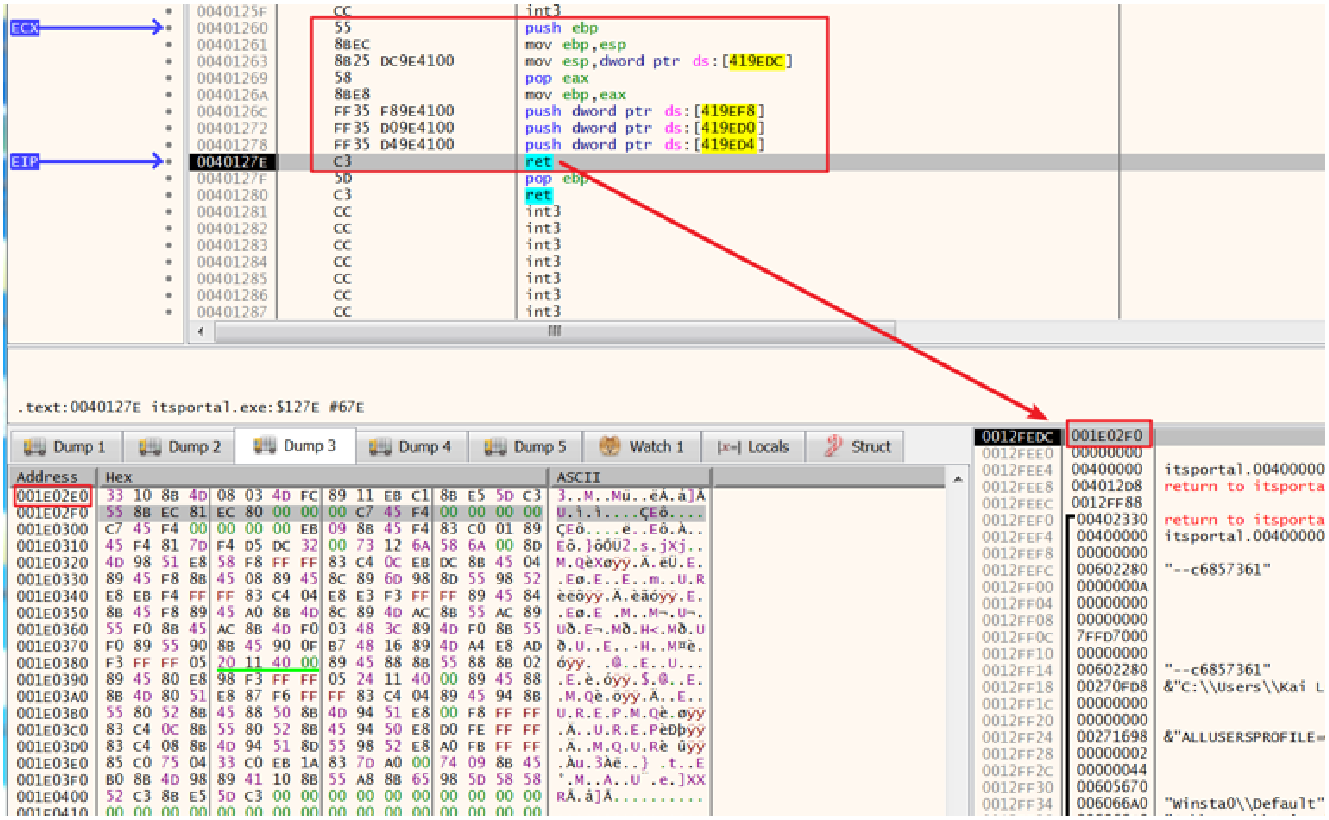
A Deep Dive into the Emotet Malware

fortinet.com/blog/threat-research/deep-dive-into-emotet-malware.html

June 6, 2019



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.

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().

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.

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.

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.

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.

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.

Figure 15. The structure of the gathered data

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

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.

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 the decodes the corresponding data with Deflate algorithm.

Additionally, the RSA key is hard-coded at offset 0x0040FBF0 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://montalegrense[.]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.