Emotet: Dangerous Malware Keeps on Evolving

By Threat Intel

Published: 2020-03-30 · Archived: 2026-04-06 01:29:21 UTC

Research into the latest Emotet variant by Symantec's Threat Engineering Team has revealed details about which compression algorithm the gang behind Emotet has customized and is using in its code.

12 min read

Mar 30, 2020

Authors: Nguyen Hoang Giang, Mingwei Zhang

Press enter or click to view image in full size

Emotet is one of the most dangerous malware threats active today. Emotet (Trojan.Emotet) began life as a banking Trojan but evolved several years ago to act as a malware loader for other threats — Emotet infects a machine and then downloads another threat e.g. the TrickBot information stealer, onto the infected system. Emotet is now one

of the biggest threat distributors out there, renting its infrastructure out to all sorts of other threats, including ransomware, information stealers, and cryptocurrency miners.

We wrote a detailed blog about Emotet's evolution in 2018, which gives you more background on the threat and its development.

Recently our threat engineering team noticed some updates to Emotet (Version 5, 20200201), and performed some analysis on the malware sample to see exactly what was going on.

What's New?

The first thing we noticed is that Emotet has updated its techniques for obfuscating its flow of code. This anti-analysis technique makes it more difficult to analyze and track modifications between variant binaries. The second thing is a change in the communication protocol between the botnet and its command and control (C&C) servers. A detailed technical analysis of these changes follows.

Anti-Analysis

Control Flow Flattening

This Emotet binary (unpacked) is using an obfuscation technique called Control Flow Flattening, which works as follows:

- Each basic block is assigned a number.
- The obfuscator introduces a block number variable, indicating which block should execute.
- Each block, instead of transferring control to a successor with a branch instruction, as usual, updates the block number variable to its chosen successor.
- The ordinary control flow is replaced with a switch statement over the block number variable, wrapped inside of a loop.

  Press enter or click to view image in full size

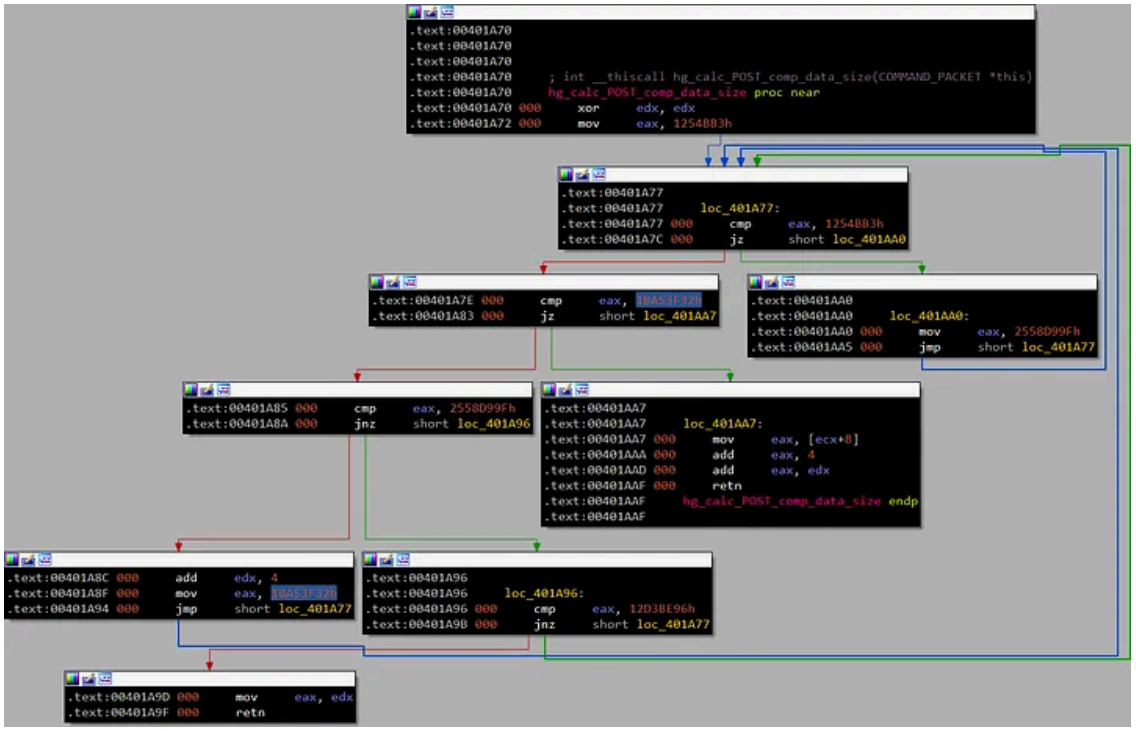

Figure 1. Control flow graph of one of the functions is obfuscated by Control Flow Flattening technique

Encrypted Strings and Resources

All strings and other resources (such as RSA public key) are encrypted and only decrypt at runtime. The decryption algorithm is described in *Figure 2*.

