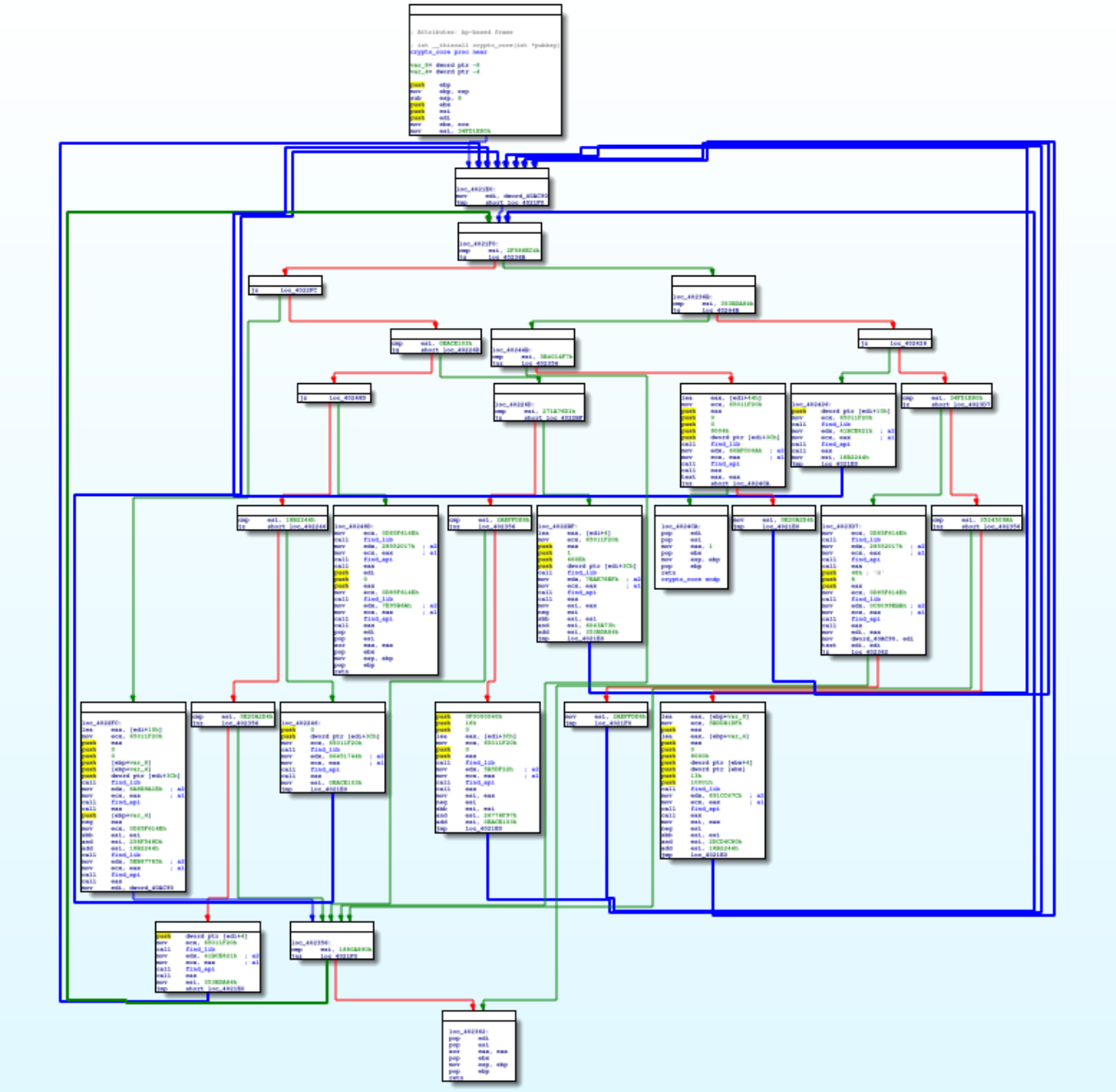
What’s up Emotet? | CERT Polska

cert.pl/en/news/single/whats-up-emotet/



What’s up, Emotet?

Emotet is one of the most widespread and havoc-wreaking malware families currently out there. Due to its modular structure, it’s able to easily evolve over time and gain new features without having to modify the core.

Its first version dates back to 2014. Back then it was primarily a banking trojan. These days Emotet is known mostly for its spamming capabilities and as a delivery mechanism of other malware strains.

