Analyzing Emotet with Ghidra — Part 1

medium.com/@0xd0cf11e/analyzing-emotet-with-ghidra-part-1-4da71a5c8d69

Cafe Babe

April 22, 2019



Apr 19, 2019

6 min read

This post I'll show how I used Ghidra in analyzing a recent sample of Emotet.

If you have read this, here is Part 2.

SHA256:

The analysis is done on the unpacked binary file. In this post I'm skipping how I unpacked the file, since what I primarily want to show is how I used Ghidra's python scripting manager to decrypt strings and API calls.

Some short descriptions:

What is Ghidra?

It is an open source reverse engineering tool suite. You can find out more here —

Why Emotet?

Emotet is a prevalent malware. Started out as a banking trojan. It is persistent and keeps evolving its infection mechanisms. There are other existing analyses done. A search can lead you there —

Why Ghidra and Emotet?

- For starters, I am looking for a new gig (a.k.a unemployed) and hence cannot afford an . Plus I want to continue being a Malware Analyst.
- Using the free version is still amazing, but I miss not being able to use IDA Python. I did use IDA's own scripting language IDC but...I like python. Implemented just one of the functions of Emotet.

Opening up Emotet with Ghidra

Ghidra is about creating projects. Following the on-screen instructions, I created a project named "Emotet". To add files to analyze into the project, simple type or go to .



1. Imported Emotet binary

Ghidra displays properties regarding the file that gets imported. Double click on the file name and it opens it up in CodeBrowser which is a tool that disassembles the file.

<i>\$</i>			CodeBrowse	er: emotet:/0031000	00.bin				- + ×
Eile Edit Analysis Navigation	<u>Search</u> Select <u>T</u> ools	<u>W</u> indow <u>H</u> elp [ခြေ၊ ဩပြာ ရ	/ 🖺 🖄 🛅 🗣 🚠 💽 🐯 🍕						
Program Trees 🛛 🔂 🏲 🗙	🛎 Listing: 00310000.b	in		1	💼 🖳 🐺 📈 🏟	🗐 • 🗙	C Decompile	er	5 h 1 4 - x
 ♥ 00310000.bin ♥ Headers ● I.ext • I.ext • I.data • I.eloc 	00310000 bin ¥	// Hea // Fam // am // a ****** * AddC	ders : 00400000-004003ff ommentToProgramScript - Th	<pre>www.www.www.www.www.www.www.www.www.ww</pre>	1 e 8446846		1 No Funct	ion	
Program Tree ×	C 04	assume DF = 0x0 (Defa IMAGE_ 0400000 4d 5a 90 IM 00 03 00 00 00 04	ult) DOS_HEADER_00400000 MGE_DO	XREF[]	::::::::::::::::::::::::::::::::::::::	-			
Symbol Tree Imports Imports Imports </td <td></td> <td>04400000 45 s 04400000 101 04400000 101 04400000 00 00 04400000 00 00 04400000 00 00 04400000 00 00 04400000 00 00 044000001 08 00 04400012 00 00 04400012 00 00 04400012 00 00 04400012 00 00 04400012 00 00 00 00</td> <td>char[2] "#2" "M" ' Z' dw 90h dw 3h dw 9h dw 4h dw 9h dw 9h dw 9h dw 9h dw 9h dw 88h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9</td> <td>e_magic e_cblp e_cp e_crle e_cparhdr e_maxalloc e_maxalloc e_ss e_sp e_csu e_ip e_cs e_lfarlc e_ers(4) e_oemid e_oemidf e_ers(2) e_res(2)</td> <td>Bytes of last page Pages in file Relocations Size of header in Maximum extra para Maximum extra para Initial (relative) Initial SP value Initial (relative) File address of re Overlay number Reserved words OEM identifier (fo Reserved words</td> <td>XREF</td> <td>1</td> <td></td> <td></td>		04400000 45 s 04400000 101 04400000 101 04400000 00 00 04400000 00 00 04400000 00 00 04400000 00 00 04400000 00 00 044000001 08 00 04400012 00 00 04400012 00 00 04400012 00 00 04400012 00 00 04400012 00 00 00 00	char[2] "#2" "M" ' Z' dw 90h dw 3h dw 9h dw 4h dw 9h dw 9h dw 9h dw 9h dw 9h dw 88h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9h dw 9	e_magic e_cblp e_cp e_crle e_cparhdr e_maxalloc e_maxalloc e_ss e_sp e_csu e_ip e_cs e_lfarlc e_ers(4) e_oemid e_oemidf e_ers(2) e_res(2)	Bytes of last page Pages in file Relocations Size of header in Maximum extra para Maximum extra para Initial (relative) Initial SP value Initial (relative) File address of re Overlay number Reserved words OEM identifier (fo Reserved words	XREF	1		
windows_vs12_32	- Console - Scripting								🗎 🧷 🗴
Filter:									
ð							00400000		

2. Emotet view in CodeBrowser

Under the Symbol Tree (usually on the left or you can go to), I filtered for "entry" to get to the binary's entry point.