Press enter or click to view image in full size

Figure 2. How Emotet decrypts strings and resources

We can use IDAPython to create a script to decrypt the data:

```
def decrypt_data(ea):
    key1 = idc.get_wide_dword(ea)
    key2 = idc.get_wide_dword(ea + 4)
```

```
    size = key1 ^ key2

       data_size = (size + 3) & 0xFFFFFFFC
    ea += 8
    s = ''
    for i in range(data_size/4):
        dec_dword = idc.get_wide_dword(ea + i * 4) ^ key1
        s += struct.pack('<I', dec_dword)
    return s[:size]

Python>print(decrypt_data(0x40a810))
wininet.dll
```

Dynamic API Resolve

In this version, Emotet resolves API(s) by looking up the hashes of the API name and DLL name once it needs-to-use instead of loading them all at one time as previous versions. We observed that this behavior is similar to the code of Dridex or the Bitpaymer/ DopplePaymer ransomware:

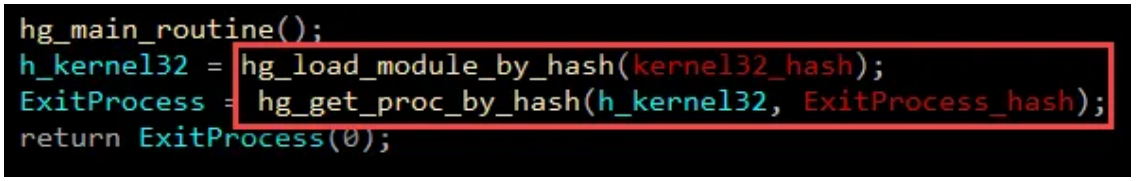

Figure 3. How Emotet loads and calls API ExitProcess by looking up hash values of kernel32 and ExitProcess

The customized hash function is calculated as follows:

```
def emotet_hash(api_name):
    i = 0
    for c in api_name:
        i = (i << 16) + (i << 6) + ord(c) - i
    # xor dword (0x165308FE) varies between binaries and different
    # between functions resolving api name hash and module name hash
    return (i & 0xFFFFFFFF) ^ 0x165308FE
```

Main Work

Upon first infection, the Emotet sample runs through two stages. During the first stage, the sample runs as a first instance, does some setup and checks the victim system, it then executes the second instance. The second instance will run in the second stage, where it communicates with embedded C&C server addresses in its binary.

First Stage — Dropper Instance

When it first runs, Emotet tries to decrypt information from additional DLLs that it requires to load. Among the DLLs it uses are:

- urlmon.dll
- userenv.dll
- wininet.dll
- shell32.dll
- crypt32.dll
- advapi32.dll
- wtsapi32.dll

Then, Emotet gets the volume serial number of the Windows partition. This volume serial number is used to create a series of mutex and event handles, with object names as follows (%X is the format of the volume serial number):

- Global\\I%X — MutexI
- Global\\M%X — MutexM
- Global\\E%X — EventE

A pair, EventE and MutexM, is created for synchronization between the first instance and second instance (by using API SignalObjectAndWait), to ensure that the second instance is only able to connect to the C&C servers once the first instance is exited.

Check Privilege and Delete Old Variant

Emotet checks its running privilege by calling API OpenSCManagerW with parameter SC_MANAGER_ALL_ACCESS, if this API call is successful, then the sample is considered to be running with high privilege.

Then the sample decrypts a list of words as below and uses the volume serial number to calculate and select two words from that list. It then combines them to get the filenames of old Emotet binaries that were dropped by Emotet's previous version. Depending on whether it is running with high privilege or not, the old binary with that filename will be deleted from CISIDL_SYSTEMX86 or CSIDL_LOCAL_APPDATA.

```
duck,mfidl,targets,ptr,khmer,purge,metrics,acc,inet,msra,symbol,driver,sidebar,restore,msg,volume,ca
```

If the sample is running with high privilege, it checks if its running filename contains a number, if so, it will try to delete any other files with the same filename. It looks like this latest sample of Emotet is trying to remove old traits of the previous version(s).

Check Setting From Registry

For its next step, Emotet checks the registry value name (is volume serial number) in:

- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Explorer — If Emotet is running with high privilege

- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer — If Emotet is running with low privilege

The data of this registry value name is the encrypted filename of the Emotet binary that will be dropped by the current version. It is XOR-encrypted and the key is the volume serial number. If this registry value is not available, it means that the Emotet sample is running in its first stage and it will set this value for use in the second instance.

Press enter or click to view image in full size

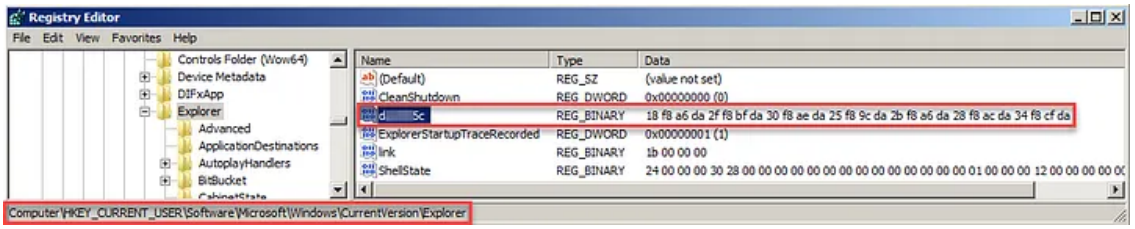

Figure 4. Data is saved in the registry as the encrypted filename of dropped file

Filename of New Dropped File

To drop the Emotet binary for the second stage, Emotet needs to get a filename. This sample will scan in CSIDL_SYSTEM, only filenames with the extensions .dll and .exe are selected. This is similar to the Bitpaymer ransomware, which also clones an executable file in the system directory to hide its binary in the ADS stream of the cloned file. This new filename is saved to the registry as previously described.

