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.

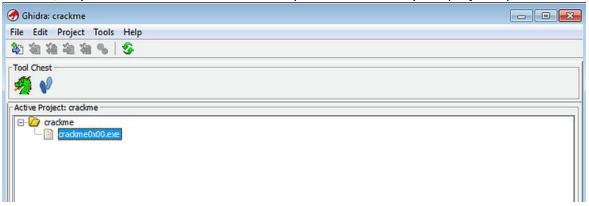
Only the basic features of the application will be tested in the laboratory.

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.



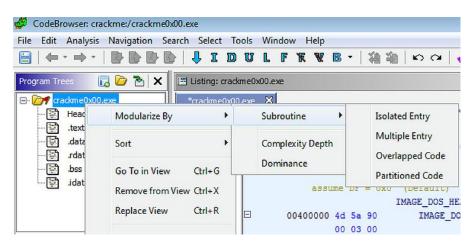
Then with a double-click on crackme0x00.exe will open the code navigator and the question will appear: "Do you want to analyze the binary file?" Once accepted, several options will be presented regarding the analysis of the excutable. At this stage we will choose "Analyze". The code window will be displayed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

rogram Trees 🛛 👔 🙋 🏊 🗙	E Listing: cradenets	900.00e			h 🚺 🐼	◎ 🖌 💩 📑・ ×	C Decompiler	S h B B + X
and method and	"oraclane0x00.exe	8				7	1 No Function	
I Headers I data I data I data I data I data		4/ /// samue DF + 0s0	Beaders ram: 0040000		XFEF	2); 00400064(•)		
	9	00400000 4d 5s 90	1HASE_DO					
Program Tree ×		00 03 00						
Symbol Tree		D- 00400000 4d 5a	char(2)	*82*	e_magic			
6 🚉 Imports 6 🍙 Exports 8 🎒 Functions		00400002 90 00 00400004 03 00 00400005 00 00	dy dy	905 35 05	e_chlp e_cp e_crlo	Sytes of last par Pages in file Relocations		
D 😂 Labels D 🗃 Classes		00400008 04 00	d¥ dx	435 035	e_cparhdr e_minalloc	Size of header in Minimum extra pa		
()Namespaces		0040000c ff ff 0040000c 00 00 0040000c 80 00	dv dv	Oh Bih	e_maxalloc e_ss e_sp	Maximum extra pa Initial (relative Initial SP value		
		00400012 00 00	dx dx	0h	e_coun e_ip	Checksum Initial IP value		
lter:		00400016 00 00	dw dw	0h 40h	e_cs e_ifarle	Initial (relativ)		
🖬 Data Type Manager 🛛 👻 🗙		 0040001a 00 00 0040001c 00 00 00 00 00 00 00 	dv 0 00 dv[4]	05.	e_ovno e_res(4)	Overlay number Reserved words		
Data Types		00400024 00 00	dw dw	015 015	e_cenid e ceminfo	OEM identifier (
BuitinTypes	Conside - Scripting					,	•	<u>a</u> 7 >

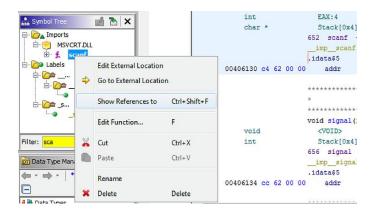
Main window

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.

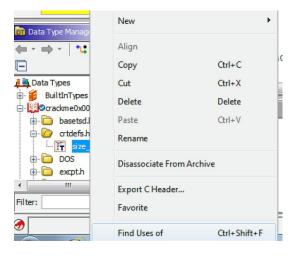
With the "Program Trees" window and a right click over the crackme0x00.exe directory you can select various options for reverse analysis of the code. For example you can go on the following suite of commands: "Modularize By" -> "Subroutine" or "Complexity depth" or "Dominance". The program allows the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



