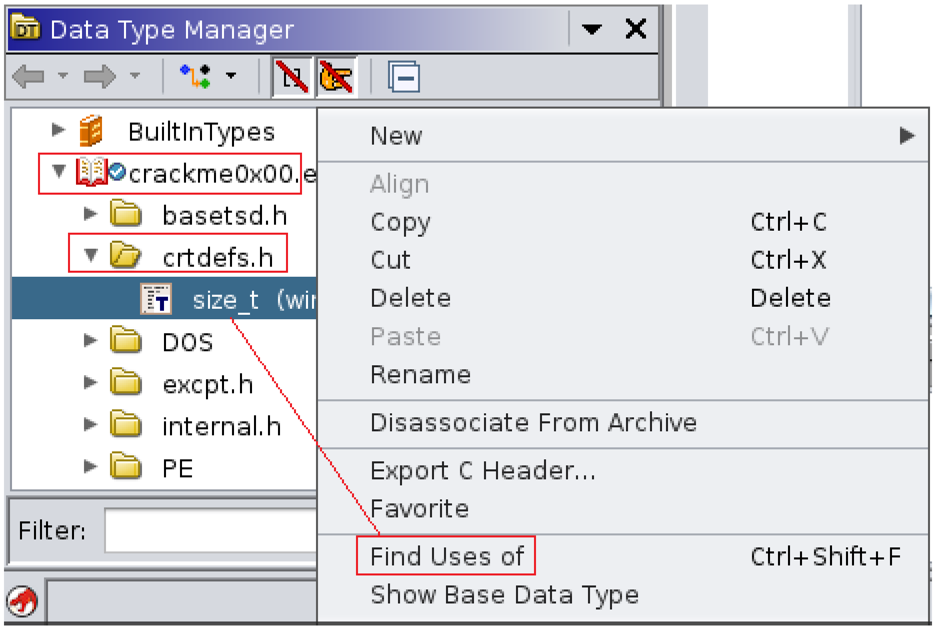
| Location | Label | Code Unit | Context |
|---|---|---|---|
| 00402d20 | _scanf | JMP dword ptr [-... | THUNK |
| 00406130 | __imp__scanf | addr MSVCRT.DLL:... | DATA |

This window displays the breakdown of the code in assembler language.

**The Data Type Manager Window**

Below the Symbol Tree window, we have Data Type Manager. The "Data Type Manager" allows you to see all the defined types, including the built-in types, those specific to the binary, and others that were included with Ghidra (such as the Windows ones we see called "windows_vs12_32").

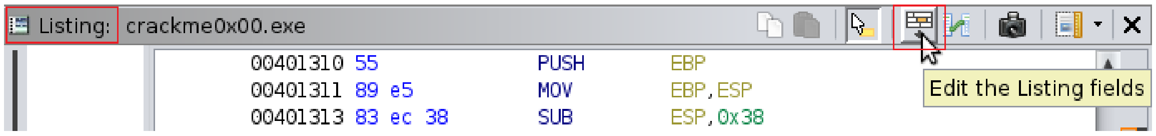
By expanding the book icon labeled crackme0x00.exe, right-clicking on a data type **size_t (windows_vs12_32),** and selecting “Find uses of,” we can see where a data type is being used within the binary.



