Laboratory No. 7

Executable analysis using Ghidra

To test a program of this type can be used for a beginner an executable created in C and then in CPP for a simple program. After learning the basic skills the process will resume on programs of the same type somewhat more complicated and finally can pass on the analysis of an executable about which we have no consciences at all. Obviously, you need to refresh the ASM knowledge with orientation on the specific LINUX compiler (see also the course).

To install Ghidra requires JDK 11. Attention to the situation where we need depending on the types of applications installed by multiple JDKs simultaneously.

This is then brought from

https://ghidra-sre.org/ghidra_9.0.4_PUBLIC_20190516.zip

In case of difficulties in the installation process, consult

https://www.ghidra-sre.org/Installation Guide.html#Install.

The user directory is created (unless there is the eight directory in which the archive is then unpacked. It will run with ./guideRun from the console or you can create a shortcut of your choice.

What is Ghidra?

Although in the course we presented a number of alternatives for executable analysis we did not detail much Ghidra because it was chosen as an application for the laboratory. This is a free, relatively new tool in the market for high-profile utilities, starting from an older NSA project called "Vaut 7".

The application can be run on all dominant operating systems and contains all the skills needed to analyze and modify any executable. These aspects were analysed in the course so they are assumed to be known. A rare feature of such programmes is that it allows a cooperative approach of a team in carrying out the analysis for the same programme.

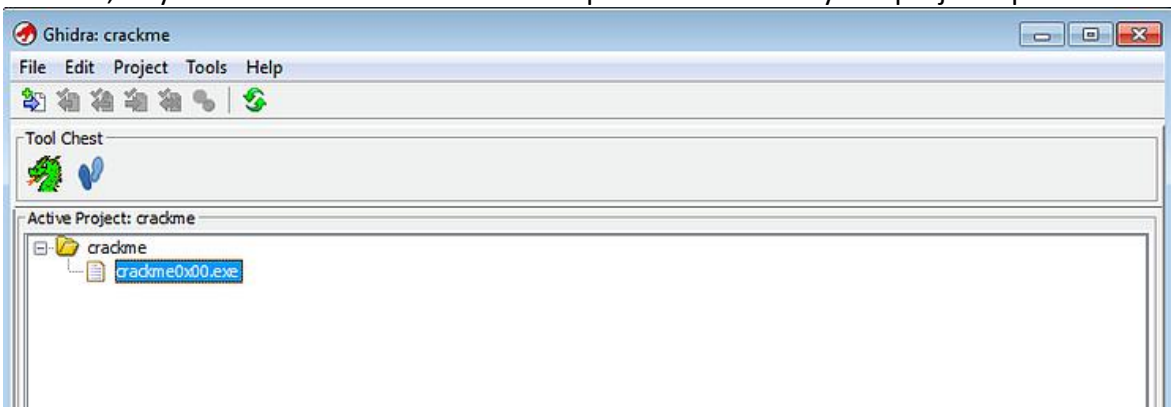
Only the basic features of the application will be tested in the laboratory.