Press enter or click to view image in full size

```
h_shell32 = hg_load_module_by_hash(shell32_hash);
SHGetFolderPathW = hg_get_proc_by_hash(h_shell32, SHGetFolderPathW_hash);
SHGetFolderPathW(0, CSIDL_SYSTEM, 0, 0, system_dir);
hg_recursive_enum_files(0, system_dir, hg_check_file_extension_hash_callback, &fnames_list);
n = 0;
for ( i = fnames_list; i; ++n )
{
  i = *i;
}
h_kernel32 = hg_load_module_by_hash(0xE40D4AEB);
GetTickCount = hg_get_proc_by_hash(h_kernel32, GetTickCount_hash);
rand_num = GetTickCount();
```

Figure 5. Emotet scans for filenames of .exe and .dll files in %SYSTEM% to generate filename of dropped file

Emotet then makes a path to drop itself to. If it is running with high privilege, it drops the binary to CSIDL_SYSTEMX86, otherwise, it drops it to CSIDL_LOCAL_APPDATA (using API SHFileOperation with the parameter FO_MOVE).

Press enter or click to view image in full size

```
sh_file_op.wFunc = FO_MOVE;
sh_file_op.pFrom = from;
sh_file_op.pTo = to;
// FOF_SILENT | FOF_NOCONFIRMATION | FOF_NOCONFIRMMKDIR | FOF_NOERRORUI | FOF_NOCOPYSECURITYATTRIBS
sh_file_op.fFlags = 0xE14;
h_shell32 = hg_load_module_by_hash(shell32_hash);
SHFileOperationW = hg_get_proc_by_hash(h_shell32, SHFileOperationW_hash);
return SHFileOperationW(&sh_file_op) == 0;
```

Figure 6. Emotet calls API SHFileOperation to clone itself to destination path

The path of the dropped file is:

- “(CSIDL_SYSTEMX86|CSIDL_LOCAL_APPDATA)\\%s\\%s.exe” % (new_filename, new_filename)

When it is running with high privilege, Emotet gains persistence by creating a service for the dropped file:

- “CSIDL_SYSTEMX86\\%s\\%s.exe” % (new_filename, new_filename)

```
Service name: new_filename
Service display name: new_filename
```

Otherwise, Emotet gains persistence in the registry by setting a registry value (only after it has its first connected to its C&C servers):

- HKEY_CURRENT_USER\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run

```
Value name: new_filename
Value data: CSIDL_LOCAL_APPDATA\\%s\\%s.exe % (new_filename, new_filename)
```

Finally, Emotet launches the second instance by calling API CreateProcessW to the dropped file.

Second Stage — Bot Instance

The second instance goes through the same steps as the first instance until it is able to get the saved data of the filename from the registry. Then it performs some checks before communicating with the C&C servers:

- Checks current filename is the same as the filename saved in the registry. If not, it runs similar to the first stage to launch another instance.
- Check if its parent process is services.exe, meaning it is running as a service. If it is, it runs similar to the first stage to launch another instance.

The communication protocol is changed in this version. We will describe this in detail below.

Get List C&C IP Port / RSA Public Key and Generate AES Key

In this version, the IP addresses and ports of the C&C servers continue storing in binary as 8-byte blocks.

Press enter or click to view image in full size

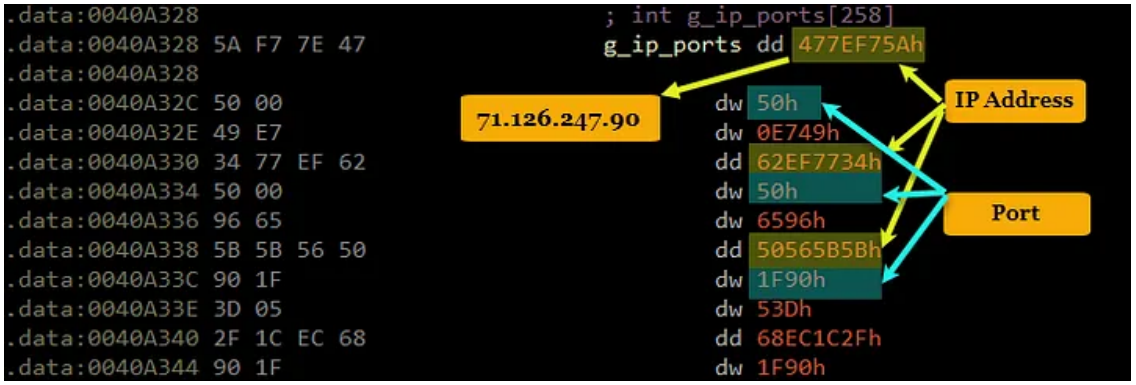

Figure 7. IP(s) and port(s) of C&C servers are embedded in binary

Emotet retrieves the RSA public key from encrypted data. RSA public key is in PEM format:

```
-----BEGIN PUBLIC KEY-----
MHwwDQYJKoZIhvcNAQEBBQADawAwaAJhANQOcBKvh5xEW7VcJ9totsjdBwuAclxS
Q0e09fk8V053lktpW3TRrzAW63yt6j1KWnyxMrU3igFXypBoI4lVNmkje4UPtIIS
fkzjEIvG1v/ZNn1k0J0PfFTxbFFeUEs3AwIDAQAB
-----END PUBLIC KEY-----
```

Then the sample will generate an AES-128-CBC session key handle and an SHA-1 hash key handle. The RSA public key, AES-128-CBC Key, and SHA-1 hash are combined to secure the connection between Emotet samples and the C&C servers.