📓 Listing: 00	310000.bin				😼 🛱 🛃	💧 🗐 •	×	C _f	Decompile: entry - (00310000.bin)	😵 🐘 🛃 f	8 - 1	×
00310000.bi	n X							2	<pre>/* WARNING: Globals starting with '_' overlap smal</pre>	ler symbols at th	ne same	1
				TORCITON	*********		· ·	3	void entry(void)			Π
			undefined st	dcall entry(yoid)			ור	5	volu chery(volu)			
	und	ofined		<return></return>				6	{			
	und	efined1	Stack[-0x21	A local 210	,	XREE[1].		7	wint uVarl:			
			entry		XREF[2]:	Entry P		8	short sVar2;			
	0040c9a0	55	PUSH	EBP				9	ushort *puVar3;			
	0040c9a1	8b ec	MOV	EBP, ESP				10	undefined4 uVar4;			
	0040c9a3	83 e4 f8	AND	ESP,0xffffff8				11	uint uVar5;			=
	0040c9a6	81 ec 74	SUB	ESP, 0x674				12	int iVar6;			
		06 00 00						13	int iVar7;			
	0040c9ac	53	PUSH	EBX				14	uint uVar8;			
	0040c9ad	56	PUSH	ESI				15	short *psVar9;			
	0040c9ae	57	PUSH	EDI				16	undefined4 extraout_ECX;			
	0040c9a1	e8 fc eb	CALL	resolveNtdll		unde		1/	undefined4 extraout_ECX_00;			
		11 11						18	ushort *puvario;			
	0040c9b4	e8 e7 f2	CALL	resolveKernel32		unde		19	ushoft *pustack1688;			
	0040-0h0	TT TT	DUCU	0				20	undefined #puStack1690.			
	00406909	00 00	PUSH	0X104				22	undefined4 uStack1676:			
	0040c0bc	od 04 24	I EA	EAX-slocal 210 [0x47c + ESP]				23	short asStack1608 [4];			
	00400906	7c 04 00 0	0	EAX=>COCat_210,[0X4/C + ESF]				24	undefined auStack1600 [520]:			
	0040c9c5	50	PUSH	FAX				25	undefined auStack1080 [4]:			
	0040c9c6	6a 00	PUSH	0×0				26	undefined auStack1076 [16]:			
	0040c9c8	ff 15 68	CALL	dword ptr [GetModuleFileNameW]				27	undefined auStack1060 [500];			
		08 41 00						28	undefined auStack560 [20];			
	0040c9ce	66 8b 84	MOV	AX,word ptr [0x478 + ESP]				29	ushort auStack540 [6];			
		24 78 04						30	undefined local_210 [524];			
		00 00						31				
	0040c9d6	8d 8c 24	LEA	ECX,[0x478 + ESP]				32	uStack1676 = 0x40c9b4;			
		78 04 00 0	0					33	resolveNtdll();			
	0040c9dc	33 ff	XOR	EDI,EDI				34	uStack1676 = 0x40c9b9;			
	0040c9d1	66 85 c0	TEST	AX,AX				35	resolveKernel32();			
	0040c9e2	74 22	JZ	LAB_0040ca06				30	USTack1676 = UX104;			
	0040c9e4	0f b7 c0	MOVZX	EAX, AX				37	pustack1680 = tocat_210;			
	0040c9e	eb 07	JMP	LAB_0040c9T0				20	$u_{stack1004} = 0;$ $u_{stack1609} = (uchart *)0x40c0co,$			
	00400989	00 00 00 0	0 LEA	ESP, [ESP]				40	(* GetModuleFileNameW)():			
		00 00 00 0	0					41	nuVar3 = auStack540:			
			LAB 0040c9f0		XREE[2].	0040 (96	-	42	puVar10 = (ushort *)0x0:			
▶	0040c9f0	69 ff 3f	TMUL	EDI.EDI.0x1003f	AURI (2).			43	while (auStack540[0] != 0) {			-
• •	1					•		4			•	

3. Entry Point of Emotet

Under Listing we see the compiled code and on the right is its decompiled code. Since I've already analyzed these binaries, some of the sub routine calls and offsets in these images will have been renamed by me. To rename an offset, right-click an offset value and select (or type).

Emotet's Function Calls

Emotet encrypts its strings and stores its API call names as hashes. So statically viewing this file, is a pain to read.

