Ghidra

New reversing tool for RE ninjas

Installing Ghidra

- Unzip the Ghidra installation file
- Platform supported
  - Microsoft Windows 7 or 10 (64 bit)
  - Linux (64-bit, CentOS 7 is preferred)
  - macOS (OS X) 10.8.3+ (Mountain Lion or later)

What is Ghidra

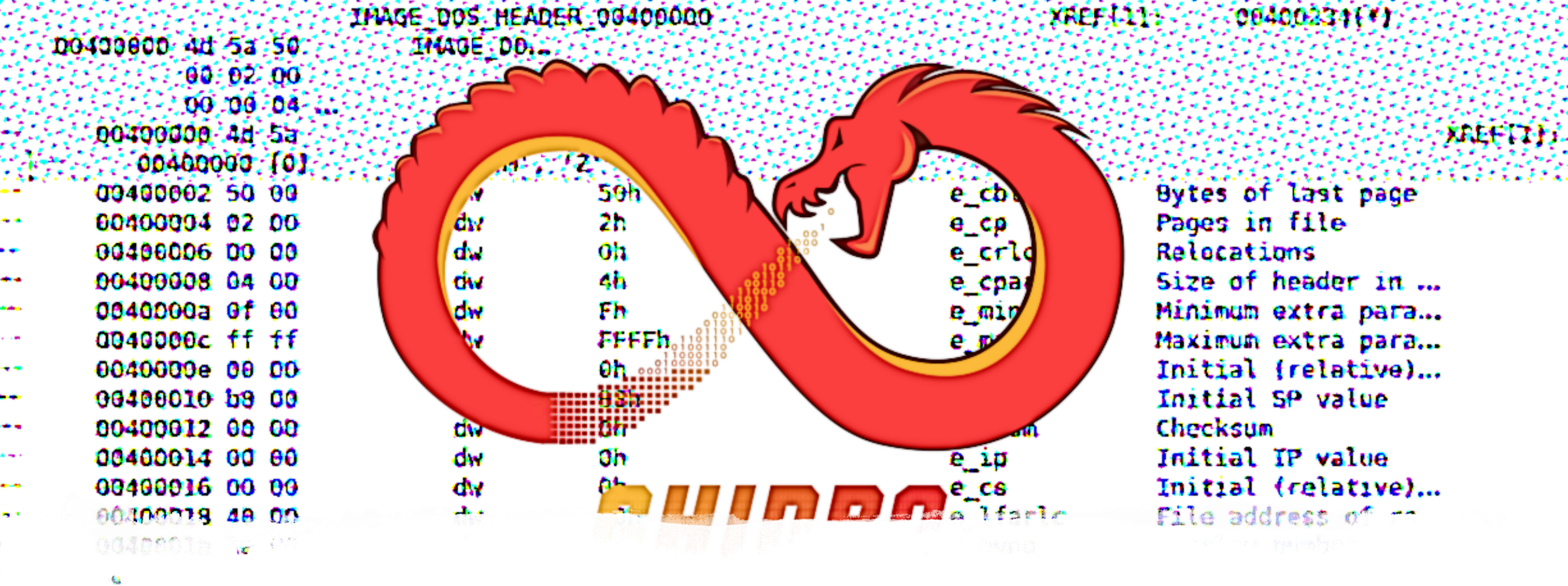
- A software reverse engineering (SRE) framework
- Full-featured, high-end software analysis tools that enable users to analyze compiled code on a variety of platforms
- Written in Java - mostly platform independent

Capabilities

- Disassembly
- Decompilation
- Graphing
- Scripting
- Etc..

Capabilities (cont.)

- Ghidra supports a wide variety of process instruction sets and executable formats.
- Users may also develop their own Ghidra plug-in or script
  - Using exposed API



Today topic

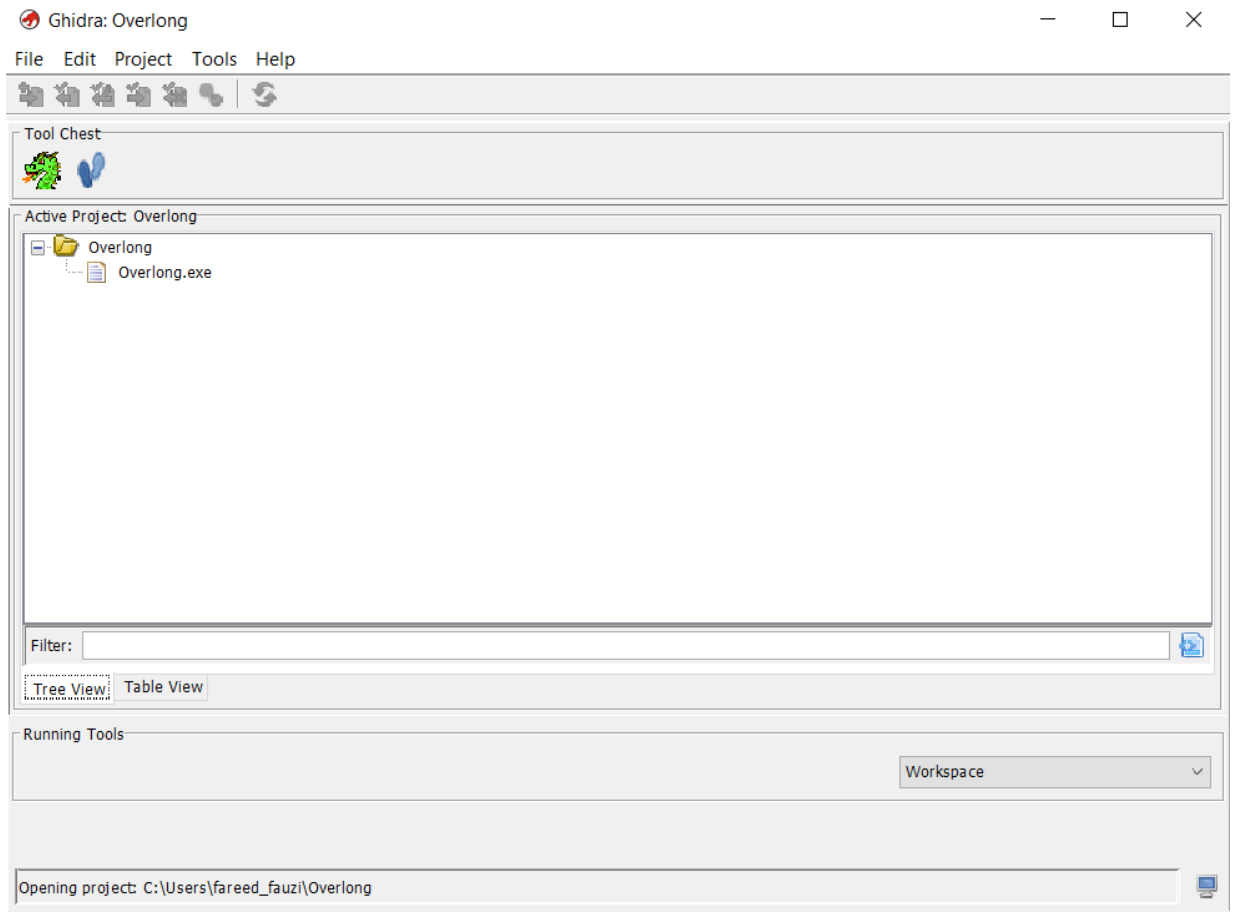
- I won't be able to cover every single feature
- I will focus on the most important and useful for reverse engineering

Starting Ghidra

- Run the following from the Ghidra installation directory
  - ghidraRun.bat (Windows)
- The first time you run Ghidra you may be asked for the path to your Java installation.