Creating a project

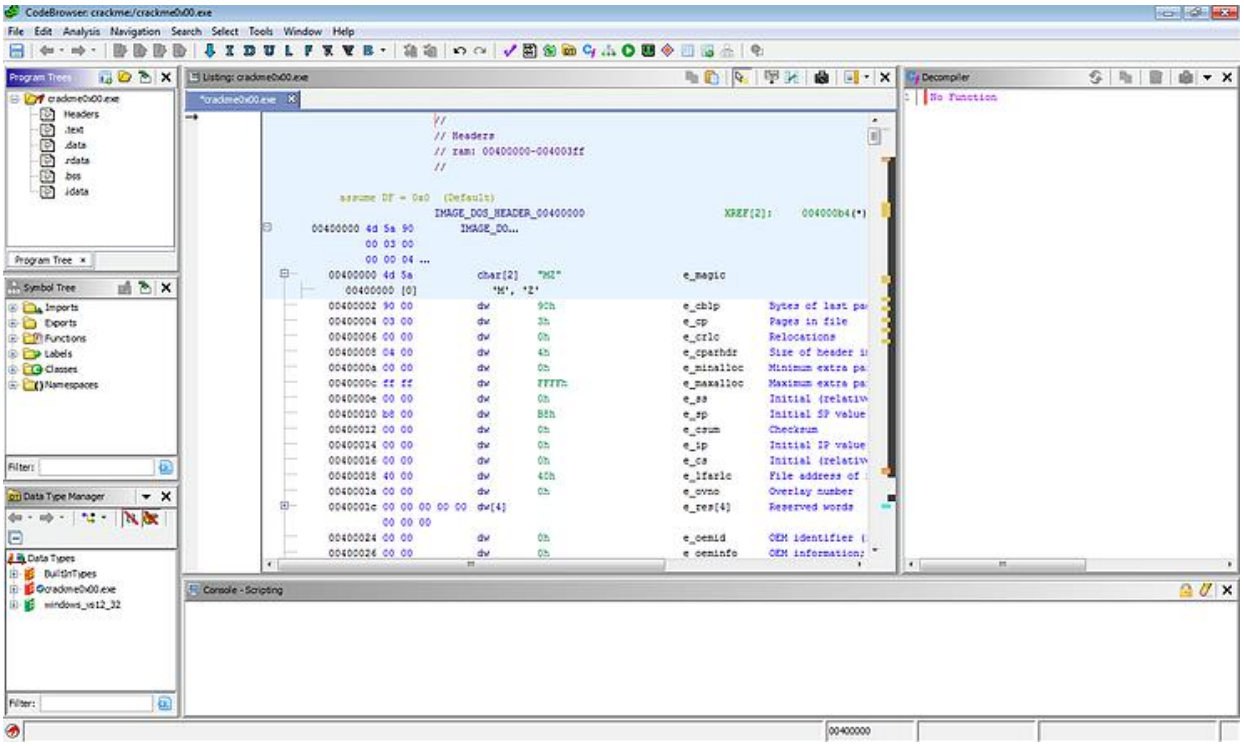
As I said at the beginning, it is recommended to analyze simpler executables. However, to highlight the primary skills of the application was used a slightly more complicated executable called crackme0x00.exe which is available at

(https://github.com/Maijin/Workshop2015/tree/master/IOLI-crackme/bin-win32).

A new project will be created and you can drag the executable directly from the double commander in its window, or you can use the classic: File -> Import File offered by the project options.



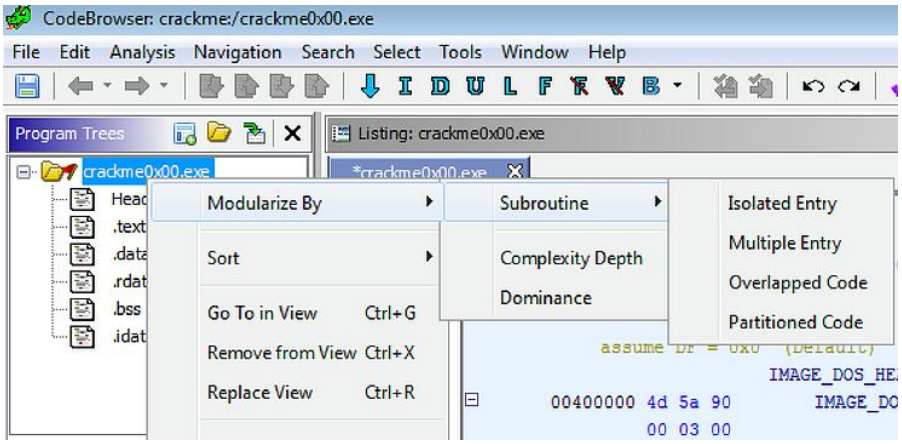
Then with a double-click on crackme0x00.exe will open the code navigator and the question will appear: "Do you want to analyze the binary file?" Once accepted, several options will be presented regarding the analysis of the excutable. At this stage we will choose "Analyze". The code window will be displayed after the operation is complete.