**The Main Windows**

**The Listing Window**

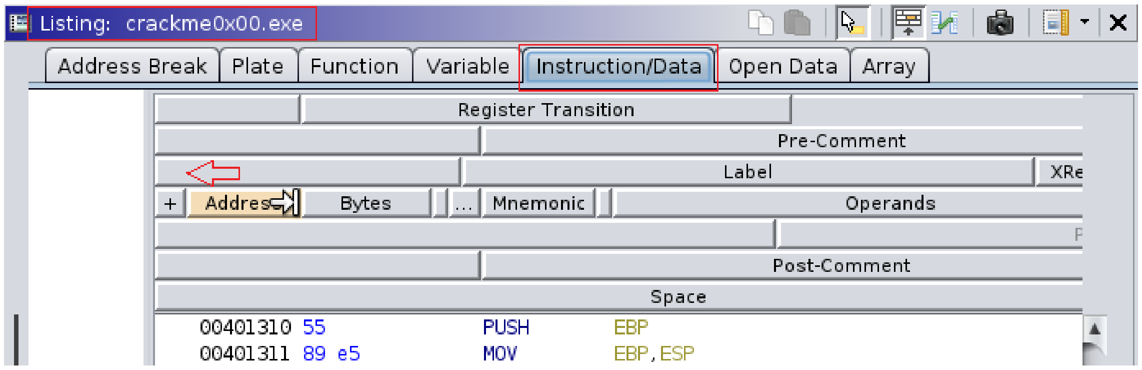
The Listing Window shows us the disassembled code and allows us to begin piecing together what different portions of the binary are doing. To customize our listing window view, we can click on the “Edit the listing fields” icon in the top right.



```
00401310 55        PUSH   EBP
00401311 89 e5     MOV    EBP,ESP
00401313 83 ec 38  SUB    ESP,0x38
```

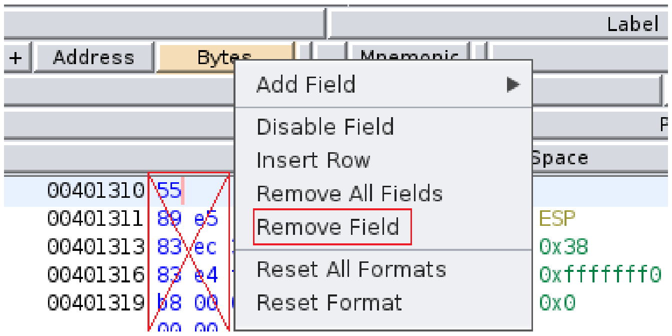
Clicking on the “Instruction/Data” tab gives us access to each element shown in the listing interface. These elements can be re-sized, moved around, disabled, or deleted. You can also add new elements by right-clicking and using the contextual menu.

Try re-sizing the “Address” field to make it smaller.



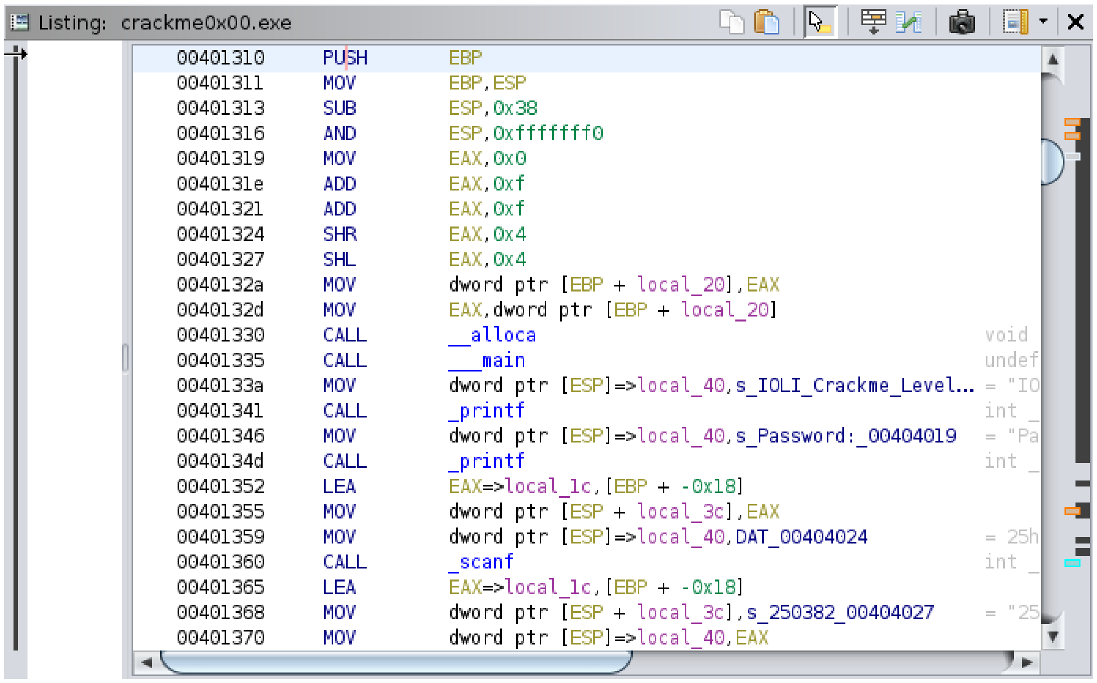
```
00401310 55             PUSH       EBP
00401311 89 e5          MOV        EBP,ESP
```