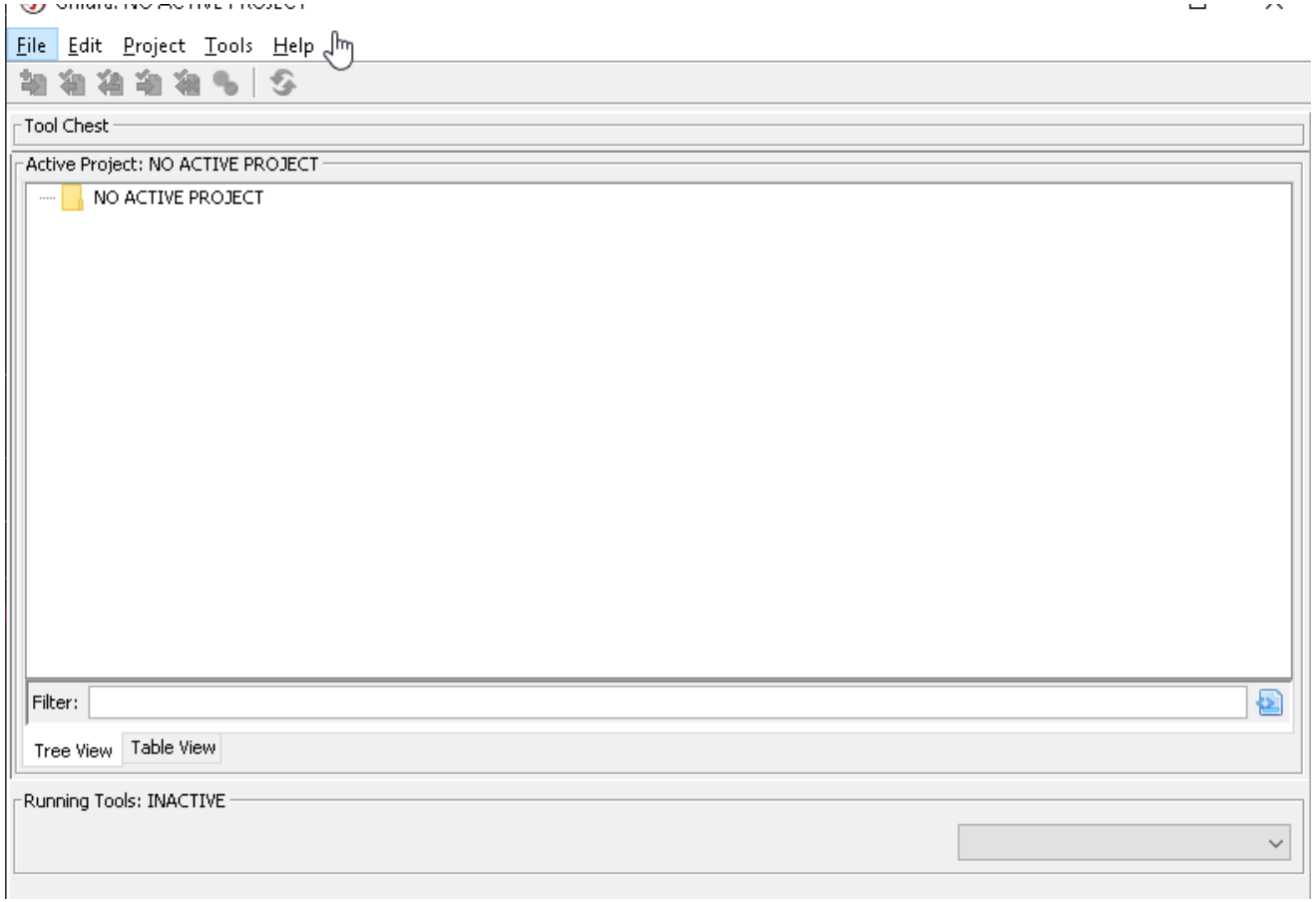
docs	3/25/2019 6:16 PM	File folder
Extensions	3/25/2019 6:16 PM	File folder
Ghidra	3/25/2019 6:16 PM	File folder
GPL	3/25/2019 6:16 PM	File folder
licenses	3/25/2019 6:16 PM	File folder
server	3/25/2019 6:16 PM	File folder
support	3/25/2019 6:16 PM	File folder
ghidraRun	3/25/2019 6:16 PM	File
ghidraRun - Shortcut	4/2/2019 8:45 AM	Shortcut
ghidraRun.bat	3/25/2019 6:16 PM	Windows Batch File
LICENSE.txt	3/25/2019 6:16 PM	Text Document

Program Manager Layout



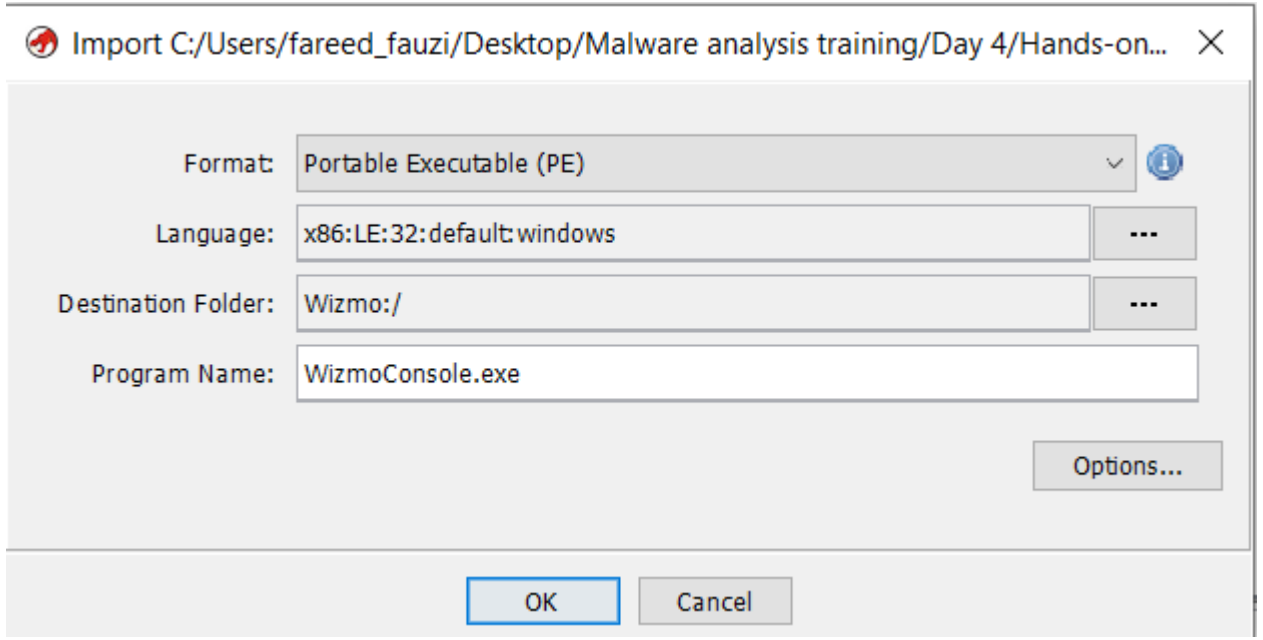
Program Manager Layout

- Everything is a project in Ghidra.
- Start by creating a project.
  - Unlike IDA, we start with an input file.
- On the first run, there are no projects and you are presented with this dialog:

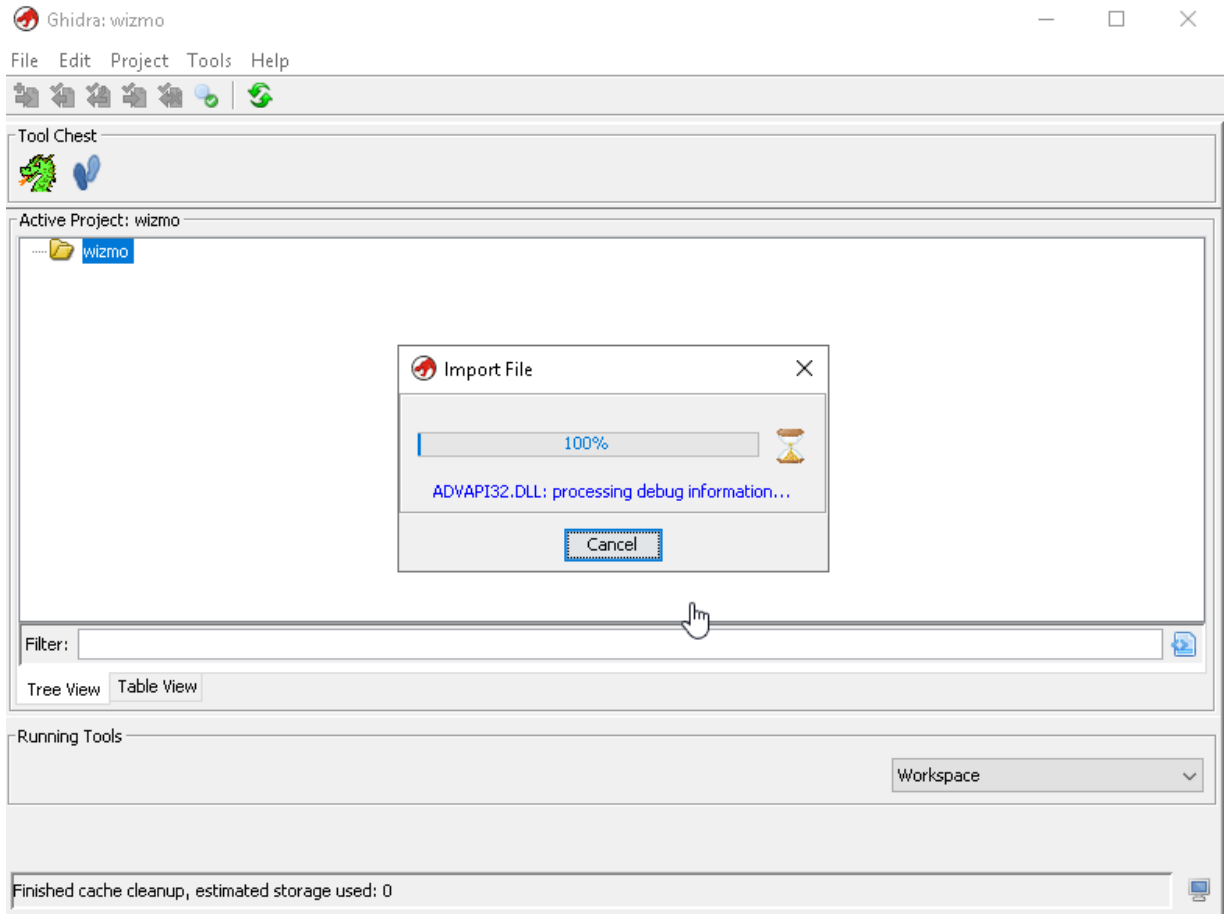


Project management

- Create a non shared project
  - File > Create project
- Create a project name
- Import a file
  - File > Import file
  - Click OK