It has recently undergone a substantial change in communication protocol and obfuscation techniques. This might be a response to the release of tools allowing researchers to easily download payloads from the C2 servers[1] and detect machines infected with Emotet[2].

In this article, we will go over the standard Emotet features and take a look at some of the changes that have been spotted.

Sample analysed: 500221e174762c63829c2ea9718ca44f
Unpacked Emotet core: e8143ef2821741cff199eeda513225d7

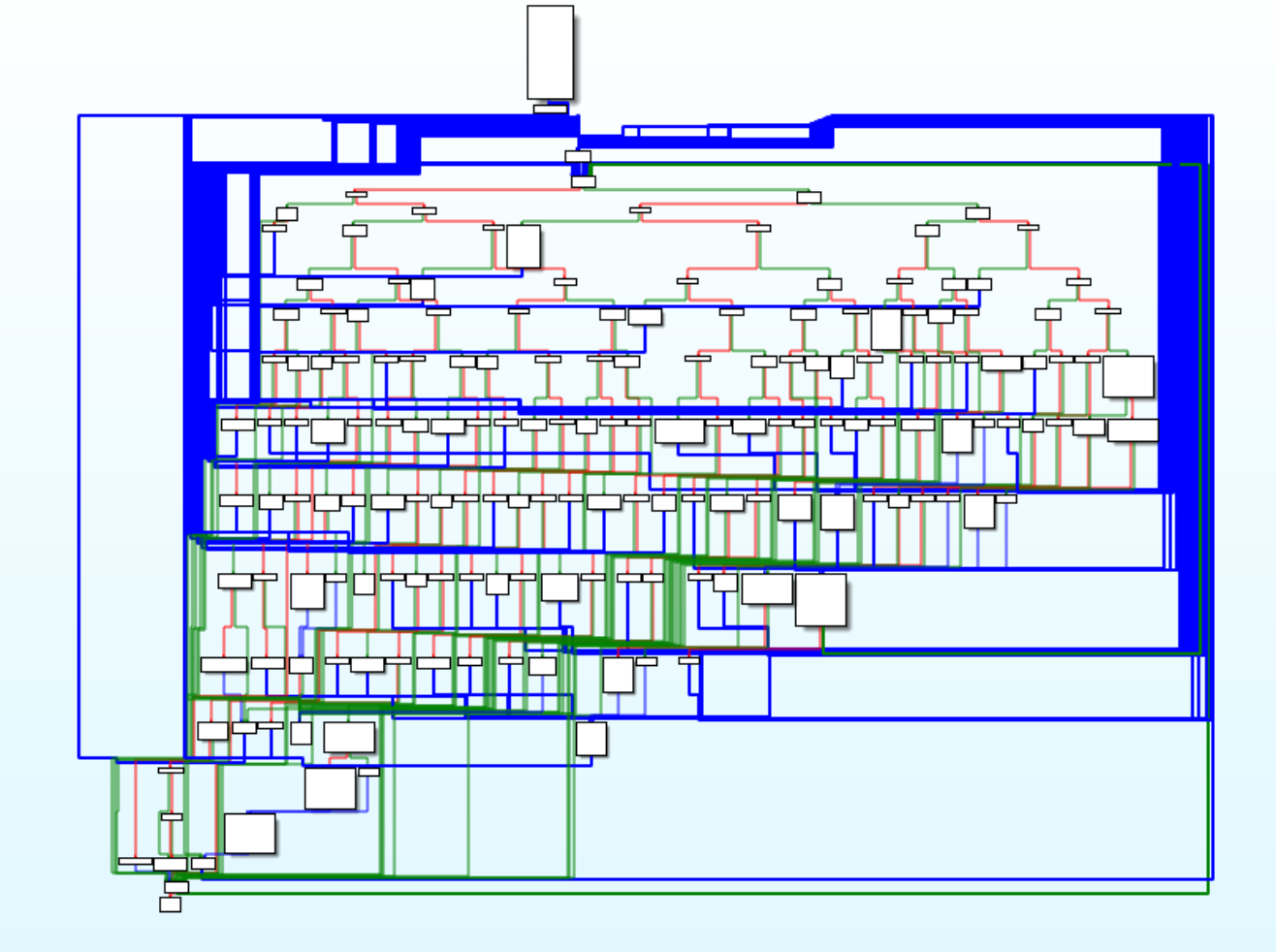
Table of Contents

Anti-analysis features

Code Flow Obfuscation

In order to make reverse engineering more difficult for researchers, a VM-like obfuscation was implemented. To achieve this, every function was split into basic blocks which were then repositioned into a simple state machine.

