Awaiting the Inevitable Return of Emotet

hornetsecurity.com/en/security-information/awaiting-the-inevitable-return-of-emotet/

Security Lab

May 5, 2020

Summary

Emotet is probably the most prolific of the recent malware distribution operations. They often change their malware to ensure it is not detected by any anti-virus software. Even though the Emotet botnet is on "spam break" recent changes in a component of the malware has prompted Hornetsecurity's Security Lab to take a look at the latest version of Emotet in order to be prepared for its next steps. Emotet has added new code obfuscation techniques. But the Security Lab explains how it can still be analyzed.

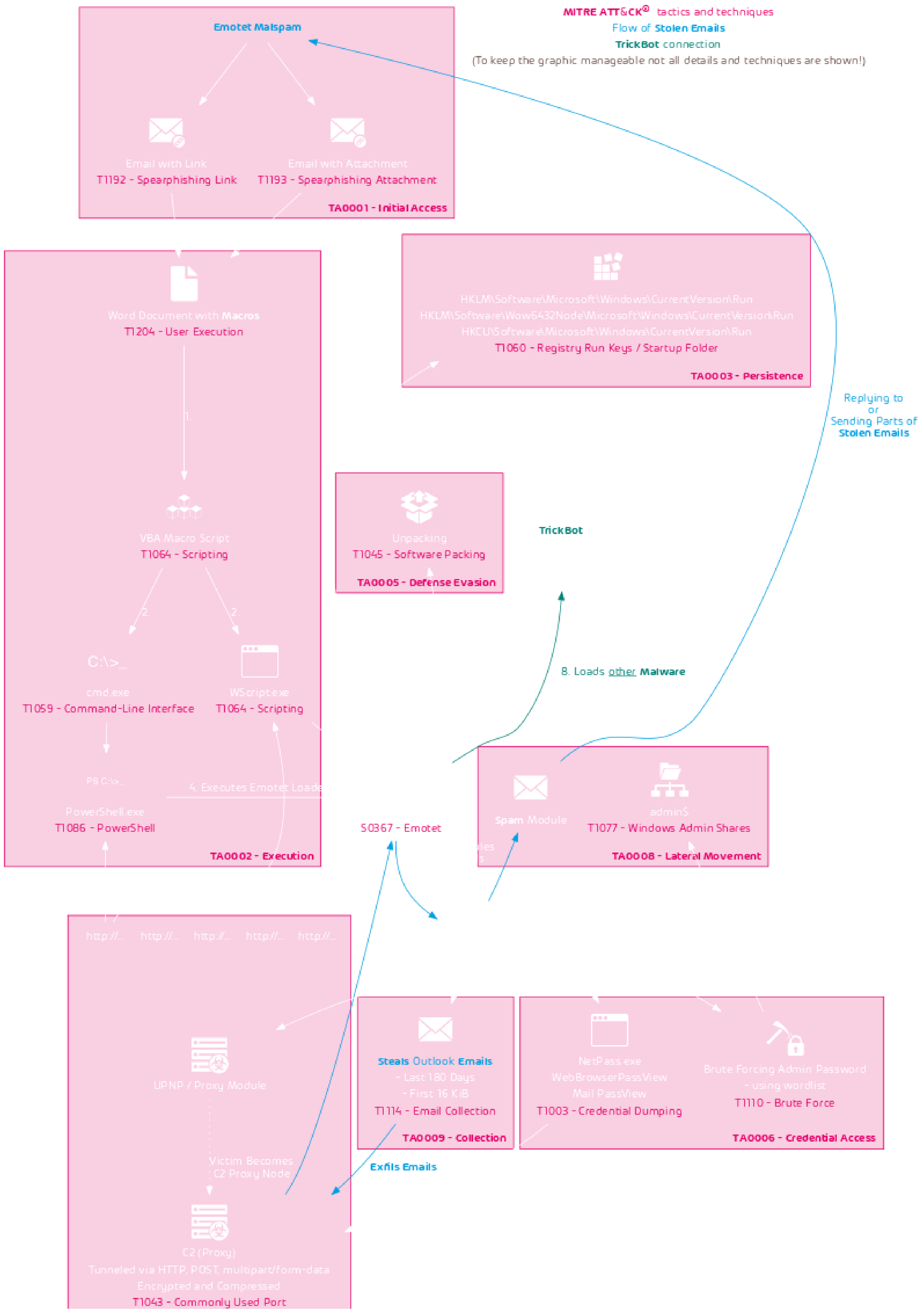
The updates to Emotet's loader do not impact Hornetsecurity's filters as the Emotet loader is never send directly attached to an email. However, the presented analysis and downloadable Ghidra scripts can help other researchers to jump start their Emotet reverse engineering despite the added obfuscation.