Ghidra start to import the file

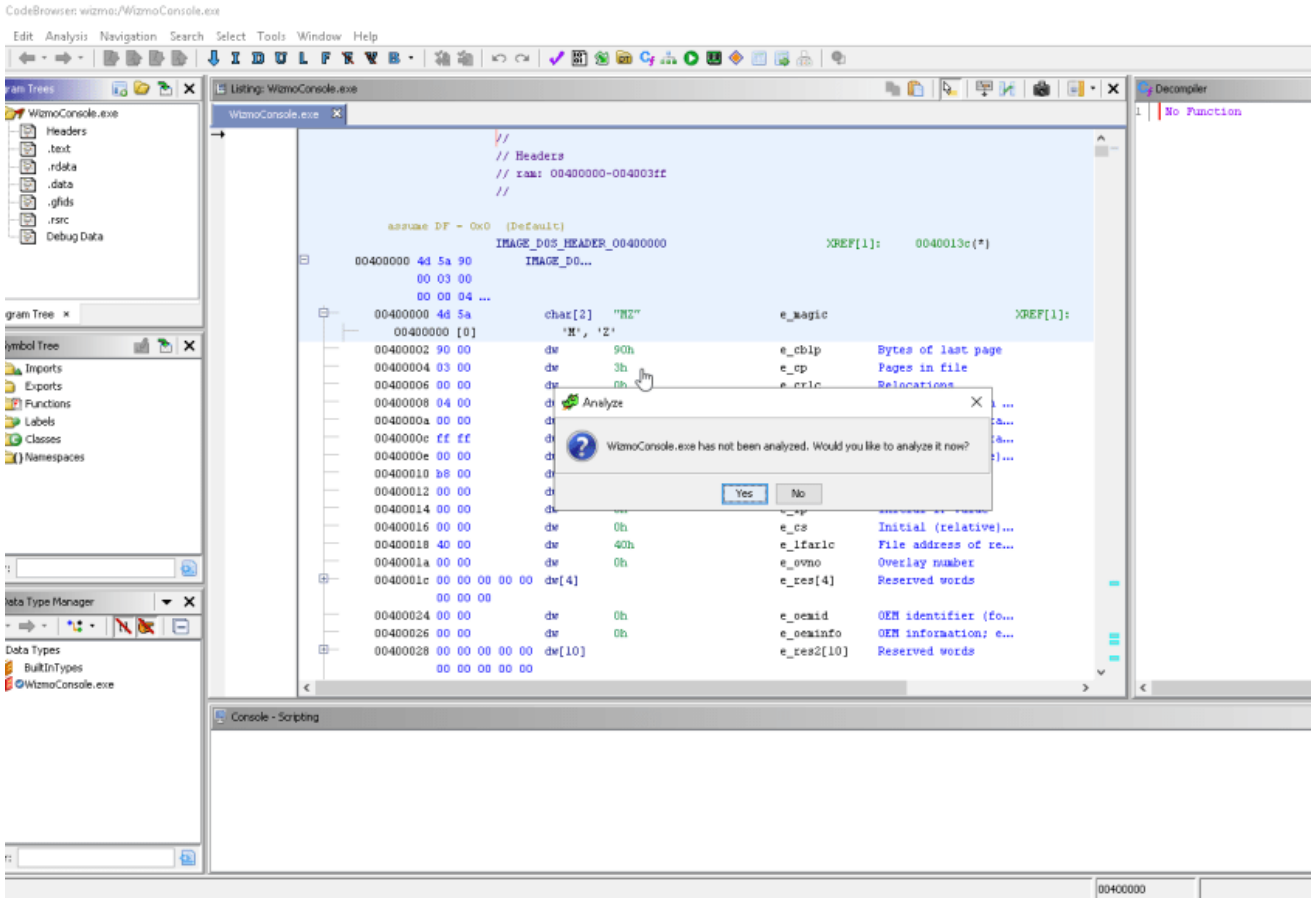


Import result

- After importing the file, you are presented with the import results summary dialog

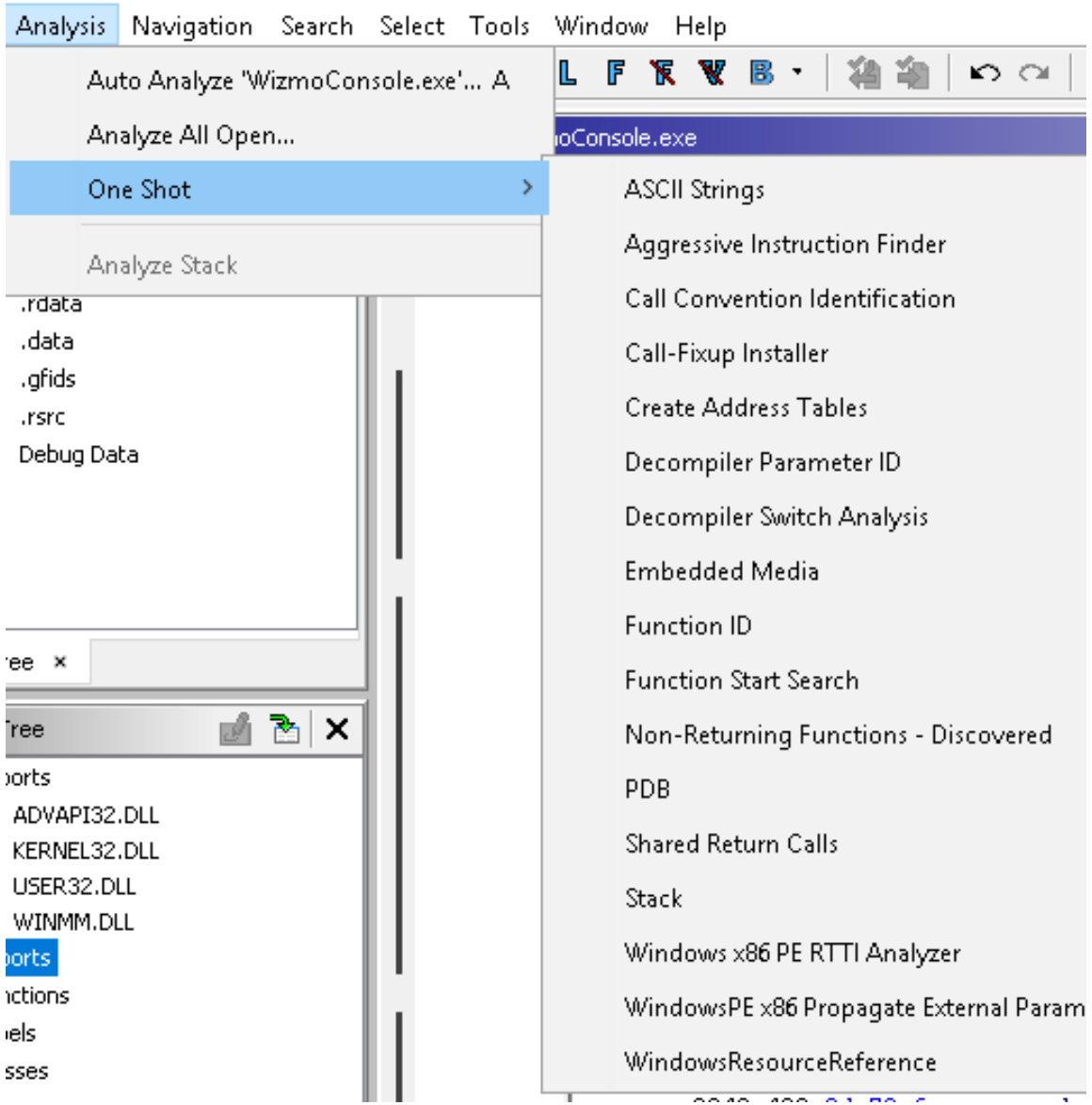
Analyze

- After you press “OK”, you get to see the code browser window and are asked whether you want to start analyzing the file:



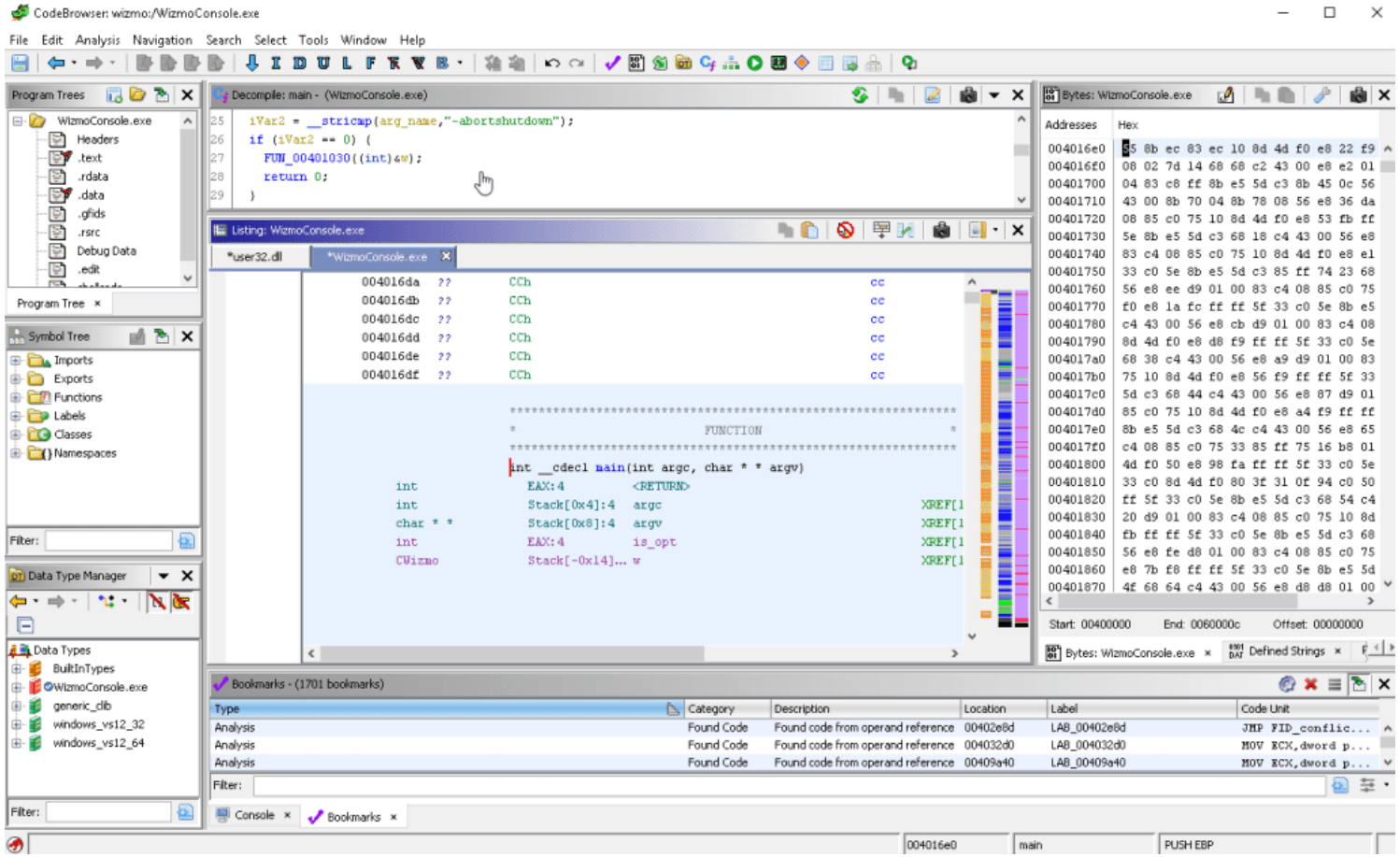
Analyze

- You can always analyze or re-analyze the file later from the "Analysis" menu:



The code browser

The code browser interface

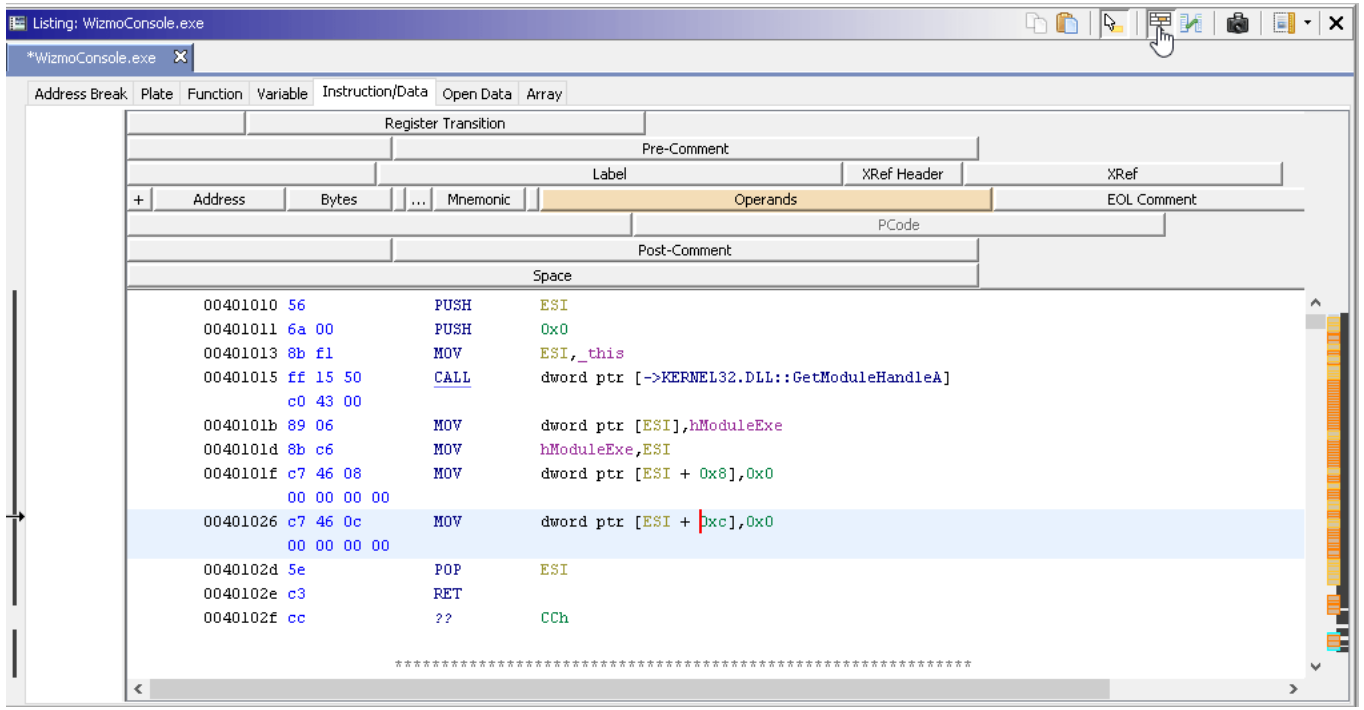


Code browser

- The code browser hosts all the visual elements of Ghidra:
  - The main menus
  - The disassembly view
  - Symbol tree
  - Program trees
  - Strings view
  - Data types manager
  - Decompiler view
  - etc.

Customize disassembly layout

- The program disassembly listing is highly customizable.
- Just press on the “Edit the listing fields” button (as indicated by the cursor) to see all the customization options.