Press enter or click to view image in full size

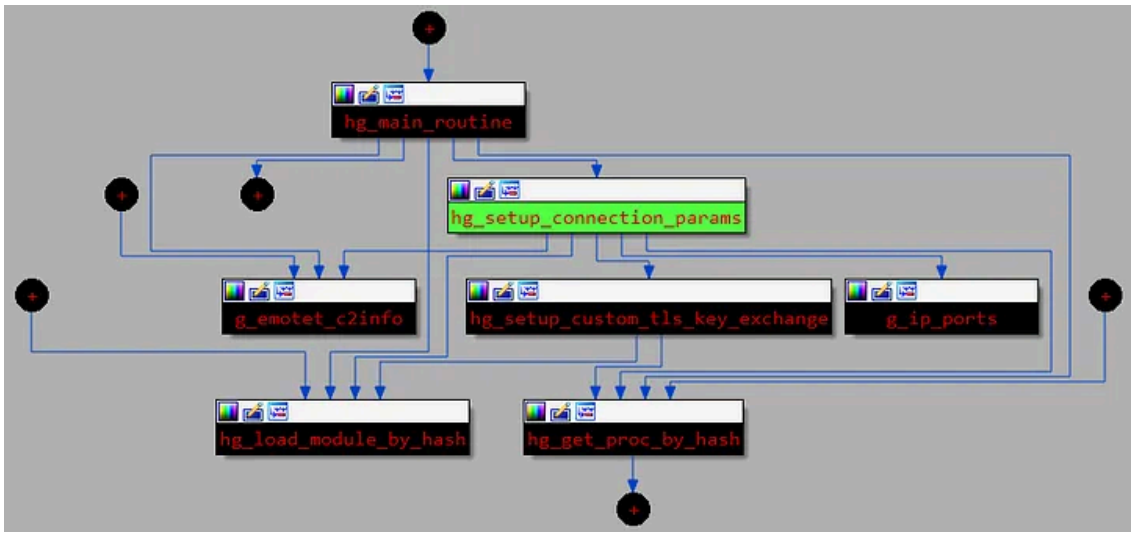

Figure 8. Emotet is retrieving IP/Port list and generating crypto key handles to secure communication

Setup Post Request

Plaintext Packet

Get Threat Intel's stories in your inbox

Next, the Emotet sample starts building a plaintext packet for the request. This is basic information to generate packets after that. The plaintext packet has the following format:

```
struct plain_packet
{
    uint32_t victim_id_size;
    uint8_t  victim_id[victim_id_size];  // hostname + hex volume serial number
    uint32_t system_info;
    uint32_t session_id;
    uint32_t bot_id;                     // 0x1343B09 or 20200201 in decimal - it looks like the dat
    uint32_t unknown_id;                 // 0x7D0 or 2000 in decimal
    uint32_t procname_buffer_size;
    uint8_t  procname_buffer[procname_buffer_size];
    uint32_t module_id_array_size;
    uint8_t  module_id_array[module_id_array_size];
}
```


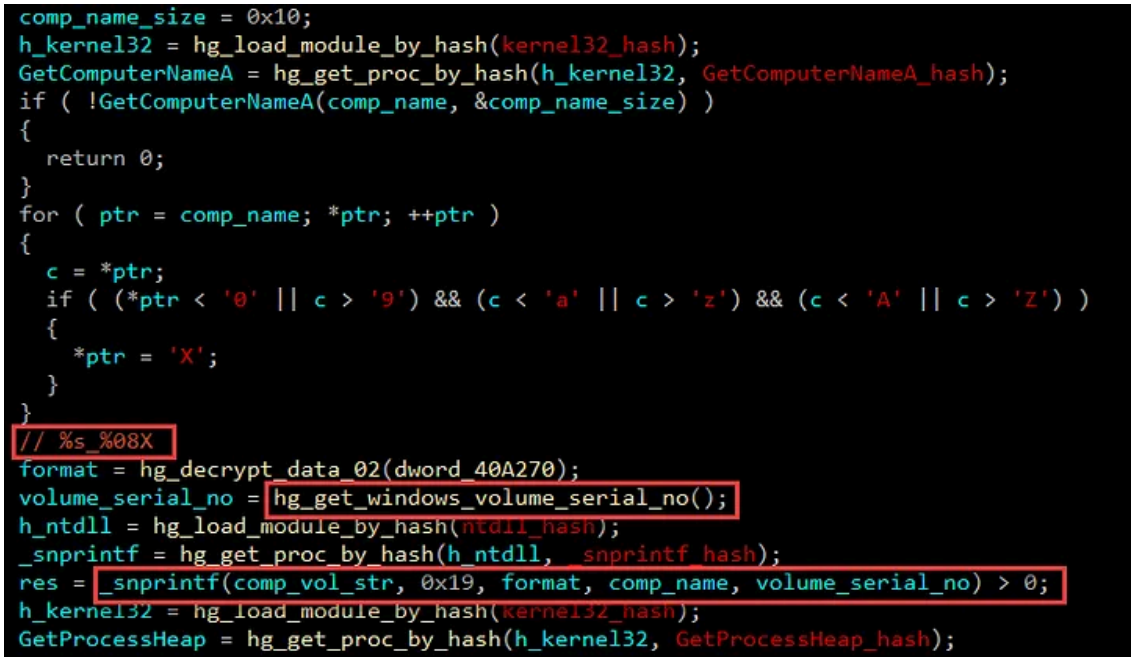

Figure 9. Emotet is generating VICTIM_ID based on computer name and volume serial number