Background

The malware now commonly known as Emotet was first observed in 2014. It was a banking trojan stealing banking details and banking login credentials from victims. But it pivoted to a malware-as-a-service (MaaS) operation providing malware distribution services to other cybercriminals.

Emotet Infection Chain

At least the initial portion of the Emotet infection chain and its used tactics and techniques as defined by the MITRE ATT&CK framework are outlined in the following flow diagram:


MITRE ATT&CK® tactics and techniques
Flow of Stolen Emails
TrickBot connection
(To keep the graphic manageable not all details and techniques are shown!)
Emotet Malspam
Email with Link
T1192 - Spearphishing Link
Email with Attachment
T1193 - Spearphishing Attachment
TA0001 - Initial Access
Word Document with Macros
T1204 - User Execution
1.
VBA Macro Script
T1064 - Scripting
2.
2.
C:\>_
cmd.exe
T1059 - Command-Line Interface
WScript.exe
T1064 - Scripting
PS C:\>_
PowerShell.exe
T1086 - PowerShell
4. Executes Emotet Loade
TA0002 - Execution
HKLM\Software\Microsoft\Windows\CurrentVersion\Run
HKLM\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Run
HKCU\Software\Microsoft\Windows\CurrentVersion\Run
T1060 - Registry Run Keys / Startup Folder
TA0003 - Persistence
Unpacking
T1045 - Software Packing
TA0005 - Defense Evasion
TrickBot
8. Loads other Malware
Replying to
or
Sending Parts of
Stolen Emails
Spam Module
admin$
T1077 - Windows Admin Shares
TA0008 - Lateral Movement
S0367 - Emotet
http://... http://... http://... http://... http://...
UPNP / Proxy Module
Victim Becomes
C2 Proxy Node
C2 (Proxy)
Tunneled via HTTP, POST, multipart/form-data
Encrypted and Compressed
T1043 - Commonly Used Port
Steals Outlook Emails
- Last 180 Days
- First 16 KiB
T1114 - Email Collection
TA0009 - Collection
Exfils Emails
NetPass.exe
WebBrowserPassView
Mail PassView
T1003 - Credential Dumping
Brute Forcing Admin Password
- using wordlist
T1110 - Brute Force
TA0006 - Credential Access

T1065 - Uncommonly Used Port
T1094 - Custom Command and Control Protocol
T1032 - Standard Cryptographic Protocol
TA0011 - Command and Control

Of particular interest is that Emotet steals emails from victims and uses them as templates for new malspam. It uses what is known as email thread hijacking where it replies to old email threads with one of its malicious emails. Victims are much more likely to open emails from known correspondents and even more likely when that email is received in the context of an existing email conversation thread. As a result Emotet distribution campaigns have been very successful. These kind of attacks are one of the main reason why security leaders need to invest into security awareness training to mitigate IT security risks through people.

Emotet is so dangerous because in addition to its own modules to steal victims emails and misuse their computers as C2 and spam servers it delivers other malware, such as TrickBot, which ultimately leads to a Ryuk ransomware infection. So even if a victim cleans the Emotet infection they may already have additional malware running on their system(s) that are **not** the initial Emotet infection that they cleaned.