Demangling the functions back to their original form is nontrivial, although possible. However, it was found that reverse engineering obfuscated binaries is still possible.



Function graph of the main function

Encrypted Strings

All used strings are encrypted almost like in the previous versions. Most noticeable difference is related to the xor key – it's not passed as a parameter anymore. Instead, it's located at the beginning of the data to be decrypted.

```
if ( result == 376234972 )
{
  if ( v32[0] & 0x10 )
  {
    if ( (v33 != 46 || v34 && (v34 != 46 || v35)) && a1 )
    {
      v7 = decrypt_string(dword_40A230);     // %s\%s
      v8 = v7;
      v27 = v36;
```

```
.data:0040A200                          ; int dword_40A200[12]
.data:0040A200 55 BA 48 19 51 BA 48 19+dword_40A200     dd 1948BA55h, 1948BA51h, 3314C970h, 0C81CAAD1h, 270C5FF8h
.data:0040A200 70 C9 14 33 D1 AA 1C C8+                                         ; DATA XREF: sub_403120+165↑o
.data:0040A200 F8 5F 0C 27 A6 55 92 8A+                 dd 8A9255A6h, 0B1809898h, 0D5E7A91Fh, 61147CADh, 3 dup(0) ; %s\*
.data:0040A230                          ; int dword_40A230[8]
.data:0040A230 17 DE FE 51 12 DE FE 51+dword_40A230     dd 51FEDE17h, 51FEDE12h, 74A2AD32h, 133C3764h, 0A5C5A835h
.data:0040A230 32 AD A2 74 64 37 3C 13+                                         ; DATA XREF: sub_403120+93↑o
.data:0040A230 35 A8 C5 A5 74 E6 C2 9E+                 dd 9EC2E674h, 2 dup(0)  ; %s\%s
.data:0040A250                          ; int dword_40A250[12]
.data:0040A250 FD FF 06 4A F2 FF 06 4A+dword_40A250     dd 4A06FFFDh, 4A06FFF2h, 196896AAh, 16369E89h, 2B609AB9h
.data:0040A250 AA 96 68 19 89 9E 36 16+                                         ; DATA XREF: sub_403F80+71↑o
.data:0040A250 B9 9A 60 2B 88 93 72 62+                 dd 62729388h, 480ED052h, 0A7D37BE1h, 7DD55ABFh, 0D309C3E1h ; WinSta0\Default
.data:0040A250 52 D0 0E 48 E1 7B D3 A7+                 dd 59E414ADh, 3008A9AAh
```

Example encrypted string

Encrypted string structure

One can decrypt those strings pretty easily using a quick Python script.

Python function used for decrypting strings

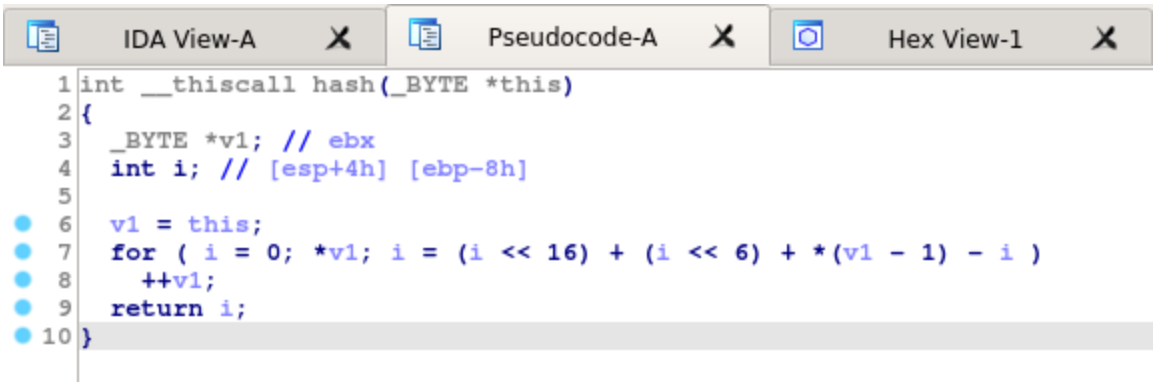
WinAPI

Another method of slowing down the analysis that the malware authors really like is hiding the Window API calls by replacing them with a custom lookup function.