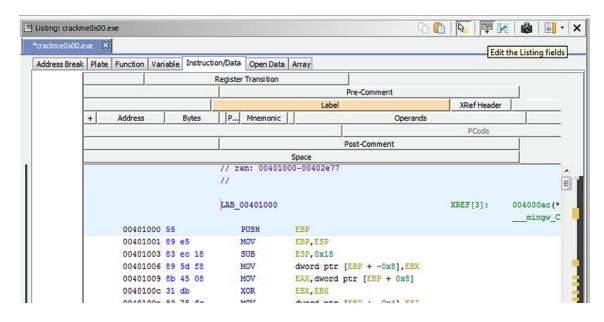
The next window under "Program Trees" is "Symbol Tree" that allows us to view, import, export, functions, labels, classes, and namespaces for an executable file. Try turning on the "Import" section to see various DLLs and their functions. If you want to view where various functions are placed in the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



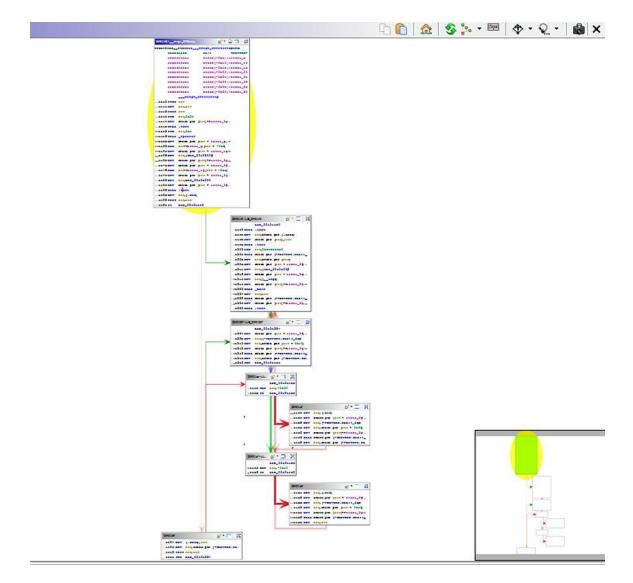
Using the "Listing" window, you can inspect the result of the code's unassembly. Ghidra gives us mute options in setting up the listbox. To do this, use the "Edit the listing fields" icon (located at the top right) and then select the "Instruction/Data" tab. Any element of the interface can be changed, 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.

	00401165	89	44	24	10	MOV	dwor	Paste	Ctrl+V	^21	if (_iob_ex:	
	00401169	a1	00	30		MOV	EAX.			22	setmode (
		40	00					Comments	,	Set EOL Comment		
	0040116e	c7	04	24		MOV	dwor			Set Plate Comment	2	
		04	50	40	00			Instruction Info			1	
	00401175	89	44	24	0c	MOV	dwo	Modify Instruction Flow		Set Post Comment		
	00401179	8d	45	f4		LEA	EAX		01.01% C	Set Pre Comment		
	0040117c	89	44	24	80	MOV	dwor	Patch Instruction	Ctrl+Shift+G	Set Repeatable Comment		
	00401180	b8	00	50		MOV	EAX,	Processor Manual				
		40	00					Processor Options		Set	Semicolon	
	00401185	89	44	24	04	MOV	dwoi	Processor Options		Show History	1	
	00401189	e8	72	1b		CALL	.te	Create Function	F	Show History		
		00	00					Create Function		33	_cexit();	
	0040118e	a1	30	50		MOV	EAX,	Create Thunk Function		= ??	.text(uExit(
		40	00					Edit Function	F	35	_Mode = ext:	
	00401193	85	c0			TEST	EAX,	currancaona			LAB_00401237:	
	00401195	74	59			JZ	LAB	Function	,	37	_setmode(*(:	
	00401197	83	10	30		MOV	[.d/	Compare Selected Functions	Ctrl+Shift+F	= 00004 38	} while(true	
		40	00						Concernance of the	39	}	
_	0040110-	9h	15	10		MOST	FDV			- nnnna 🔨 🚺 40		

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.