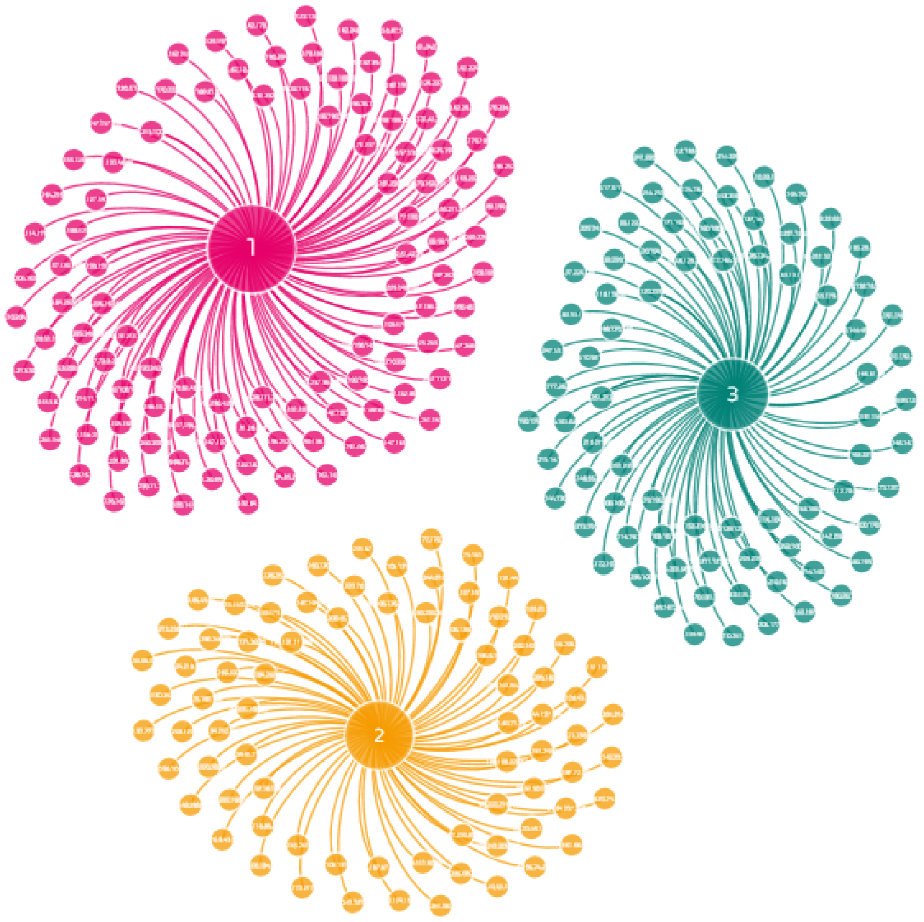
When victims become infected with Emotet they become part of the Emotet botnet.

Botnet

The Emotet botnet is split into separate botnets. Researchers called them Epoch 1 and 2 because they received payload updates at different times. Each Epoch has its own unique RSA key used for its C2 communication. On 2019-09-17 a portion of the Epoch 1 botnet was separated into the Epoch 3 botnet.

Each bot connects to C2 servers of its Epoch. If a victim is infected by an Emotet document belonging to Epoch 1, the document will download the Emotet loader from the Epoch 1 infrastructure and subsequently become part of Epoch 1.

The current structure of the Emotet botnet's Tier 1 C2 servers is as follows:

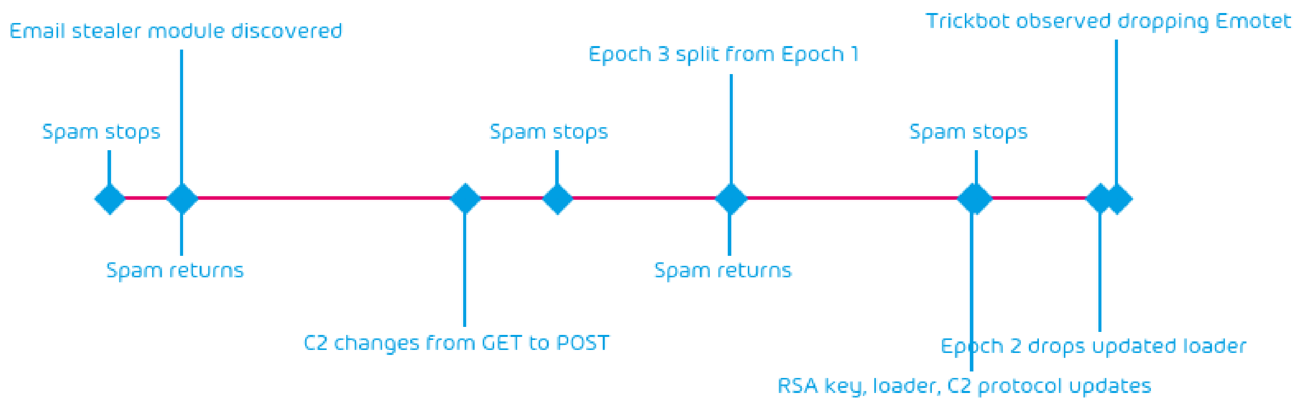


Changes are deployed to the E2 botnet first. It is possible that this is done as a test to ensure that in case of introduced breaking changes only one part of the total botnet is lost.

(Recent) History

While we could dig deep into the entangled history of Emotet, its origins and shared code with Feodo, or its relations to Cridex and Dridex, we rather focus on the more recent history.

As hopefully everyone knows that currently (at the time of writing) the Emotet botnet does not send spam. The botnet often has these breaks. But it often comes back from such a break with updates rendering it more dangerously as before. A timeline of recent "spam breaks" is as follows:


Email stealer module discovered
Spam stops
Spam returns
C2 changes from GET to POST
Spam stops
Epoch 3 split from Epoch 1
Spam returns
Spam stops
RSA key, loader, C2 protocol updates
Epoch 2 drops updated loader
Trickbot observed dropping Emotet

Returning 2018-10-31 Emotet delivered a new Email Stealer Module. Returning 2019-09-16 Epoch 3 was split from Epoch 1. In preparation of this publication on 2020-04-29 TrickBot (a notorious malware often distributed via Emotet) has been observed dropping the Emotet loader. Speculations are that due to the long absence in Emotet malspam, which is used to seed the botnet with new bots, and current infections being constantly cleaned, the number of Emotet bots may have shrunk to a critical low number, so they are seeded by the operators behind the TickBot malware in a quid-pro-quo for Emotet's TrickBot distribution efforts. But these are only speculations and nothing clear is known at this time.

What is known is that now with changes to the Emotet loader being dropped from Epoch 2 since 2020-04-20 the questions are when and with what new tricks will Emotet return this time. Hence, Hornetsecurity's Security Lab has analyzed the recent changes in the Emotet malware.