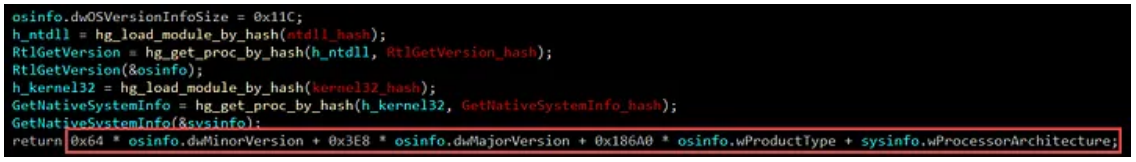

Figure 10. Emotet is calculating system info value based on OS version, product type, and processor architecture

Press enter or click to view image in full size

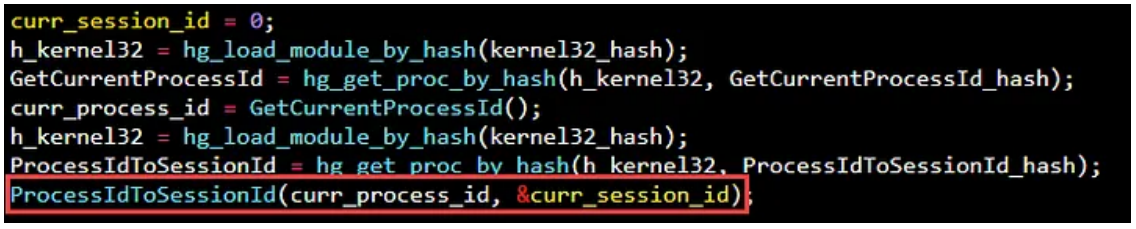

Figure 11. Emotet gets session ID from its process ID

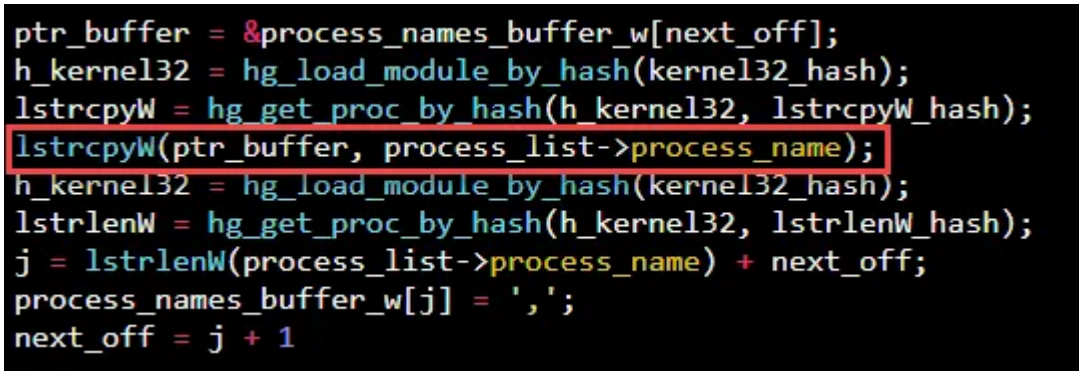

Figure 12. Emotet enumerates running process names (no duplicated process names, no process names where parent process ID is 0, and not including Emotet's process name) puts them in buffer, separated by comma, and in ASCII

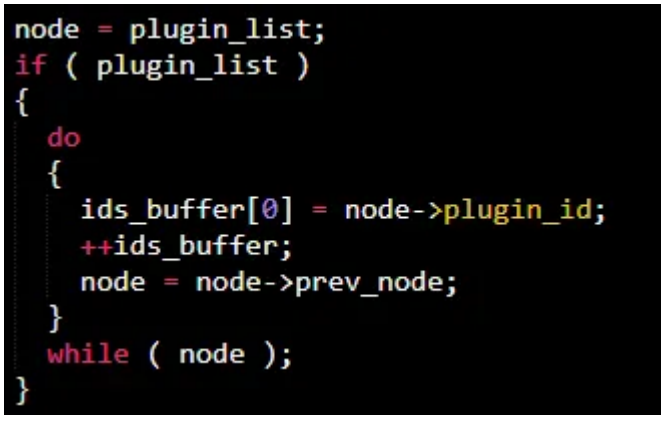

Figure 13. Emotet enumerates ID of plugins, which it downloads from C&C and executes. Plugin ID(s) are saved to buffer as DWORD(s).

Press enter or click to view image in full size

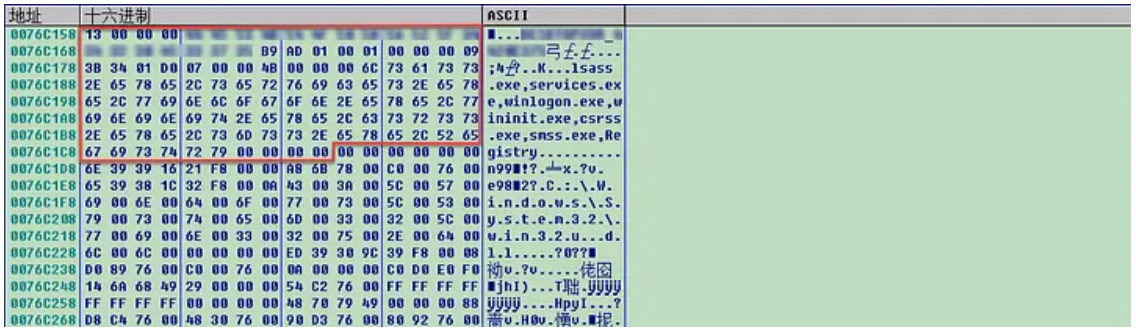

Figure 14. Full buffer of plaintext packet, which Emotet created

Emotet then compresses this plaintext packet by using an unknown compression algorithm. After digging deeply into this compression algorithm, we can point out which compression library the actors behind Emotet are using and what they customized in that library. Let's keep going and look at how the compressed data is repacked to a command packet.