Delete the “Bytes” field.



```
00401310 55
00401311 89 e5          ESP
00401313 83 ec          0x38
00401316 83 e4          0xfffffff0
00401319 b8 00          0x0
```

Close out the editor and return to your Listing Window. The address field has been moved to the left, and the bytes field is no longer present.



```
00401310    PUSH    EBP
00401311    MOV     EBP,ESP
00401313    SUB     ESP,0x38
00401316    AND     ESP,0xfffffff0
00401319    MOV     EAX,0x0
0040131e    ADD     EAX,0xf
00401321    ADD     EAX,0xf
00401324    SHR     EAX,0x4
00401327    SHL     EAX,0x4
0040132a    MOV     dword ptr [EBP + local_20],EAX
0040132d    MOV     EAX,dword ptr [EBP + local_20]
00401330    CALL    __alloca                                        void
00401335    CALL    ___main                                         undef
0040133a    MOV     dword ptr [ESP]=>local_40,s_IOLI_Crackme_Level...  = "IO
00401341    CALL    _printf                                         int _
00401346    MOV     dword ptr [ESP]=>local_40,s_Password:_00404019  = "Pa
0040134d    CALL    _printf                                         int _
00401352    LEA     EAX=>local_1c,[EBP + -0x18]
00401355    MOV     dword ptr [ESP + local_3c],EAX
00401359    MOV     dword ptr [ESP]=>local_40,DAT_00404024          = 25h
00401360    CALL    _scanf                                          int _
00401365    LEA     EAX=>local_1c,[EBP + -0x18]
00401368    MOV     dword ptr [ESP + local_3c],s_250382_00404027    = "25
00401370    MOV     dword ptr [ESP]=>local_40,EAX
```