Executing API calls using hash lookups isn't a new thing in Emotet. In contrast to previous versions however, the new version fetches them on a need-to-use basis instead of loading them all at once and storing them in a data section.

```
15
16  v2 = find_lib(0x850FA728);
17  v3 = find_api(v2, 703798143);
18  v3(v14, 0, 520);
19  if ( *this )
20  {
21    v4 = v14 - this;
22    do
23    {
24      v5 = *this;
25      this += 2;
26      *&this[v4 - 2] = v5;
27      if ( v5 == 92 )
28      {
29        v6 = find_lib(0xD85F614E);
30        v7 = find_api(v6, -1488501220);
31        v8 = v7(v14);
32        if ( v8 == -1 )
33        {
34          v9 = find_lib(0xD85F614E);
35          v10 = find_api(v9, -1023523628);
36          if ( !v10(v14, 0) )
37          {
38            v11 = find_lib(0xD85F614E);
39            v12 = find_api(v11, 36543150);
40            if ( v12() != 183 )
41              return 0;
42          }
43        }
44        else if ( !(v8 & 0x10) )
45        {
46          return 0;
47        }
48      }
49    }
50    while ( *this );
```

Api lookup function being used



Simple hash function used for function name hashing

It can be solved rather easily. All one has to do is just reimplement the hashing function, iterate over common WinAPI function names and create an enum with all recovered hashes.

It's very important to set the accepted type in find_api to the newly-created enum type. This will allow IDA to automatically place the enum values in function calls.

Comparison of a single function before and after applying the enum type

Deleting previous versions of itself

While analysing the encrypted strings, one of lists of keywords present in earlier versions was noticed. It was used to generate random system paths in which to put the Emotet core binary. This seemed weird because this method was replaced with completely random file paths.

After closer inspection and confirmation by @JRoosen[3] it turned out that these keywords are used to delete Emotet binaries that were dropped there by previous versions.

```
v0 = get_volume_info();
exe_keywords = decrypt_string(dword_40A860);  // duck,mfidl,targets,ptr,khmer,purge,metrics,acc,inet,msra,symbol,driver,
                                              // sidebar,restore,msg,volume,cards,shext,query,roam,etw,mexico,basic,url,
                                              // createa,blb,pal,cors,send,devices,radio,bid,format,thrd,taskmgr,timeout,
                                              // vmd,ctl,bta,shlp,avi,exce,dbt,pfx,rtp,edge,mult,clr,wmistr,ellipse,vol,
                                              // cyan,ses,guid,wce,wmp,dvb,elem,channel,space,digital,pdeft,violet,thunk
split_by_comma(exe_keywords, v36, v0);
v2 = find_lib(0xD85F614E);
GetProcessHeap = find_api(v2, GetProcessHeap);
v33 = GetProcessHeap(v32, v34, v35[0]);
v4 = find_lib(0xD85F614E);
HeapFree = find_api(v4, HeapFree);
HeapFree(v33, 0, exe_keywords);
v6 = *(dword_40AC98 + 1100) == 0;
v35[0] = v37;
if ( v6 )
{
  v9 = find_lib(2594562649);
  SHGetFolderPathW = find_api(v9, SHGetFolderPathW);
  SHGetFolderPathW(0, 28);
  v11 = decrypt_string(dword_40AAF0);         // %s\%s
  v12 = find_lib(0x850FA728);
  _snwprintf = find_api(v12, _snwprintf);
  _snwprintf(v37, 260, v11, v37, v36);
  v14 = find_lib(0xD85F614E);
  GetProcessHeap_1 = find_api(v14, GetProcessHeap);
  v29 = GetProcessHeap_1(0, 0, v35[0]);
  v16 = find_lib(3630129486);
  HeapFree_1 = find_api(v16, HeapFree);
  HeapFree_1(v29, 0, v11);
}
else
{
  v7 = find_lib(2594562649);
```

Part of the function used for deleting older versions of Emotet

Extracting static configuration

Public key

The RSA public key is stored as a regular encrypted string. It's embedded in the binary in order to encrypt the AES keys used for secure communication with the C2. This will deter all communication eavesdropping attempts.