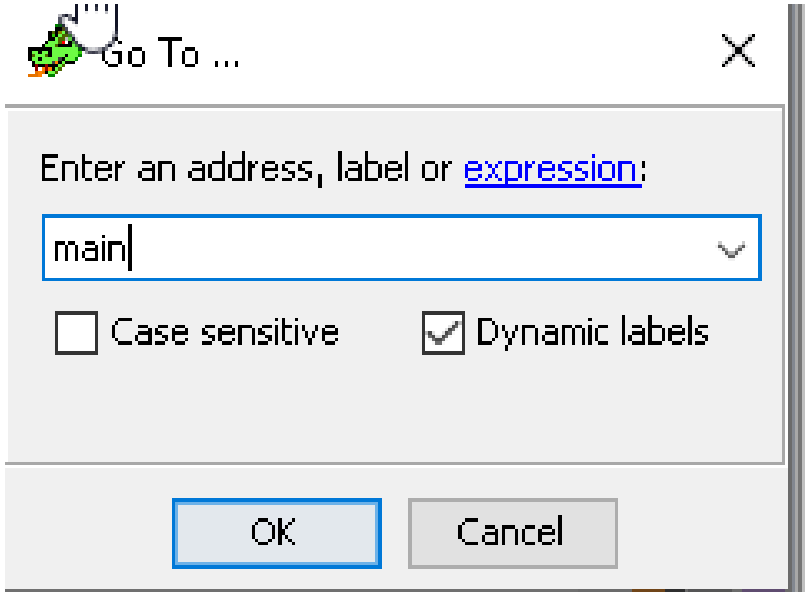


Customize disassembly layout

- Click and drag the fields to re-arrange the visual elements in the disassembly listing (disasm view) window.
- This advanced visual customization is also not available in IDA Pro.

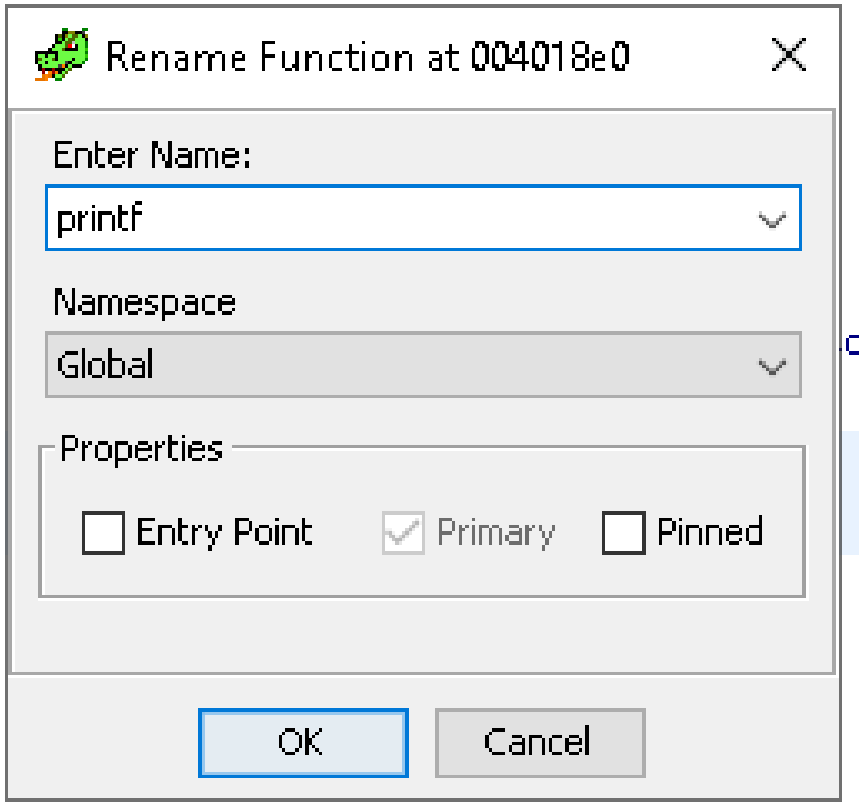
Jump to address or a label

- Inside the code browser disassembly listing, you can press “G” to jump to an address or a label:



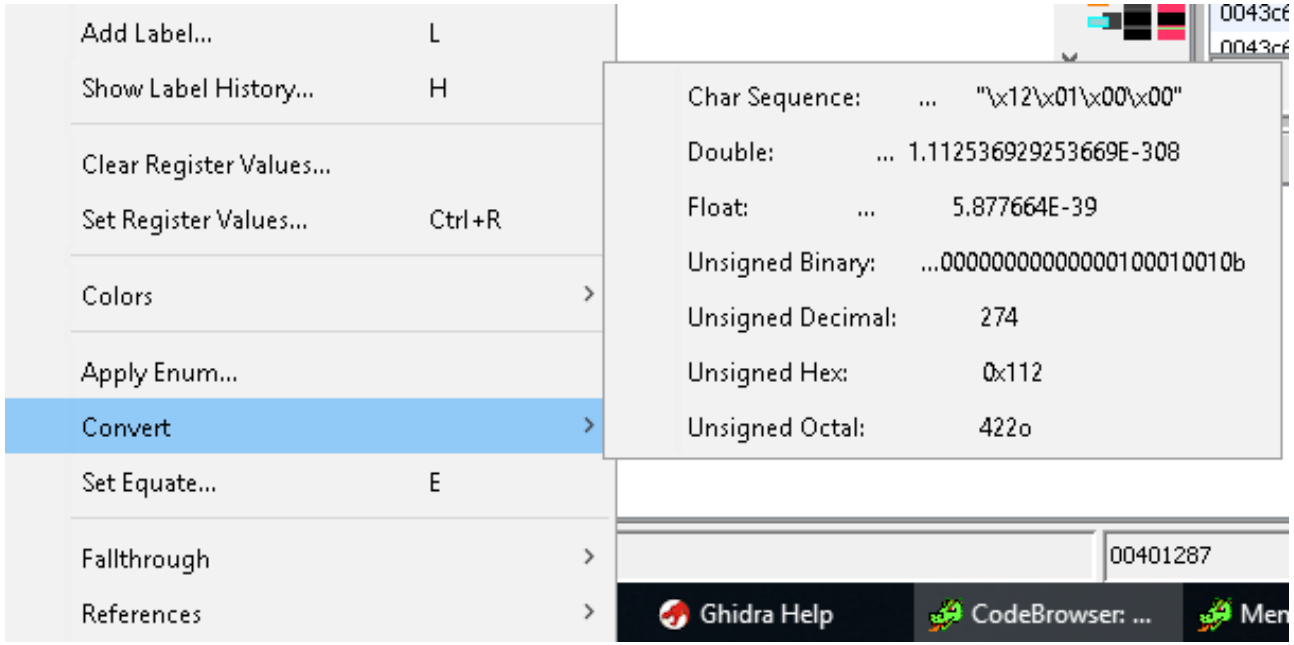
Rename function

- Simply rename a function or label by pressing L on the label we want



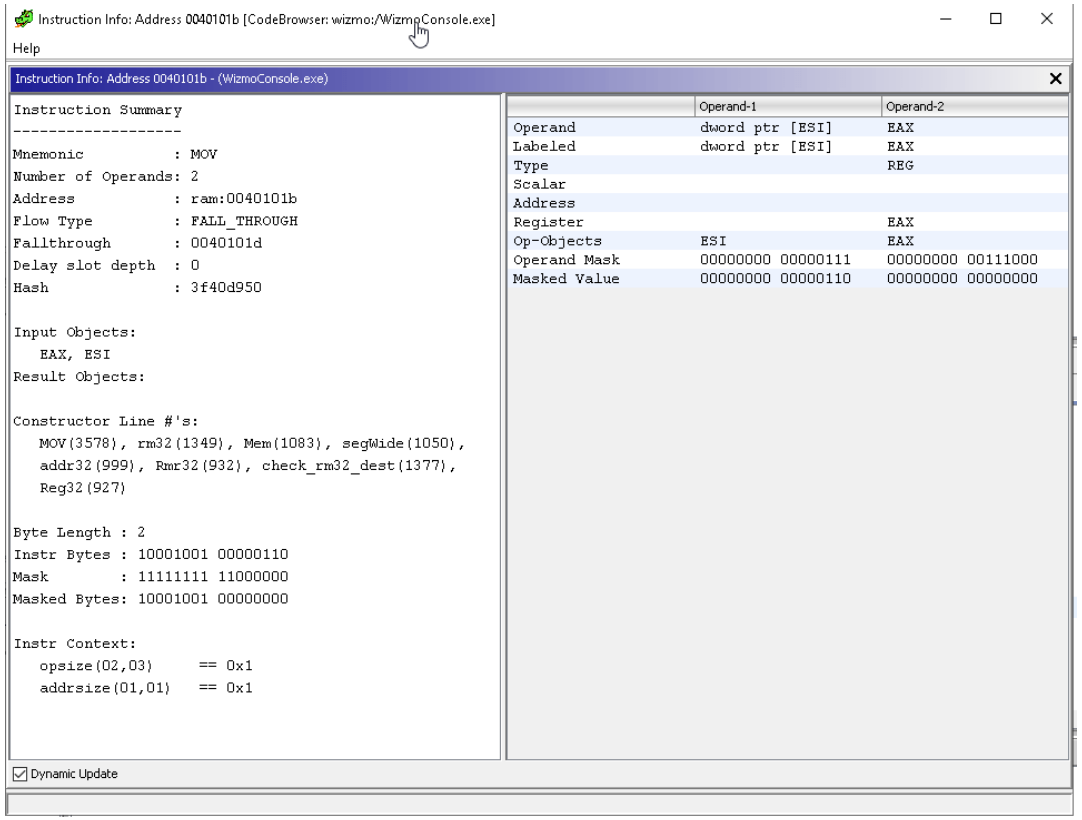
Numerical conversion

- You can also right-click on a number in the listing to convert it to another numerical representation:



Instruction info

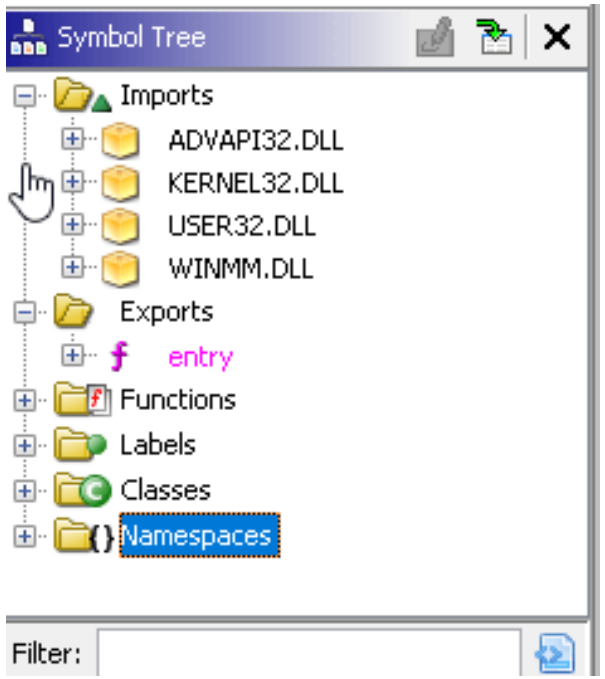
- To view information about an instruction in the code browser, just right click and select “Instruction Info”:



The symbol tree

Symbol tree

- The symbol tree window lets you see all the symbols in the program,
  - such as the exports, imports, classes, functions, labels, etc.



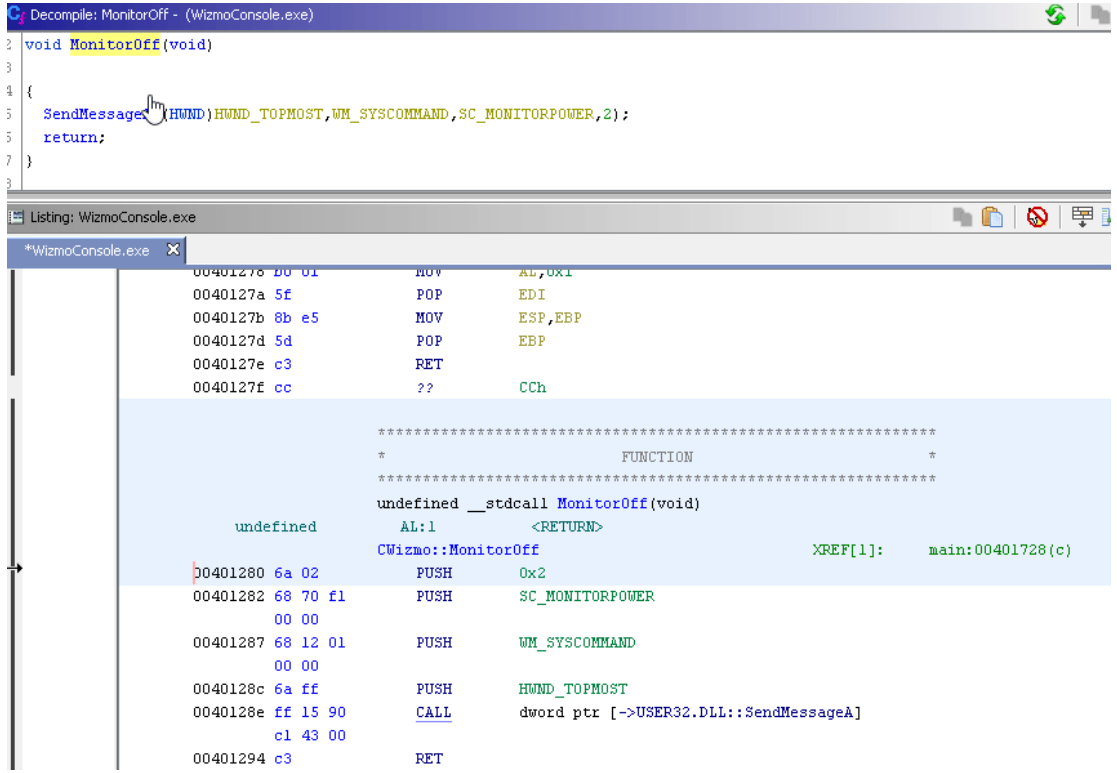
Symbol tree

- As you explore the imported entry, you can double-click to jump to it in the code browser.

The decompiler

The decompiler

- The decompiler is a neat and most welcome feature in Ghidra:



The decompiler

- You can toggle the decompiler view from the Window menu.
- It synchronizes with the disassembly listing.
- When you navigate in the decompiler view, you will see the corresponding disassembly lines in the listing window.

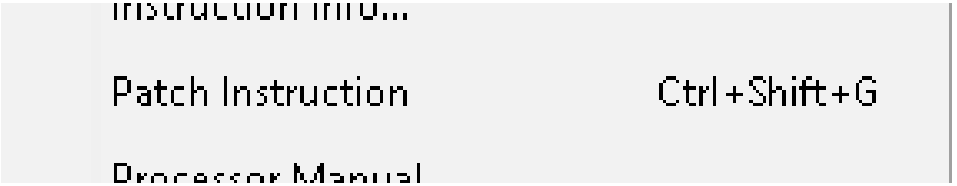
The decompiler

- Ghidra's decompiler is interactive and customizable:
  - Rename functions
  - Add comments
  - Change function prototypes
  - Change variable names and types
  - etc.

Code patching and the hex viewer

Code patching

- Ghidra provides lots of functionality to patch code and then save the patched result.
- To patch an instruction, just right click and select:



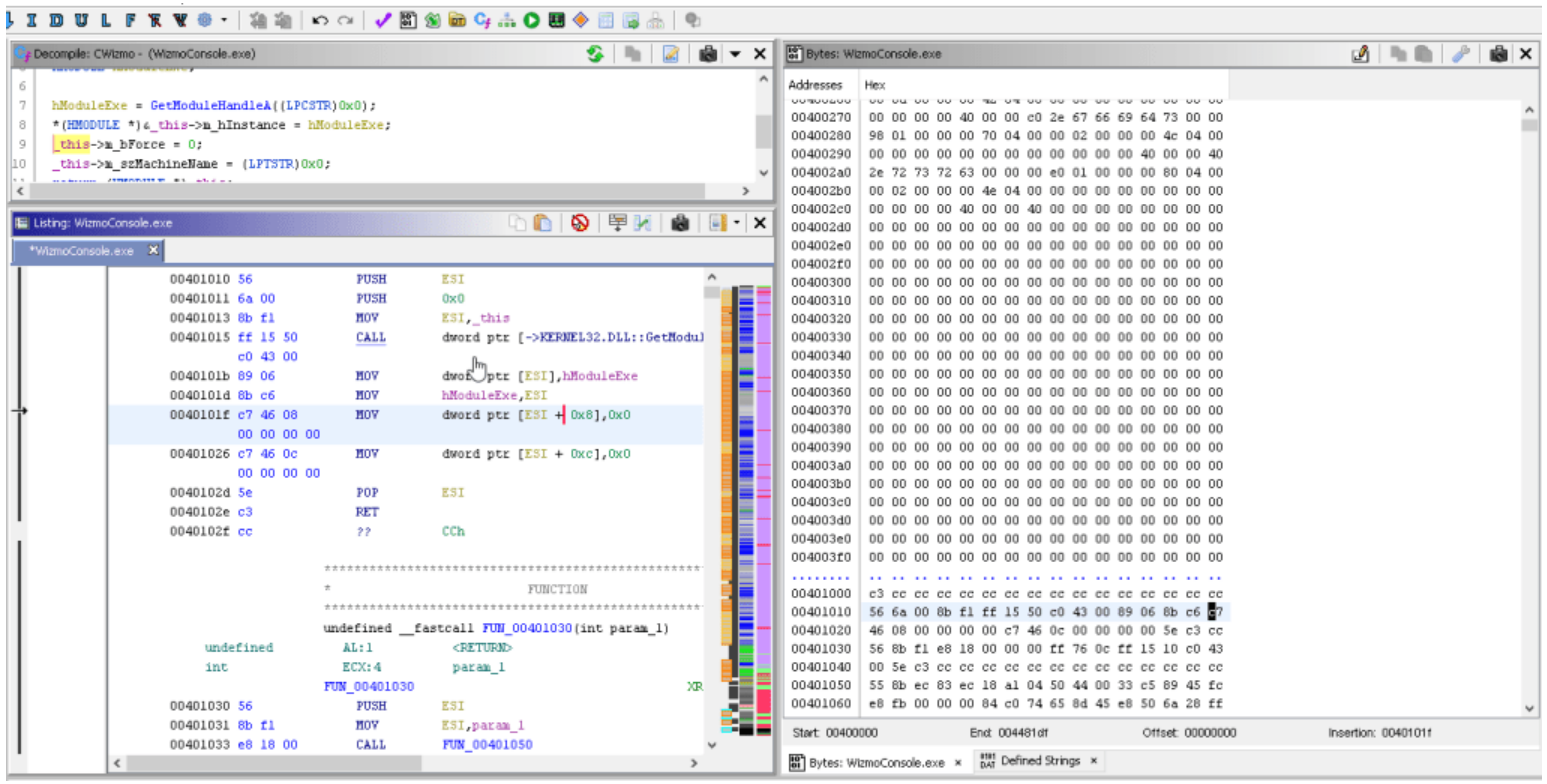
Code patching

- You will then be presented by an instruction editor / assembler:

```
0040101b 89 06          MOV     dword ptr [ESI],hModuleExe
0040101d 8b c6          MOV     hModuleExe,ESI
0040101f c7 46 08       MOV     dword ptr [ESI + 0x8],0x0
         00 00 00 00
00401026 c7 46 0c       MOV     dword ptr [ESI + 0xc],0x0          Hello world
         00 00 00 00
0040102d 5e             POP     ESI
```

Hex viewer

- Just toggle the hex view from the “Window > Bytes” menu:



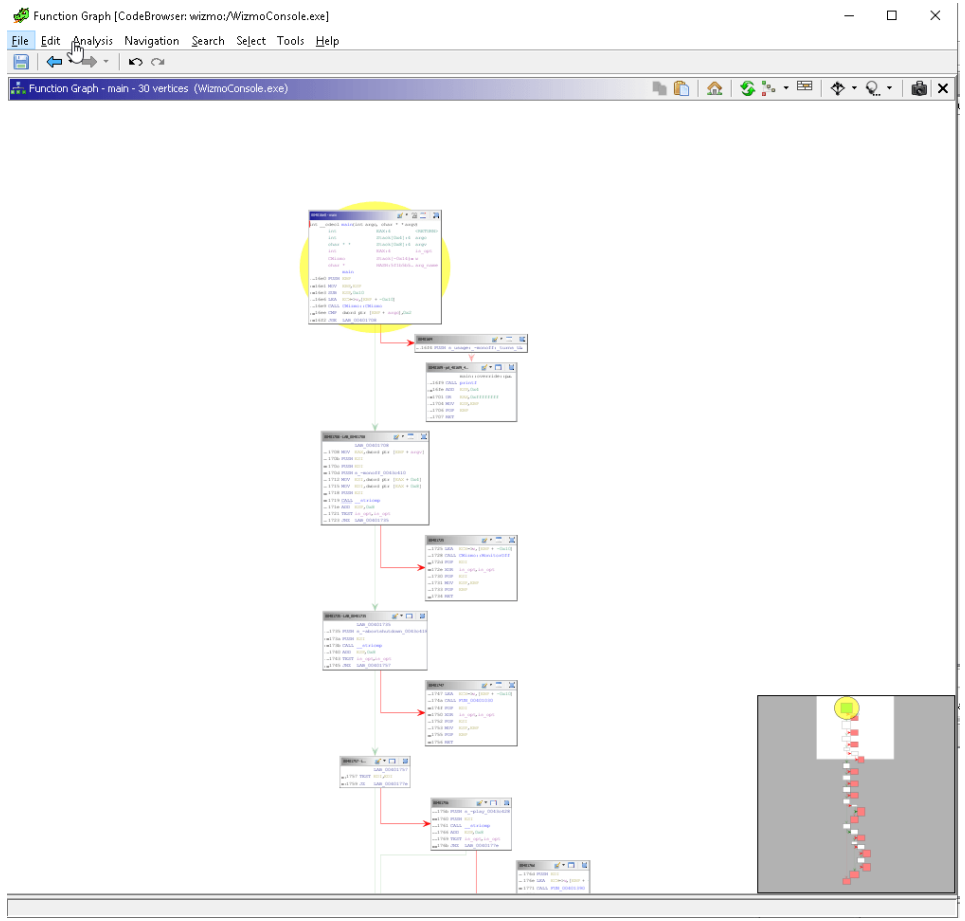
Graph view

Graph view

- Ghidra, like IDA also supports a graph view
- Combined with the facilities from the “Select” menu, the graph view becomes a powerful tool:

Graph view

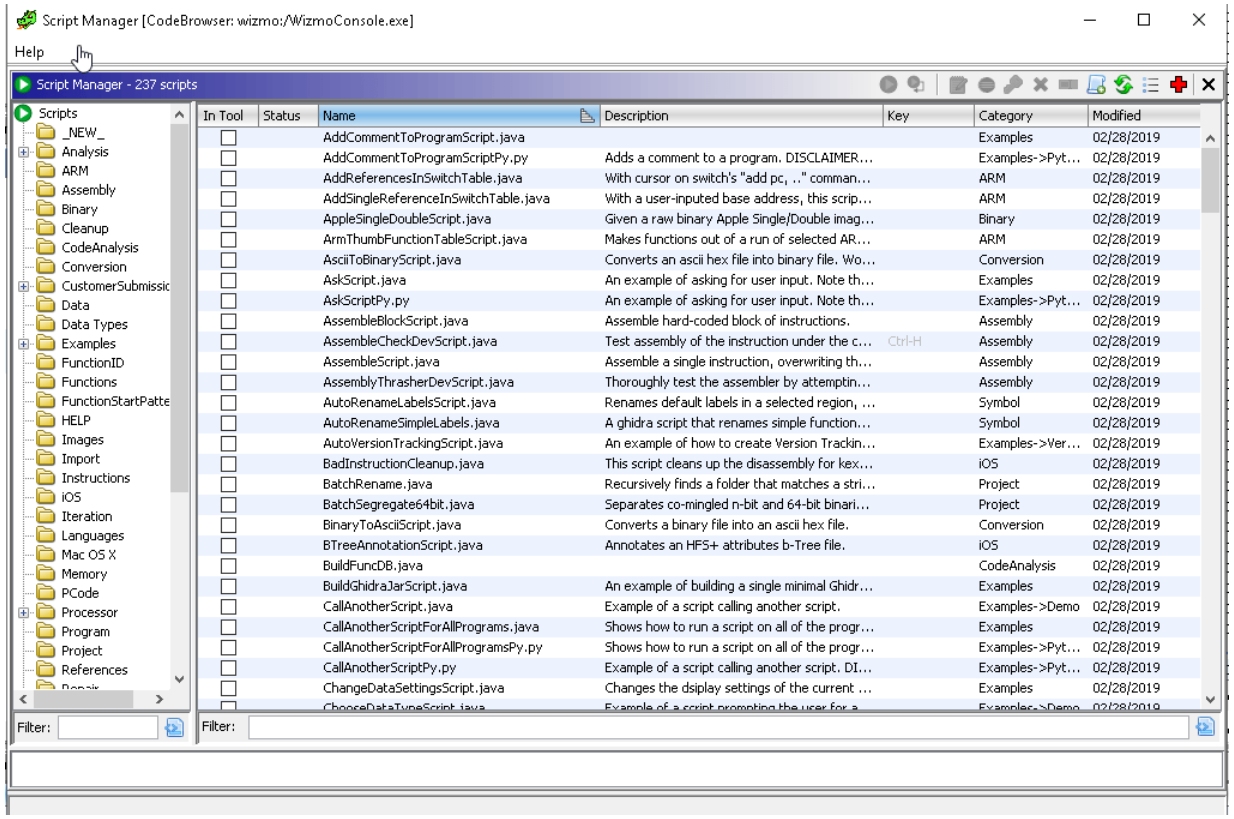
- Go to windows > Function graph



Scripting features

Scripting features

- Select scripting from the “Window > Script manager” menu
- Ghidra, out of the box, ships with 200+ scripts written in Java:



Miscellaneous features

Look for strings

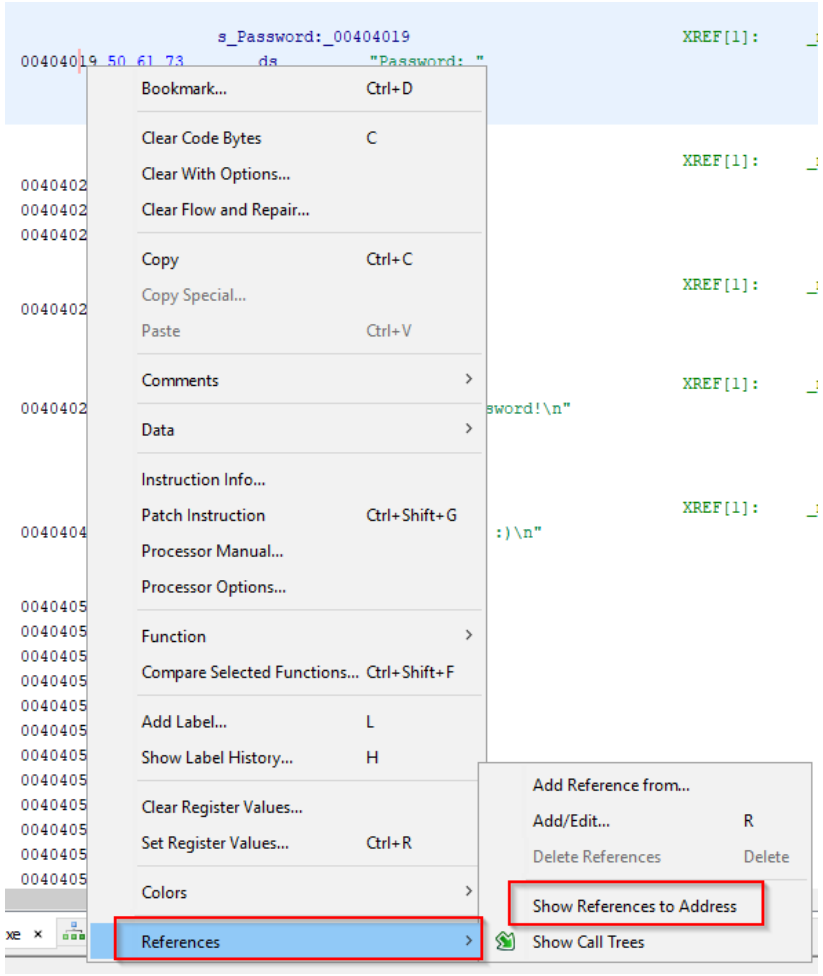
- Window > Defined strings

Defined Strings - 83 items

Location	String Value	String Representation	Data Type
.strtab::000000c3	__init_array_end	"__init_array_end"	ds
.strtab::000000d4	_DYNAMIC	"_DYNAMIC"	ds
.strtab::000000dd	__init_array_start	"__init_array_start"	ds
.strtab::000000f0	__GNU_EH_FRAME_HDR	"__GNU_EH_FRAME_HDR"	ds
.strtab::00000103	_GLOBAL_OFFSET_TABLE_	"_GLOBAL_OFFSET_TABLE_"	ds
.strtab::00000119	__libc_csu_fini	"__libc_csu_fini"	ds
.strtab::00000129	_ITM_deregisterTMClone...	"_ITM_deregisterTMClone...	ds
.strtab::00000145	puts@@GLIBC_2.2.5	"puts@@GLIBC_2.2.5"	ds
.strtab::00000157	_edata	"_edata"	ds
.strtab::0000015e	__stack_chk_fail@@GLIB...	"__stack_chk_fail@@GLIB...	ds
.strtab::0000017a	printf@@GLIBC_2.2.5	"printf@@GLIBC_2.2.5"	ds
.strtab::0000018e	__libc_start_main@@GLI...	"__libc_start_main@@GLI...	ds
.strtab::000001ad	__data_start	"__data_start"	ds
.strtab::000001ba	strcmp@@GLIBC_2.2.5	"strcmp@@GLIBC_2.2.5"	ds
.strtab::000001ce	__gmon_start__	"__gmon_start__"	ds
.strtab::000001dd	__dso_handle	"__dso_handle"	ds
.strtab::000001ea	_IO_stdin_used	"_IO_stdin_used"	ds
.strtab::000001f9	__libc_csu_init	"__libc_csu_init"	ds
.strtab::00000209	__bss_start	"__bss_start"	ds
.strtab::00000215	main	"main"	ds
.strtab::0000021a	_Jv_RegisterClasses	"_Jv_RegisterClasses"	ds
.strtab::0000022e	__isoc99_scanf@@GLIBC...	"__isoc99_scanf@@GLIB...	ds
.strtab::00000248	__TMC_END__	"__TMC_END__"	ds
.strtab::00000254	_ITM_registerTMCloneTable	"_ITM_registerTMCloneTa...	ds
00400001	ELF	"ELF"	ds
00400238	/lib64/ld-linux-x86-64.so.2	"/lib64/ld-linux-x86-64.so.2"	ds
00400379	libc.so.6	"libc.so.6"	ds
00400383	__isoc99_scanf	"__isoc99_scanf"	ds
00400392	puts	"puts"	ds
00400397	__stack_chk_fail	"__stack_chk_fail"	ds
004003a8	printf	"printf"	ds
004003af	strcmp	"strcmp"	ds
004003b6	__libc_start_main	"__libc_start_main"	ds
004003c8	__gmon_start__	"__gmon_start__"	ds

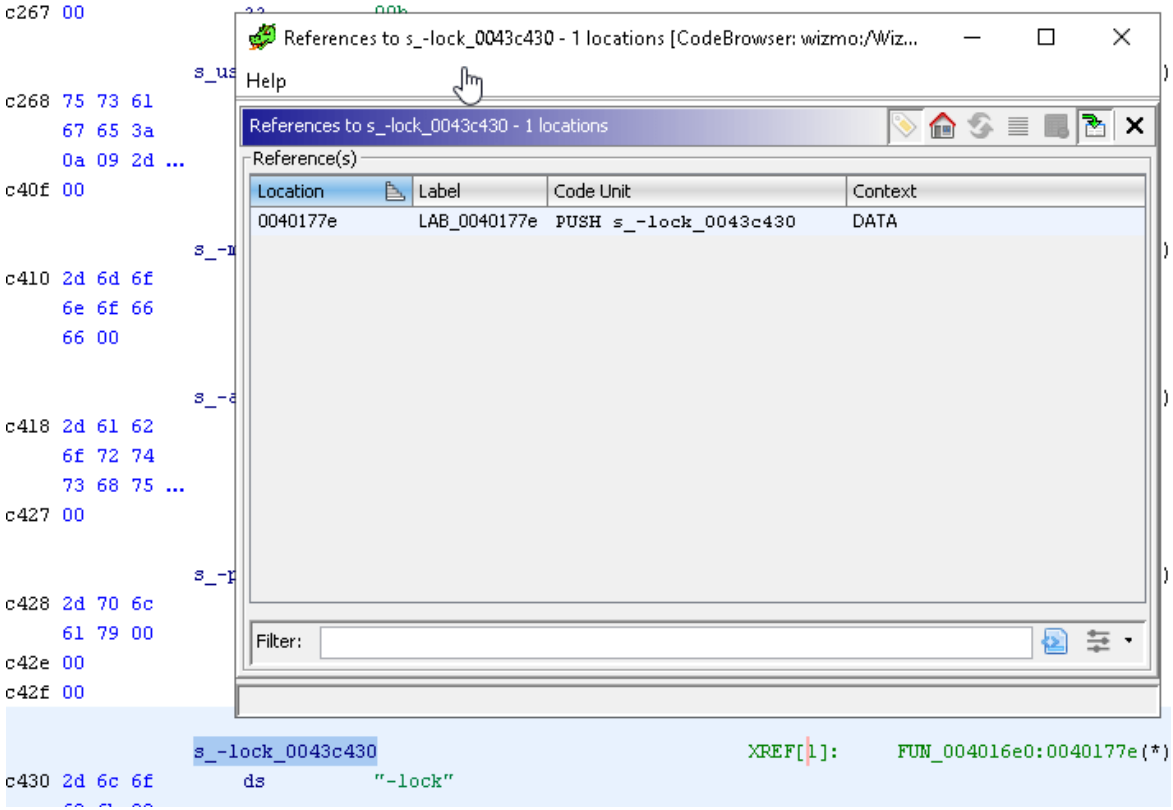
Cross reference

- Simply right click > References > Show References to Address



Strings cross referencing

- With strings cross referencing, you can discover malicious strings or locate the code that refers / implements certain features



Conclusion

- Ghidra may good on customization and decompilation
- It has no debugger tool
- It little bit slow
- But its open source and free!

Let’s do Crackme Challenges