Without going into much detail about Emotet's payload (that would require another blog entry), I will show how to make this binary a bit more easy to follow. It does require to initially go through each function and figure out the math (possibly using , or whichever debugger so to make it a little less painful).

In this case I wanted to figure 2 methods used by Emotet. The first function is a simple xor routine that it uses to decrypt strings. It looked deceiving complex (because of the use of shift operators in the function), only till after running one iteration in that I realized what was happening.... The second function finds which API name matches which hash (I will cover this in <u>Part 2</u>). This I felt was a bit more clever, but still easy to understand after running in .

Then using Ghidra's Script Manager, I'll show how I implemented the python scripts to decrypt the strings and resolve the API calls used in the binary.

How are the Strings encrypted?

In the binary, I've noticed a lot of references to the function call at . This call decrypts for the strings. I renamed it to . To find references made to the function, right click the function and select .

References to decode_strinser: emotet:/003	10000.bin]	- + ×
--	------------	-------

Teib										
References to decode_string	gs - 26 locations	📎 🏫 🌮 🔳 💽 🔀 🗙								
Reference(s)										
Location 🖹 Label	Code Unit	Context								
00401518	CALL decode_strings	UNCONDITION 🔺								
00401cfc	CALL decode_strings	UNCONDITION								
0040672a	CALL decode_strings	UNCONDITION								
00406754	CALL decode_strings	UNCONDITION								
00406932	CALL decode_strings	UNCONDITION								
004070d4	CALL decode_strings	UNCONDITION								
00407968	CALL decode_strings	UNCONDITION								
00407afa	CALL decode_strings	UNCONDITION								
00408a18	CALL decode_strings	UNCONDITION								
004097e4	CALL decode_strings	UNCONDITION								
00409c22	CALL decode_strings	UNCONDITION								
0040ad90	CALL decode_strings	UNCONDITION								
0040adf4	CALL decode_strings	UNCONDITION								
0040ae67	CALL decode_strings	UNCONDITION								
0040aecd	CALL decode_strings	UNCONDITION								
0040cal0	CALL decode_strings	UNCONDITION								
0040cc53	CALL decode_strings	UNCONDITION								
0040cca4	CALL decode_strings	UNCONDITION								
0040ccf4	CALL decode_strings	UNCONDITION								
0040cd22	CALL decode_strings	UNCONDITION 💌								
Filter:										

4. References to decode_strings

٢

Hole

	LAB 0040ca06		XREF[1]:	0040c9e2(j)
0040ca06 ba 3e 5a	MOV	EDX,0x77265a3e		-
0040ca0b b9 e0 fc	MOV	ECX,DAT_0040fce0		%X
40 00	CALL	decede strings		acy _ offect
ff ff	CALL	decode_strings		edx = key
0040ca15 <mark>8b f0</mark>	MOV	ESI.EAX		

5. Call being made to decode_strings

The function takes in 2 arguments that are stored in and (Image 5). is the offset of the encrypted string. is the xor key. The decrypted string gets stored in memory allocated in the heap and the address gets passed to .

(Side Track: I have added the string " $ecx = offset \ n \ edx = key$ " as a repeatable comment to the function. Right click the address and select or type)

The first dword at the offset xor'ed with the key returned the length of the string. The next subsequent set of dwords were xor'ed up until the string's length.

Now for the more exiting part, automating this with a python script in Ghidra.

0

6. Script Manager Icon

In the top toolbar section of Ghidra, we see this icon in image 6. It takes us to the Script Manager. Else you can select .