The public key isn't stored in plaintext, but fetched like rest of the encrypted strings. Thus, it can be decrypted using the same script:

The resulting key is encoded using DER format and can be parsed using the following script:

Result PEM-encoded public key

C2 list

The method of retrieving C2 hosts has not changed. They are still stored as 8-byte blocks containing packed IP address and port.

```
copied_c2 = v6;
if ( v6 )
{
  v7 = v6[6];
  v6[3] = c2_data;
  v6[5] = c2_data;
  for ( v6[1] = 0; c2_data[2 * v7]; v6[6] = v7 )
    ++v7;
  if ( crypto_core(pubkey) )
    return 1;
  v9 = copied_c2;
  v10 = find_lib(0xD85F614E);
  GetProcessHeap_1 = find_api(v10, GetProcessHeap);
  v15 = GetProcessHeap_1();
  v12 = find_lib(0xD85F614E);
  HeapFree = find_api(v12, HeapFree);
  HeapFree(v15, 0, v9);
}
```

```
.data:0040A2A0 51 CB F9 C8 3A A9 C3 1B+              dd 36444E5Eh, 5063EF54h, 662BBD09h, 0B0FFBE0Ah, 7DBE7B88h
.data:0040A2A0 A7 55 15 2D 50 C3 99 55+              dd 5CA0D0A2h, 2CBA0F7Eh, 0EB5B2088h, 92EDF730h
.data:0040A328                                ; int c2_data[258]
.data:0040A328 60 57 49 AD                    c2_data         dd 0AD495760h           ; DATA XREF: crypto_stuff+44↑o
.data:0040A328                                                                        ; crypto_stuff+4B↑o ...
.data:0040A32C 50 00                                          dw 80
.data:0040A32E 9A 37                                          dw 379Ah
.data:0040A330 87 E9 DE 47                                    dd 47DEE987h
.data:0040A334 BB 01                                          dw 443
.data:0040A336 30 F2                                          dw 0F230h
.data:0040A338 16 4E FA 3C                                    dd 3CFA4E16h
.data:0040A33C BB 01                                          dw 443
.data:0040A33E 37 59                                          dw 5937h
.data:0040A340 5B 5B 56 50                                    dd 50565B5Bh
.data:0040A344 90 1F                                          dw 8080
.data:0040A346 48 01                                          dw 148h
.data:0040A348 2F 1C EC 68                                    dd 68EC1C2Fh
.data:0040A34C 90 1F                                          dw 8080
.data:0040A34E E7 7E                                          dw 7EE7h
.data:0040A350 DB 5C F1 A2                                    dd 0A2F15CDBh
.data:0040A354 90 1F                                          dw 8080
.data:0040A356 9B C2                                          dw 0C29Bh
.data:0040A358 68 2D D0 4A                                    dd 4AD02D68h
```

host
port
padding

Communication

Path generation

Keyword-generated paths have been abandoned in favour of fully random ones.

Each new path consists of a random amount of alphanumeric segments separated by slashes.

```
187         if ( v6 > 610269795 )
188         {
189           if ( v6 == 657870806 )
190           {
191             path = v95;
192             segment_no = random_tick % 6 + 1;    // random_seg_length = <1,6)
193             if ( random_tick % 6 != -1 )
194             {
195               do
196               {
197                 segment_length = (random_tick & 0xF) + 4;
198                 random_alphanum_utf16(segment_length, path, &random_tick);// generate random path
199                 v58 = &path[2 * segment_length];
200                 *v58 = '/';                       // append slash
201                 path = v58 + 2;
202                 --segment_no;
203               }
204               while ( segment_no );
205               v5 = v117;
206               v4 = v115;
207             }
208             v3 = v112;
209             *path = 0;
210             v6 = 375988509;
```