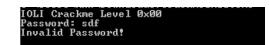
Finally, we see the "Decompile" window that displays a high-level code generated by Ghidra on the right and is associated with the ASM code from disassembly. When an instruction from this automatic code is marked, it will be marked and set asm instructions corresponding to them.

crackme0x00.exe	e - (7 addresses selected)		P 🗍 🗘	1 🛱 N 👘	• ×	C	Decompile: UndefinedFunction_00491099 - 🗤 🧐 🏠 📔 🧟
× 9(9)(0)(0)						11	undefined4 uVar4;
	00401077 8b 5d f8 0040107a 89 ec 0040107c 5d 0040107d c2 04 00	MOV MOV POP RET	EBX,dword ptr [EBP + -0x8] ESP,EBP EBP 0x4		Î	12 13 14 15 16	<pre>bVar2 = false; uVar1 = **ppuParm1; if (uVar1 < 0xc0000092) { if (uVar1 < 0xc0000092) { if (uVar1 < 0xc0000084) {</pre>
÷	00401020 34 05 00	LAB_00401080 CHP	EAX, 0xc0000005	XREF[1]:	004	17 18 19	<pre>if (uVar1 == 0xc0000005) { uVar4 = 0; signal(0xb);</pre>
	00401085 74 5d 00401087 3d 1d 00 00 c0	JZ CMP	LAB_004010e4 EAX, 0xc000001d			20 21 22	<pre>if (extraout_EAX_01 != (code *)0x1) { if (extraout_EAX_01 != (code *)0x0) { (*extraout_EAX_01) (0xb,uVar4); (*extraout_EAX_01) (0xb,uVar4); } }</pre>

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

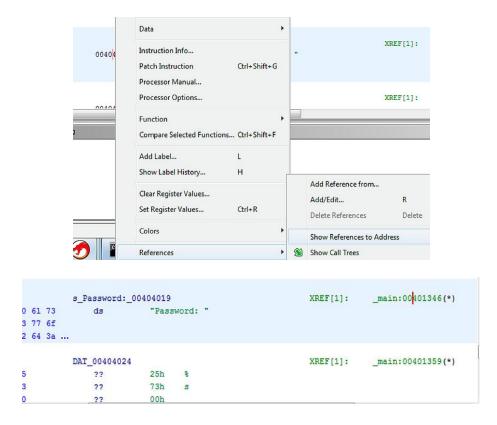
It is noted that the program requires a password that obviously when we insert one at random will report it as invalid.



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.

	100		-	53)					00400218	idata	"idata"	char[8]
				Congestioned				and the second	00404000	IOLI Cradme Lev	TOLI Cradme Le	ds
1				s_Passwo	rd:_00		XREF[1]:	_main:(00404019	Password:	"Password: "	ds
00404019				ds		"Password: "			00404027	250382	"250382"	ds
	73	77	61						0040402e	Invalid Password!	"Invalid Password	ds
	72	64	3a .						00404041	Password OK :)	"Password OK :) \n"	ds
									00.00.00.0	LINCCOULDD DU	Turneenung nu	44

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 rereretore 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.

		Checks if the	input is equal to password "250382"		char local_1c [24]:
00401368 c7 4 04 2 40 0	27 40	MOV	dword ptr [ESP + local_3c],s_250382_00404027 - TENDER	9 10	.text(local_40); main();
00401370 89 0	04 24	MOV	dword ptr [ESP]=>local_40,EAX	11	printf("IOLI Crackme Level 0x00\n");
00401373 e8 9 00 0		CALL	stroup interspicture	13	<pre>printf("Password: "); .text("%s",local_lc);</pre>
00401378 85 0	00	TEST	EAX, EAX	14	/* Checks if the input is
0040137a 74 0	0e	JZ	1AB_0040138a	15	<pre>iVar1 = stromp(local_lc,"250382");</pre>
0040137c c7 0 2e 4	04 24 40 40 0	MOV	dword ptr [ESP]=>local_40,s_Invalid_Password! = "Invalid Passw	16	<pre>if (iVar1 == 0) { printf("Password OK :)\n");</pre>
00401383 e8 a		CALL	printf int printf (char	18 19	else (
00401388 eb 0	0c	JMP	LAB_00401396	20 21	<pre>printf("Invalid Password!\n"); }</pre>
		LAB_0040138a	XREF[1]: 0040137a(j)	22	return 0;
0040138a c7 0	04 24	MOV	dword ptr (ESP)=>local_40,s_Password_OK_:)_004 = "Fassword OK :	23	1

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.