Script Manager [CodeBrowser: emotet:/00310000.bin] - + ×										
Heip Script Manager - 241	scripts				0 9			+ ×		
🜔 Scripts 📃	In T	Sta	Name	Description	Key	Category	Modified			
- DEW_			AddCommentToProgramScript.java		-	Examples	04/03/2019			
• 🔲 Analysis			AddCommentToProgramScriptPy.py	Adds a com		Examples->	04/03/2019	=		
ARM			AddReferencesInSwitchTable.java	With cursor		ARM	04/03/2019			
Binary			AddSingleReferenceInSwitchTable.java	With a user-i		ARM	04/03/2019			
- Cleanup			AppleSingleDoubleScript.java	Given a raw		Binary	04/03/2019			
- 🔁 CodeAnalysis			ArmThumbFunctionTableScript.java	Makes functi		ARM	04/03/2019			
- 🛅 Conversion			AsciiToBinaryScript.java	Converts an		Conversion	04/03/2019			
🗢 🚞 CustomerSubmi			AskScript.java	An example		Examples	04/03/2019			
– 🛄 Data 📃			AskScriptPy.py	An example		Examples->	04/03/2019			
Data Types			AssembleBlockScript.java	cript.java Assemble ha			04/03/2019			
Examples	S AssembleCheckDevScript.java		AssembleCheckDevScript.java	Test assembl	Ctrl-H	Assembly	04/03/2019			
- Eunctions			AssembleScript.java	Assemble a s		Assembly	04/03/2019			
- D FunctionStartPa	tPa AssemblyThrasherDevScript.java Thorough	Thoroughly t	roughly t Assembly							
- 🛅 HELP	AutoRenameLabels		AutoRenameLabelsScript.java	Renames def		Symbol	04/03/2019			
- 🛅 Images			AutoRenameSimpleLabels.java	Labels.java A ghidra scri			04/03/2019			
- 🚞 Import			AutoVersionTrackingScript.java	An example		Examples->	04/03/2019			
- Distructions			BadInstructionCleanup.java	This script cl		iOS	04/03/2019			
- OS			BatchRename.java	Recursively f		Project	04/03/2019			
			BatchSegregate64bit.java	Separates co		Project	04/03/2019	_		
			DiT-Alici			<u></u>	04/02/2010	-		
Filter: 🔤	Filter:							- 🔁		
	•)									

7. Script manager

The Script Manager displays a list of scripts written in either Java or Python. They come with the installation. The script manager also has some python script examples. So, I filtered for .py scripts to help me understand how to proceed in writing a python script. The Python Interpreter interacts with Ghidra's Java API through <u>Jython</u>. The documentation on the Java APIs provided can be found in a zipped file in the docs directory of your Ghidra installation.

8. Create new script icon

To create a new python script, select this icon — image 8. Select Python and enter a name you'd like to give to your script.

💋 Help		Script Manage	r, Te	est.py [leB	rowser: emotet:/00310000.bin] - + x	
Help Script Manager - 242 Scripts NEW Analysis ARM Assembly Binary Cleanup	scripts	Q Q 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	· · · · · · · · · · · · · · · · · · ·	··· ·· ···	•••	×	Test.py ↓ TODO write a description for this script #@author #@acategory _NEW_ #@keybinding #@menupath #@toolbar
Cleanup CodeAnalysis CodeAnalysis CodeAnalysis Conversion CustomerSubmi Data Data Types FunctionID FunctionStartPa Functions FunctionStartPa HELP Images Import SoS		SearchMnem SelectFuncti SetEquateSc SetHeadless ShowCcallsS ShowEquate SplitExtensi SplitExtensi SplitMultiple SplitUnivers StringParam SwbsToFuncs	· · · · · · ·			#TODO	#TODO Add User Code Here
Test.py No Description Author: Category: Key Binding: Menu Path:	Filter:	SwitchOverri Test.py					

8. A sample test.py script created

Additionally, going through the help docs (under) and reading under , there is a description of the metadata tags that gets generated when creating a new script.

I've uploaded the script into my github repo and you can follow it here — <u>https://github.com/0xd0cf11e/ghidra/blob/master/ghidra_emotet_decode_strings.py</u>

MOV	ECX,DAT_00410130	%s\%s
CALL	decode_strings	ecx = offset edx = key

9. Decrypted string displayed as comment

The idea behind the script is to display the strings that get decrypted as comments next to the instruction where its offset is moved to (Image 9).

				DAT_00410130		
00410130	24			??	24h	\$
00410131	11			??	11h	
00410132	23			??	23h	#
00410133	ld			??	1Dh	
00410134	25	73	5c	ds	"%s\\	%s"
	25	73				

10. Bytes patched in the binary.

.

And as well to patch the bytes in the binary (Image 10).

```
# get all code references made to the function
refs = getReferencesTo(toAddr(loc))
```

First step, I wanted to find all the code references made to the function.

```
# The parameters passed to the decode function
# are in registers ecx and edx
# iterate through max 100 instructions
# to search for the values moved to the register
i = 0 # counter
ecx = 0 # offset with data
edx = 0 # xor key
comm = 0 # offset to comment on
while((i < 100) and ((ecx == 0) or (edx == 0))):
        inst = getInstructionBefore(inst)
        if "MOV ECX" in inst.toString():
               comm = inst.getAddress()
                ecx = inst.getAddress(1)
                print("ECX = %s" % ecx)
        if "MOV EDX" in inst.toString():
                edx = getInt(inst.getAddress().add(1))
                print("EDX = %s" % edx)
        i += 1
```

Iterating through each reference, the next step was locating for the opcode instructions and . The instructions weren't always immediately before the call to the function. So I iterated through a max of 100 instructions to search for the opcodes.

After that I was all set to carry out the xor routine and patch the bytes and comment at the instruction offset where was carried out.