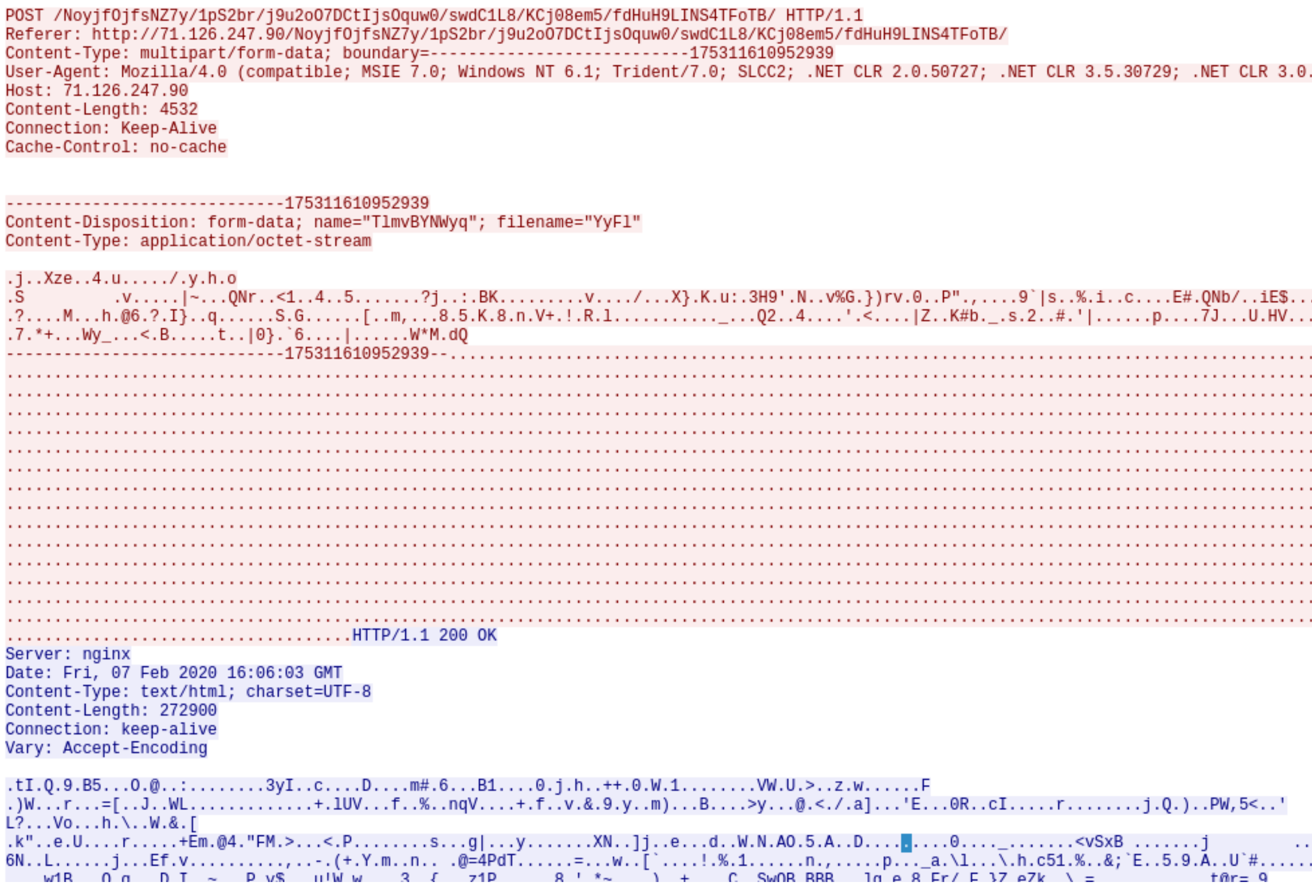
Path generation algorithm

Additionally, instead of simply uploading the payload data inside the POST body, it is now sent as a file upload using multipart/form-data enctype.

The method of generating random attachment names and filenames is quite similar to the one used in generating URL paths.

```
111         v33 = (v31 & 0xF) + 4;
112         v34 = 0;
113         do
114           alphabet[v34++] = v32++;
115         while ( v32 <= 'Z' );
116         for ( k = 'a'; k <= 'z'; ++k )
117           alphabet[v34++] = k;
118         for ( l = 0; l < v33; ++l )
119         {
120           v37 = find_lib(0x850FA728);
121           v38 = find_api(v37, RtlRandomEx);
122           random_filename[l] = alphabet[v38(&v74) % v34];
123         }
124         random_filename[v33] = 0;
125         v39 = get_string(dword_40A060);          // --%S
126                                                  // Content-Disposition: form-data; name="%s"; filename="%s"
127                                                  // Content-Type: application/octet-stream
128         v6 = v71;
129         v40 = v39;
130         v41 = v73;
131         v60 = v70;
132         v58 = v39;
133         v57 = v71 - v73;
134         v56 = v73;
135         v42 = find_lib(0x850FA728);
136         _snprintf = find_api(v42, _snprintf);
137         v3 = _snprintf(v56, v57, v58, v60, random_name, random_filename) + v41;
138         v73 = v3;
139         v44 = find_lib(0xD85F614E);
140         v45 = find_api(v44, GetProcessHeap);
141         v61 = v45();
142         v46 = find_lib(0xD85F614E);
143         v47 = find_api(v46, HeapFree);
144         v47(v61, 0, v40);
```