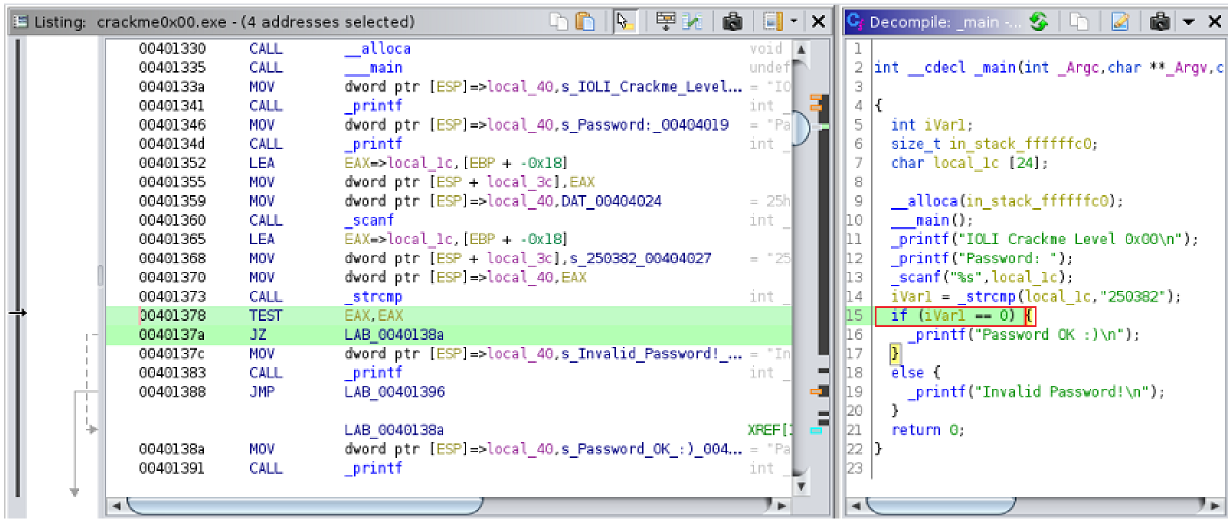
## Contextual Editor

By right-clicking anywhere within the assembly code shown in the Listing Window, the contextual menu within the disassembly listing can be seen. The contextual menu allows you to perform patching instructions, setting a bookmark, commenting, and editing labels.

## Decompile Window

To the right of the Listing Window, we have the  Decompile window. The Decompile Window shows us Ghidra's best estimation of the high-level code representing the assembly code in the listing/function graph windows.

Highlight one of the "if" statements shown in the decompile window. In the Listings Window, you will see that it highlights the corresponding assembly. This feature allows you to build a mental mapping of what groups of assembly instructions map to which high-level instructions.



```
00401330    CALL    __alloca                                        void
00401335    CALL    ___main                                         undef
0040133a    MOV     dword ptr [ESP]=>local_40,s_IOLI_Crackme_Level... = "IO
00401341    CALL    _printf                                         int
00401346    MOV     dword ptr [ESP]=>local_40,s_Password:_00404019  = "Pa
0040134d    CALL    _printf                                         int
00401352    LEA     EAX=>local_1c,[EBP + -0x18]
00401355    MOV     dword ptr [ESP + local_3c],EAX
00401359    MOV     dword ptr [ESP]=>local_40,DAT_00404024          = 25h
00401360    CALL    _scanf                                          int
00401365    LEA     EAX=>local_1c,[EBP + -0x18]
00401368    MOV     dword ptr [ESP + local_3c],s_250382_00404027    = "25
00401370    MOV     dword ptr [ESP]=>local_40,EAX
00401373    CALL    _strcmp                                         int
00401378    TEST    EAX,EAX
0040137a    JZ      LAB_0040138a
0040137c    MOV     dword ptr [ESP]=>local_40,s_Invalid_Password!_... = "In
00401383    CALL    _printf                                         int
00401388    JMP     LAB_00401396

                    LAB_0040138a                                    XREF[
0040138a    MOV     dword ptr [ESP]=>local_40,s_Password_OK_:)_004... = "Pa
00401391    CALL    _printf                                         int
```

```c
int __cdecl _main(int _Argc,char **_Argv,c

{
  int iVar1;
  size_t in_stack_ffffffc0;
  char local_1c [24];

  __alloca(in_stack_ffffffc0);
  ___main();
  _printf("IOLI Crackme Level 0x00\n");
  _printf("Password: ");
  _scanf("%s",local_1c);
  iVar1 = _strcmp(local_1c,"250382");
  if (iVar1 == 0) {
    _printf("Password OK :)\n");
  }
  else {
    _printf("Invalid Password!\n");
  }
  return 0;
}
```

The Decompile Window allows you to see what the high-level language would most likely look like.

## Summary

Reverse engineering malware is one of the highest-level skillsets within cybersecurity and one of the highest-paid. Ghidra is an excellent reverse engineering tool capable of running on nearly any platform and priced very attractively (free). In this series on Reverse Engineering, we will be using this tool from the US NSA to reverse engineer multiple pieces of malware, beginning with the simple and progressing to the more advanced.