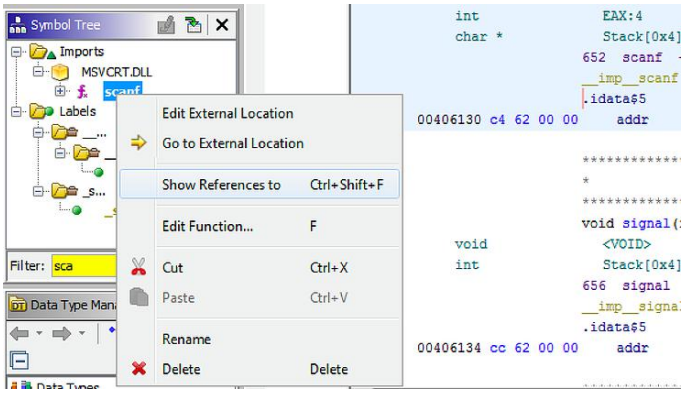
Main window

The first thing observed (and quite rarely) is that it provides help adapted to the context by pressing F1 when the mouse is above an entity on the menu.

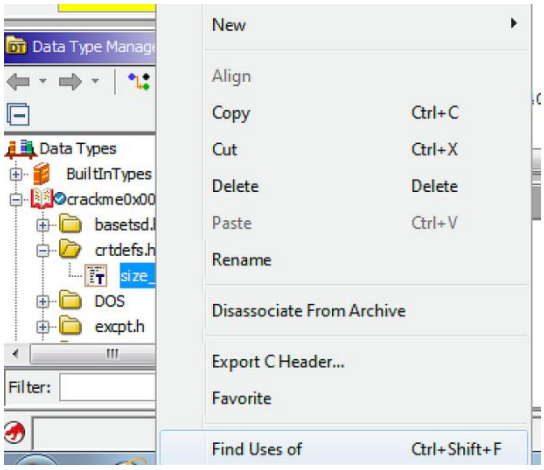
With the "Program Trees" window and a right click over the crackme0x00.exe directory you can select various options for reverse analysis of the code. For example you can go on the following suite of commands: "Modularize By" -> "Subroutine" or "Complexity depth" or "Dominance". The program allows the creation of new directories and direct mouse dragging according to the user's needs.



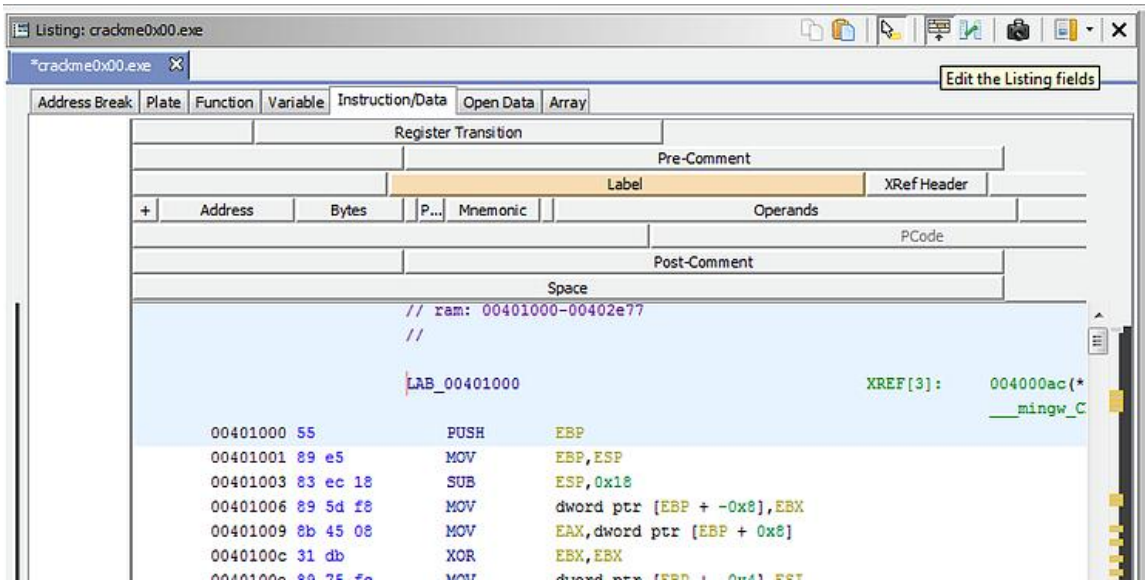
The next window under "Program Trees" is "Symbol Tree" that allows us to view, import, export, functions, labels, classes, and namespaces for an executable file. Try turning on the "Import" section to see various DLLs and their functions. If you want to view where various functions are placed in the executable you can right click and then double-click the result to see the entire section.