Client Command Packet

The command package is composed of the compressed data of the plaintext packet:

```
struct client_cmd_packet
{
    uint32_t cmd_id;            // 0x01 for register command
    uint32_t comp_data_size;
    uint8_t  comp_data[comp_data_size];
}
```

Finally, Emotet performs encryption on the command packet (by AES-128-CBC) to generate a final packet, which is posted to the C&C servers through HTTP POST. The format of the final packet is as follows:

```
struct final_packet
{
    uint8_t enc_session_key[0x60];  // reversed order of encrypted session key (AES-128-CBC key is e
    uint8_t comp_data_hash[0x14];   // SHA-1 hash of Command Packet (before encryption)
    uint8_t encrypted_data[];       // AES-128-CBC encrypted data of Command Packet
}
```

Press enter or click to view image in full size

Figure 15. Emotet repackages the final packet before embedding it to a POST header request

At this point, Emotet has gotten the data to post it to the C&C server. Next, it sets up fields in the POST header.

Generate URI Path

This URI path is composed of 1–6 substrings. Each substring is 4–19 bytes and is randomly generated by selecting from [A-Za-z1–9]. The referrer path is the same as the URI path.

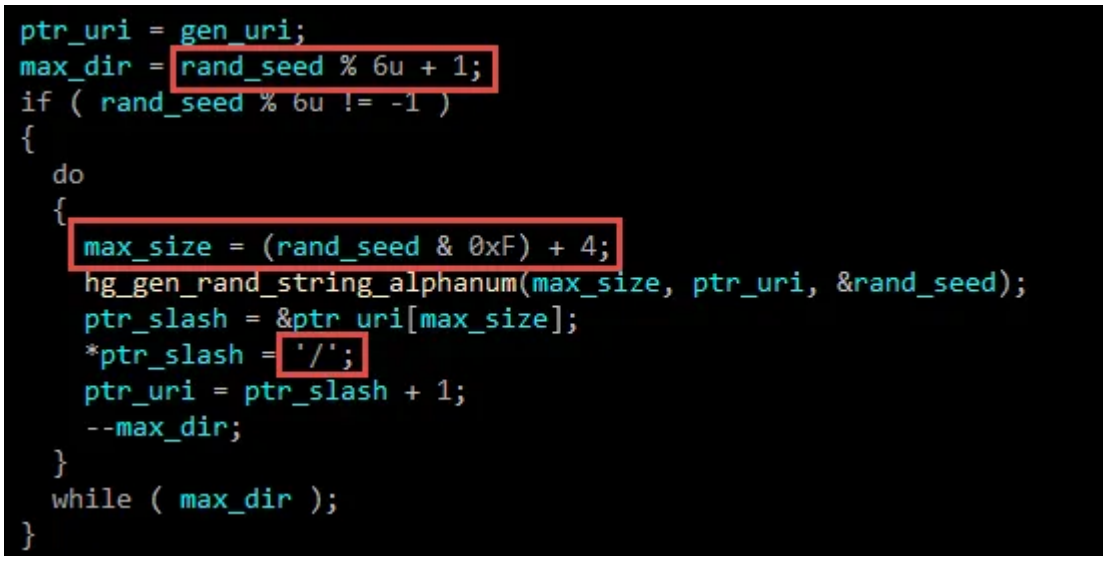

Figure 16. Emotet generates subpaths for the POST request

Referrer Path

The referrer path is set by the IP address of the C&C server and generated URI path above.

```
Referrer: http://%s/%s\r\n
```

Multipart/Form-Data

Currently, instead of sending the final packet as a simple POST body, Emotet submits that data by encoding it in multipart/form-data. The multipart/form-data has the following format**:**

```
Content-Type: multipart/form-data; boundary=%s\r\n\r\n--%S\r\nContent-Disposition: form-data; name="
```

Boundary

Generated by random numbers and written in the following format:

```
--------------------------%04u%04u%04u%03u
```

Press enter or click to view image in full size

```
rand_04 = RtlRandomEx(&rand_seed);
num_01 = rand_01 % 0x3E8;
num_04 = rand_04 % 0x2710;
h_ntdll = hg_load_module_by_hash(ntdll_hash);
_snwprintf = hg_get_proc_by_hash(h_ntdll, _snwprintf_hash);
_snwprintf(boundary, 0x40, boundary_format, num_04, rand_03 % 0x2710, rand_02 % 0x2710, num_01);
```