Part of function responsible for encoding the data as a file



Example request and response dissected in Wireshark

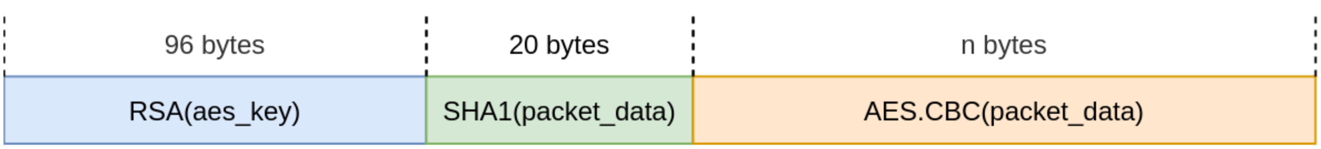
Request structure

This the part that has gone under the most changes. Protocol buffers have been dropped in favour of a custom binary protocol.

Packet encryption

Just like in previous versions, all packets are encrypted using AES-CBC with 16 nullbytes as IV. The AES key is generated using the CryptGenKey function, encrypted using the decoded RSA public key and appended to each request.

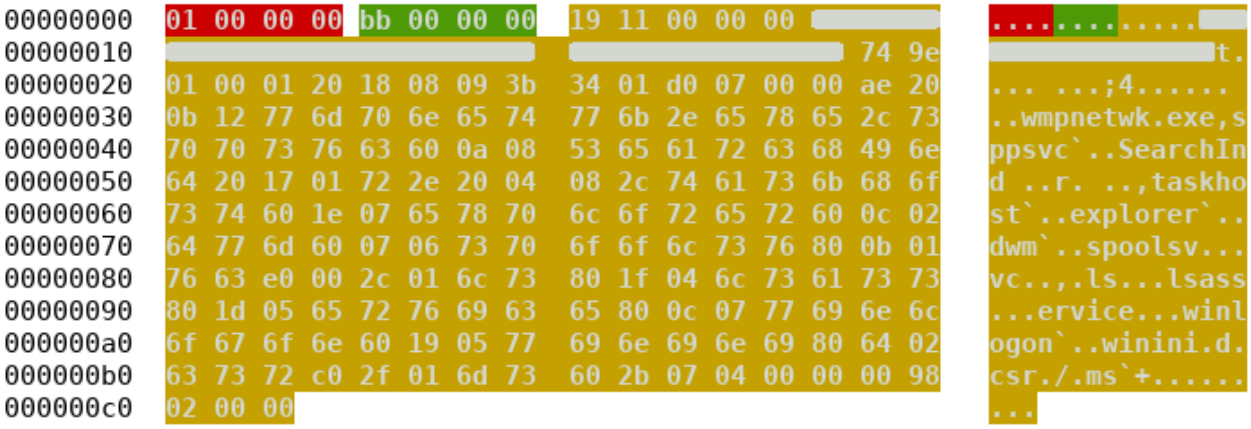
Additionally, an SHA-1 hash of the packets contents is also sent for integrity verification purposes.



The packet encryption structure

Packet structure

Command packets are compressed and encapsulated in a simple packet structure.




```
struct packet:
- command: 0x1
- packet_len: 0xbb
- packet_data: b'\x19\x11\x00\x00\x00[redacted]t\x9e\x01\x00\x01 \x18\x08
\t;4\x01\xd0\x07\x00\x00\xae \x0b\x12wmpnetwk.exe,sppsvc`\n\x08SearchInd \x17\x0
1r. \x04\x08,taskhost`\x1e\x07explorer`\x0c\x02dwm`\x07\x06spoolsv\x80\x0b\x01vc
\xe0\x00,\x01ls\x80\x1f\x04lsass\x80\x1d\x05ervice\x80\x0c\x07winlogon`\x19\x05w
inini\x80d\x02csr\xc0/\x01ms`+\x07\x04\x00\x00\x00\x98\x02\x00\x00'
```

Outer packet dissection presented using dissect.cstruct

Packet compression

Another change is the compression algorithm used for compressing and decompressing packet body.