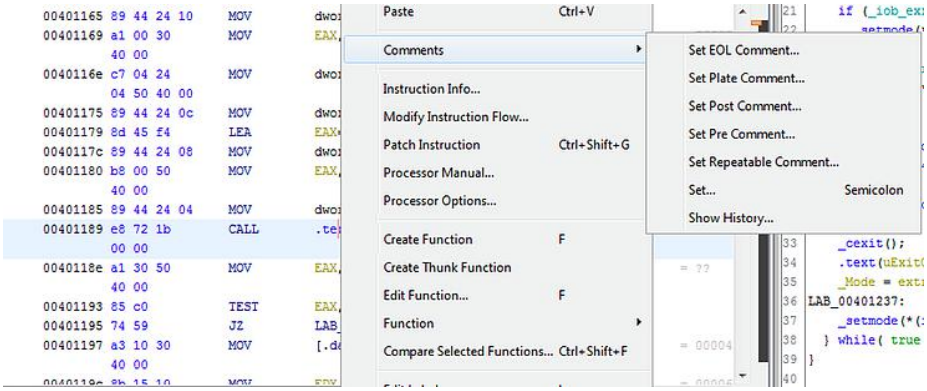
The "Data Type Manager" option allows us to view specific types including those created by the user that are common to a user including the one included in the Guide (e.g. those in the "windows_vs12_32" window). Try unpacking the card icons by right-clicking on the Data Type option. Then click "Find Doors of" to see where that type of data is used in the executable.