Figure 17. Emotet generates boundary string

Form Name and Attachment Filename

This form name is generated randomly. It is composed of 4–19 bytes and is randomly generated by selecting from [A-Za-z]. Similarly, the attachment filename is generated with the same algorithm.

```
h_ntdll = hg_load_module_by_hash(ntdll_hash);
RtlRandomEx = hg_get_proc_by_hash(h_ntdll, RtlRandomEx_hash);
max_size = RtlRandomEx(&rand_seed) & 0xF + 4;
hg_gen_rand_string_alphabet_a(max_size, rand_name, &rand_seed);
rand_name[max_size] = 0;
h_ntdll = hg_load_module_by_hash(ntdll_hash);
RtlRandomEx = hg_get_proc_by_hash(h_ntdll, RtlRandomEx_hash);
max_size = RtlRandomEx(&rand_seed) & 0xF + 4;
hg_gen_rand_string_alphabet_a(max_size, rand_filename, &rand_seed);
rand_filename[max_size] = 0;
```

Figure 18. Emotet generates form name and filename of multipart/form-data

Now, Emotet is ready to POST data to a C&C server. If the C&C server address is live and replies to the message, Emotet will parse and decrypt the message by AES-128-CBC then verify the SHA-1 hash of the decrypted data using the RSA public key.

Responded Packet

The respond packet is formatted as follows:

```
struct c2_resp_packet
{
    uint8_t signature[0x60];   // signature data to be verified
    uint8_t decrypted_data_hash[0x14];  // SHA-1 hash of encrypted data after decryption
    uint8_t encrypted_data[];  // AES-128-CBC encrypted data of the C&C's responded message
}
```

Press enter or click to view image in full size

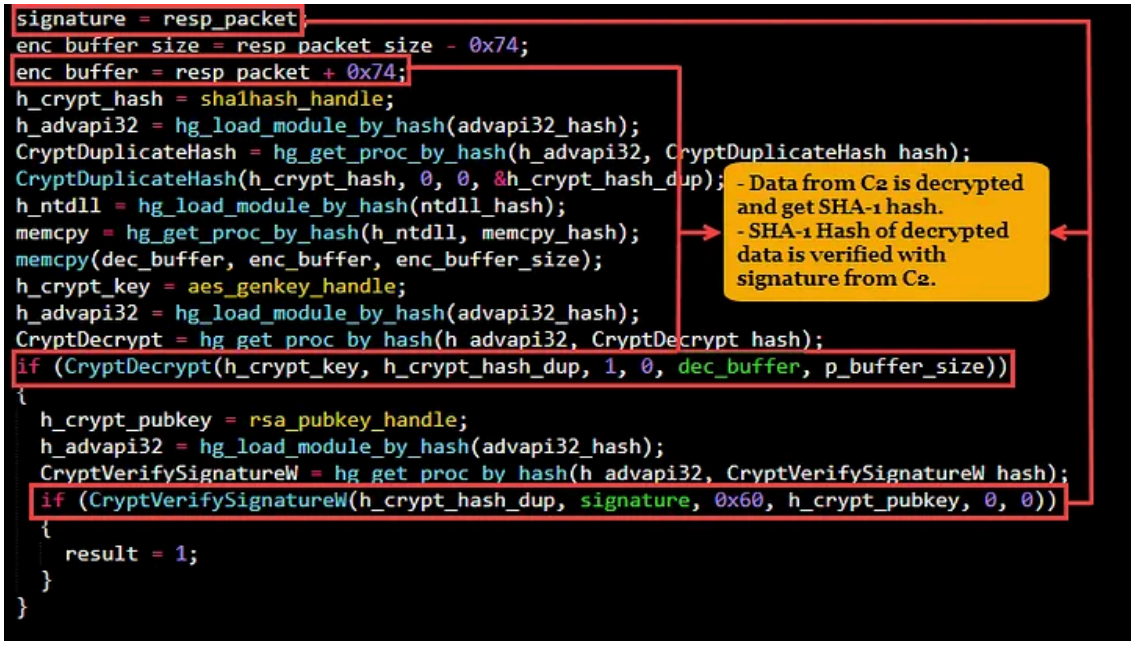

Figure 19. Emotet receives a packet from the C&C, decrypts and verifies it

After receiving valid decrypted data, Emotet decompresses it and then parses commands and data from the decompressed packet. Data from the decompressed packet is a chain of blocks of the C&C server's command packets:

```
struct decomp_data
{
    uint32_t cmd_packet1_size;          // size of cmd_packet1 data
    uint8_t  cmd_packet1[cmd_packet1_size];
    uint32_t cmd_packet2_size;
    uint8_t  cmd_packet2[cmd_packet2_size];
    .....
    uint32_t cmd_packetN_size;
    uint8_t  cmd_packetN[cmd_packetN_size];
}
```

Each block of the C&C server's command packets is formatted as follows:

```
struct c2_cmd_packet
{
    uint32_t id;                    // plugin id
    uint32_t command;               // command id
    uint32_t payload_size;          // payload size
    uint8_t  payload[payload_size]; // payload data
}
```

Currently, this Emotet sample receives three commands from the C&C server:

- Command ID 01: Downloads an executable file and executes it
- Command ID 02: Downloads an executable file, checks if there is another active Terminal Service session other than the current session identifier of the running Emotet sample, then launches downloaded executable in that active Terminal Service session:

```
- Calls API WTSQueryUserToken to obtain the Primary User token of the requested Terminal Service ses
- Calls API CreateProcessAsUser to launch process
```

- Command ID 03: Downloads a module/plugin, loads it and calls to its main function

Compression/Decompression Library

As we already mentioned, this Emotet sample is using an unknown compression library to compress packets to send to the C&C server and to decompress packets received from the C&C server. This is one of the recent changes seen in the current Emotet sample as, in previous versions, Emotet was using the zlib library. Although we can easily reverse engineer and re-implement the compression and decompression routines from the current Emotet binary, uncovering exactly which compression library it is using helps us understand more about the sample and implement compression and decompression exactly as the sample does.