Historically, the zlib algorithm has been used for that. It's hard to pinpoint the exact algorithm that is now used, but the procedure evolution_unpack[4] from quickbms project was found to correctly uncompress the data received from the C2 servers

```
  v4 = data;
  v5 = output;
  v19 = output;
  input_ending = &data[input_length];
  v6 = &output[output_len];
  for ( output_ending = &output[output_len]; ; v6 = output_ending )
  {
    v7 = *v4;
    in = v4 + 1;
    if ( v7 >= 0x20 )
      break;
    v9 = v7 + 1;
    if ( &v5[v9] > v6 )
      return 0;
    v19 = &v5[v9];
    qmemcpy(v5, in, v9);
    v4 = &in[v9];
    v5 += v9;
    v10 = output;
LABEL_10:
    if ( v4 >= input_ending )
      return v5 - v10;
  }
  v11 = v7 >> 5;
  v12 = (v7 & 0x1F) << 8;
  if ( v11 == 7 )
    v11 = *in++ + 7;
  v13 = *in;
  v4 = in + 1;
  v14 = &v5[-v12 - 1 - v13];
  o = &v5[-v12 - 1 - v13];
  if ( &v5[v11 + 2] <= output_ending )
  {
    v10 = output;
    if ( v14 >= output )
    {
      v15 = v11 + 2;
      qmemcpy(v5, o, v15);
      v5 = &v19[v15];
      v19 += v15;
      goto LABEL_10;
    }
  }
  return 0;
```

Pseudocode of the new algorithm used to uncompress packets

It was decided to reimplement the uncompression procedure in Python, the resulting script is listed below.

Register packet structure

As mentioned earlier, the protobuf structures have been abandoned in favour of custom structures.

One of the observed packet types is the command used to register the bot on the botnet and receive modules to execute.

The register packet structure can be easily presented using the following c struct:

```
00000000  11 00 00 00 66 6c 61 67  7b 75 5f 67 6f 74 5f 70  ....flag {u_got_p
00000010  77 6e 65 64 7d 74 9e 01  00 01 00 00 00 09 3b 34  wned}t........;4
00000020  01 d0 07 00 00 ae 00 00  00 77 6d 70 6e 65 74 77  .........wmpnetw
00000030  6b 2e 65 78 65 2c 73 70  70 73 76 63 2e 65 78 65  k.exe,sppsvc.exe
00000040  2c 53 65 61 72 63 68 49  6e 64 65 78 65 72 2e 65  ,SearchIndexer.e
00000050  78 65 2c 74 61 73 6b 68  6f 73 74 2e 65 78 65 2c  xe,taskhost.exe,
00000060  65 78 70 6c 6f 72 65 72  2e 65 78 65 2c 64 77 6d  explorer.exe,dwm
00000070  2e 65 78 65 2c 73 70 6f  6f 6c 73 76 2e 65 78 65  .exe,spoolsv.exe
00000080  2c 73 76 63 68 6f 73 74  2e 65 78 65 2c 6c 73 6d  ,svchost.exe,lsm
00000090  2e 65 78 65 2c 6c 73 61  73 73 2e 65 78 65 2c 73  .exe,lsass.exe,s
000000a0  65 72 76 69 63 65 73 2e  65 78 65 2c 77 69 6e 6c  ervices.exe,winl
000000b0  6f 67 6f 6e 2e 65 78 65  2c 77 69 6e 69 6e 69 74  ogon.exe,wininit
000000c0  2e 65 78 65 2c 63 73 72  73 73 2e 65 78 65 2c 73  .exe,csrss.exe,s
000000d0  6d 73 73 2e 65 78 65 04  00 00 00 98 02           mss.exe.....
```

```
struct hello_packet:
- bot_name_len: 0x11
- bot_name: b'[redacted]'
- os_version: 0x19e74
- session_id: 0x1
- magic: 0x1343b09
- some_another_magic: 0x7d0
- proclist_len: 0xae
- proclist: b'wmpnetwk.exe,sppsvc.exe,SearchIndexer.exe,taskhost.exe,explorer.exe,dwm.exe,spoolsv.exe,svchost.exe,lsm.exe,lsass.exe,services.exe,winlogon.exe,wininit.exe,csrss.exe,smss.exe'
- unknown_len: 0x4
- unknown: b'\x98\x02'
```

Register packet dissection presented using dissect.cstruct

Summary

The goal of this article was to help other researchers with their Emotet research after recent changes.

Emotet has once again proven to be an advanced threat capable of adapting and evolving quickly in order to wreak more havoc.

This article barely scratches the surface of the Emotet's inner workings, and should be treated as a good entry point, not as a complete guide. We encourage everyone to use this information, and hopefully share further results and/or discrupt the botnet's operations.

Further reading

References

1: https://d00rt.github.io/emotet_network_protocol/

2: https://github.com/JPCERTCC/EmoCheck

3: https://twitter.com/JRoosen/status/1225188513584467968

4: https://github.com/mistydemeo/quickbms/blob/master/unz.c#L5501