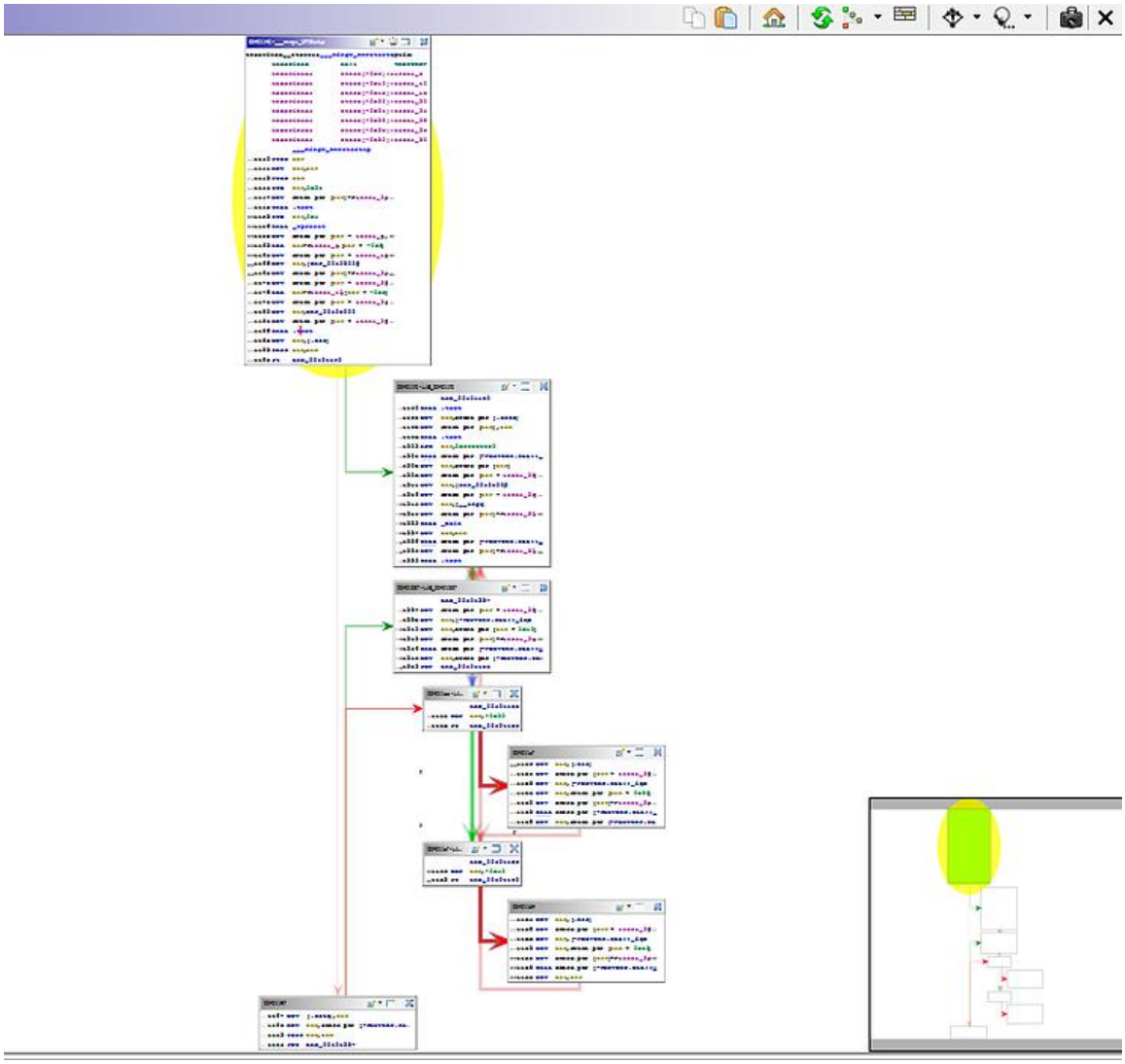
Using the "Listing" window, you can inspect the result of the code's unassembly. Ghidra gives us mute options in setting up the listbox. To do this, use the "Edit the listing fields" icon (located at the top right) and then select the "Instruction/Data" tab. Any element of the interface can be changed, moved, disabled, or removed. You can also add new items by right-clicking and using the context menu. Try changing the "Address" field to reduce the size and remove the bytes field .



You'll see a new context menu in the ASM source for right-click reverse analysis anywhere in the window that contains the code in the assembly language. Here you can correct instructions, set bookmarks, and edit shortcuts. Try adding a comment by right-clicking one of the instructions. If a double click is made on one of the functions indicated by "CALL" we will be sent to its code. Navigate using the top left arrows using the "save" icon or the "Alt-Left Arrow Key" and "Alt-Right Arrow Key" combinations.



To create CFCs (see course) Ghidra offers "Function Graph", which can be accessed through the command suite "Window" -> "Function Graph". It can also be reached via the Edit the listing fields button. This graph does not include comments but allows them to be added as needed with the help of Field Editor.

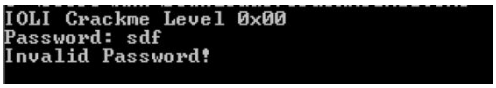


Finally, we see the "Decompile" window that displays a high-level code generated by Ghidra on the right and is associated with the ASM code from disassembly. When an instruction from this automatic code is marked, it will be marked and set asm instructions corresponding to them.

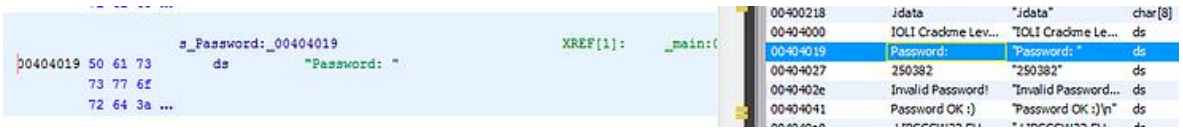
Variable names can be changed or comments can be added (using a right-click option). They will also be displayed in the "Listing" window. If you want to set parameters that need to be displayed and right-click in the window and select "Properties". Verify that you can name a local variable for example by using a name closer to its role. And you notice that the change appears immediately in the "Listing" window.