The compression library is actually a well-known library named LibLZF. But the people behind Emotet made some modifications to that library when they integrated it into the current code base of this bot. We will highlight the modifications implemented below.

Browsing to the source code of LibLZF, we noticed this comment in file lzf_c.c:

```
/*
 * don't play with this unless you benchmark!
 * the data format is not dependent on the hash function.
 * the hash function might seem strange, just believe me,
 * it works ;)
 */
#ifndef FRST
# define FRST(p) (((p[0]) << 8) | p[1])
# define NEXT(v,p) (((v) << 8) | p[2])
# if ULTRA_FAST
#  define IDX(h) ((( h            >> (3*8 - HLOG)) - h  ) & (HSIZE - 1))
# elif VERY_FAST
#  define IDX(h) ((( h            >> (3*8 - HLOG)) - h*5) & (HSIZE - 1))
# else
#  define IDX(h) ((((h ^ (h << 5)) >> (3*8 - HLOG)) - h*5) & (HSIZE - 1))
# endif
#endif
```

Figure 20. LibLZF defines how to calculate values

Interestingly, this library's decompression routine does not depend on the hash function used in the compression routine. The hash function is calculated in lzf_c.c (source code of compression routine):

```
#if ULTRA_FAST || VERY_FAST
          --ip;
# if VERY_FAST && !ULTRA_FAST
          --ip;
# endif
          hval = FRST (ip);

          hval = NEXT (hval, ip);
          htab[IDX (hval)] = ip - LZF_HSLOT_BIAS;
          ip++;

# if VERY_FAST && !ULTRA_FAST
          hval = NEXT (hval, ip);
          htab[IDX (hval)] = ip - LZF_HSLOT_BIAS;
          ip++;
# endif
#else
          ip -= len + 1;

          do
            {
              hval = NEXT (hval, ip);
              htab[IDX (hval)] = ip - LZF_HSLOT_BIAS;
              ip++;
            }
          while (len--);
#endif
```

Comment this block

Figure 21. How the hash table is calculated in the compression routine of LibLZF

And in the current Emotet binary, LibLZF is compiled with these modifications:

In file lzfP.h:

- Set ULTRA_FAST is 0
- Set VERY_FAST is 0

In file lzf_c.c:

- Comment out to block code as in *Figure 21*

From this finding, we grabbed a project that is binding LibLZF to Python, changed the code as described above, and compiled it. Figure 22 shows the result of our testing with that Python compiled module on data we dumped from the Emotet sample's memory:

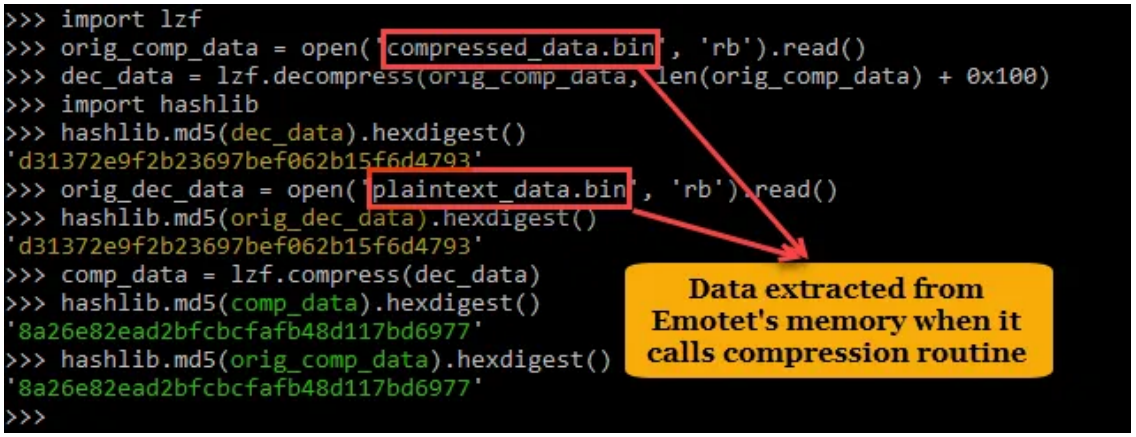

Figure 22. Verify compressed/decompressed data with Python compiled module

Conclusion

Over time, the Emotet botnet has evolved and will no doubt continue to evolve in the future. It has proven itself to be an extremely effective weapon for cyber criminals, and is one of the most dangerous botnets active today.

IOC

Analyzed Sample:

aa0cbe599839db940f6cc2f4ca1383dbb9937b8c7dd6460847c983523cd63c39

References/Further Reading

Control flow flattening: https://github.com/obfuscator-llvm/obfuscator/wiki

Control flow flattening write-up by Rolf Rolles: http://www.hexblog.com/?p=1248

LibLZF library (by Marc Lehmann): http://oldhome.schmorp.de/marc/liblzf.html

https://research.checkpoint.com/2018/emotet-tricky-trojan-git-clones/

https://www.cert.pl/en/news/single/whats-up-emotet/

Check out the Security Response blog and follow Threat Intel on Twitter to keep up-to-date with the latest happenings in the world of threat intelligence and cyber security.

Like this story? Recommend it by hitting the heart button so others on Medium see it and follow Threat Intel on Medium for more great content.