Running the Program

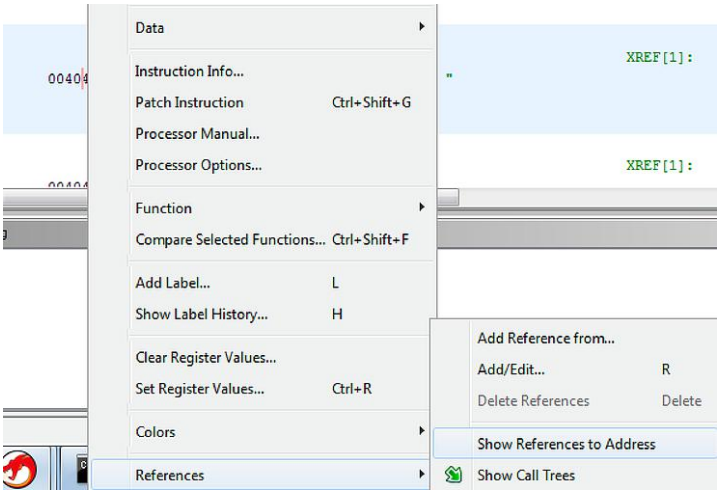
It is noted that the program requires a password that obviously when we insert one at random will report it as invalid.

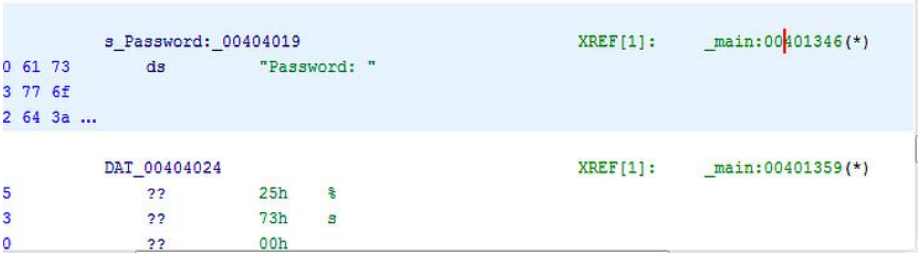


Let's move on to primary processing of this executable by inspecting the program strings using the "Window -> Defined Strings" sequence. You can see a portion of text displayed in the command line. We're basically resuming the analysis presented in the course. Let's look at the ASM code section that handles the password. Double-click the "Password" field in the "Defined Strings" area and you will be automatically taken to the section where the code is kept executable.



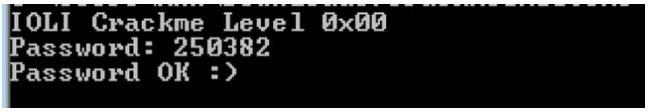
Left-click the address and select "References -> Show References to Address". Then you can click in that area to take us to the basket area that contains the rerereretore to "Password". Another possibility is to double-click the address at the top right of the line that has a note of type XREF[1] notes. Try to find out which assembly section is responsible for verifying the password entered by the user. Possibly rename some variables to clarify the code a little.





Note that in the reference to "Password" there is a scanf call to retrieve the input from the user and then a strcmp call. It is noted that the user string given is saved in EAX and stored in the local variable "local_40". Line "250382" is also saved in the local variable "local_3c" and then both are sent to strcmp. The result of the comparison is then compared to zero and if so then "Password OK :)" will be displayed. otherwise we will display "Invalid Password!". Text.

Let's run the app again but this time with the alleged password ("250382"). Note that this time the password was correct.



Laboratory job

1. **Testing of the application according to the steps described in the laboratory work.**
2. **Resume steps on the pay app presented as an example in the course.**

Homework

To implement in C and then in CPP a program similar to those analyzed then to compile in linux with GCC and in native C# or on mono without included troubleshooting information and then resume the analysis process. A base code report and screenshots will be made during the analysis (in essential areas) for each case Students who will excel can receive for real malware analysis to make a report on the sample received - obviously after signing a legal agreement.