Technical Analysis

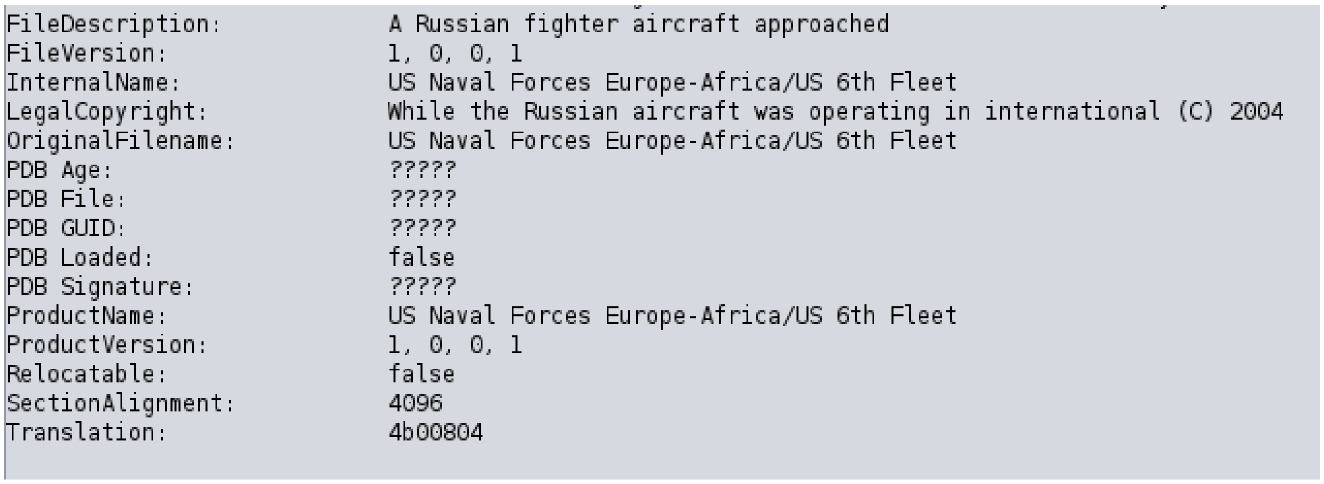
Due to a lack of current Emotet spam we can not analyze any malicious emails or documents. We therefore present an analysis of the Emotet loader binary.

Emotet Loader Binary

We analyzed both the "old" version that was dropped from Epoch 1 on 2020-02-20 (which has not received any notable changes since the 2020-02-05 update), and the "new" version dropped from Epoch 2 on 2020-02-20 onwards.

Packer

The metadata of the new version contains fragments of news article text. The File Description, Internal Name, Original Filename, Product Name, and Copyright texts are filled with fragments of news articles:


```
FileDescription:            A Russian fighter aircraft approached
FileVersion:                1, 0, 0, 1
InternalName:               US Naval Forces Europe-Africa/US 6th Fleet
LegalCopyright:             While the Russian aircraft was operating in international (C) 2004
OriginalFilename:           US Naval Forces Europe-Africa/US 6th Fleet
PDB Age:                    ?????
PDB File:                   ?????
PDB GUID:                   ?????
PDB Loaded:                 false
PDB Signature:              ?????
ProductName:                US Naval Forces Europe-Africa/US 6th Fleet
ProductVersion:             1, 0, 0, 1
Relocatable:                false
SectionAlignment:           4096
Translation:                4b00804
```

This is another indicator that Emotet uses the same packer as Trickbot, which has been seen with news article text in its metadata shortly before Emotet featured news article texts. Researchers have duped this crypter “Exes’r’rus” [JRoosen].

Because malware packers frequently change, this change is not unusual. It has been seen around 2020-02-14, but The Emotet loader payload can still be extracted via generic (and automated) unpacking mechanisms.

Obfuscation

Emotet added more obfuscation. The most notable change is control flow flattening and junk code. This makes the binary more annoying to analyze as well as allow for simpler polymorphic changes. Most of the obfuscation was already added in the 2020-02-05 change.

Control Flow Flattening

Control flow flattening is an obfuscation technique. It splits a code sequence into multiple parts and rearranges then in a way that is more difficult to follow. A common way is to place the code parts into a loop and on each loop run executing different code parts in the loop body determined by a state variable.

The code sequence:

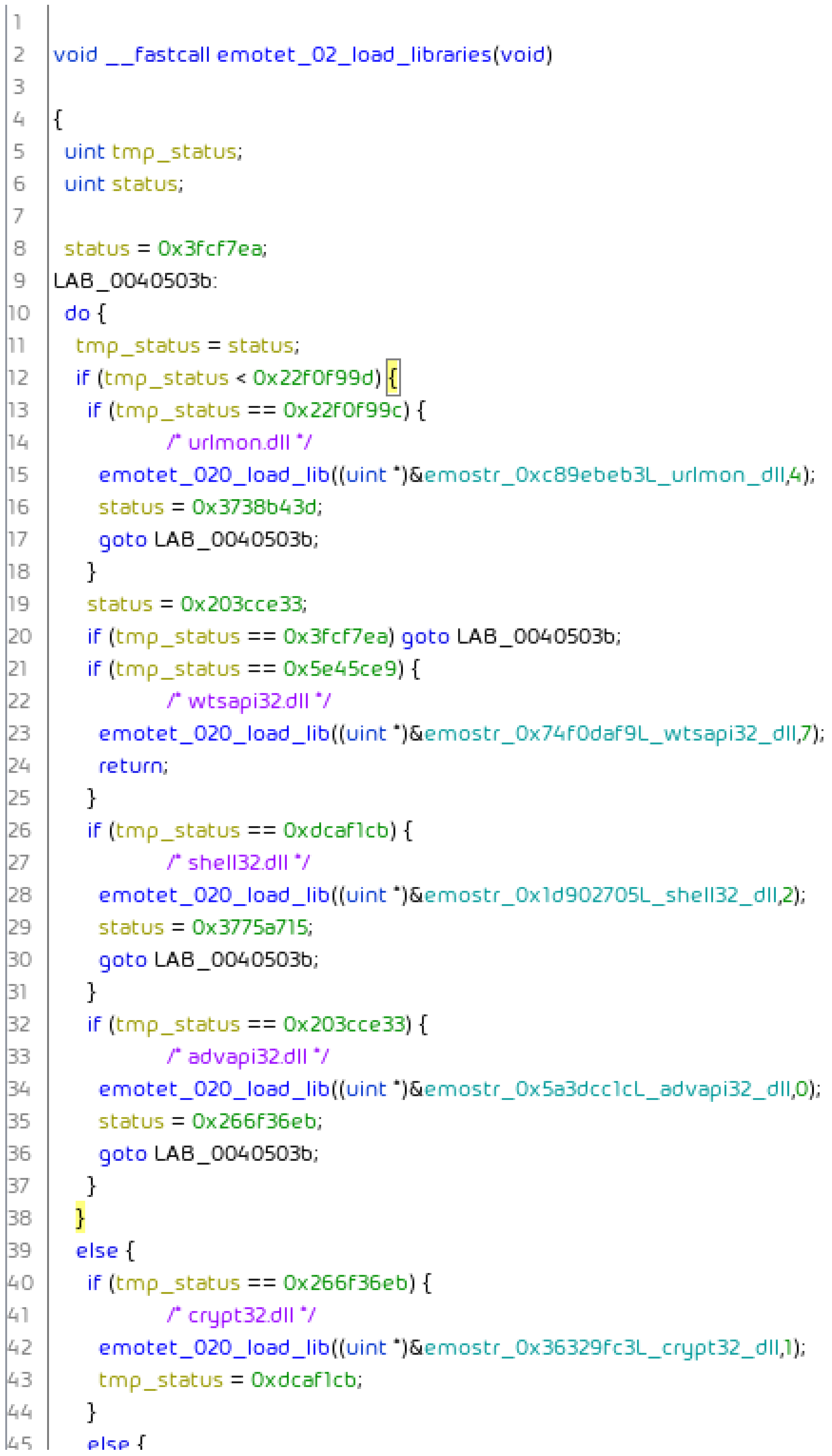
```
CODE_1;
CODE_2;
CODE_3;
return;
```

is turned into:

```
state = STATE_1
do
{
        if ( state > MAGIC_VALUE_1 )
        {
                if ( state == STATE_2 )
                {
                        CODE_2;
                        state = STATE_3;
                }
                if ( state == STATE_1 )
                {
                        CODE_1;
                        state = STATE_2;
                }
        }
        else if ( state == JUNK_STATE_1 )
        {
                JUNK_CODE_1;
                state = STATE_4
        }
        else
        {
                if ( state == STATE_3 )
                {
                        JUNK_CODE_2;
                        state = JUNK_STATE_1;
                }
                if ( state == STATE_4 )
                {
                        CODE_3;
                        return;
                }
        }
} while(1);
```

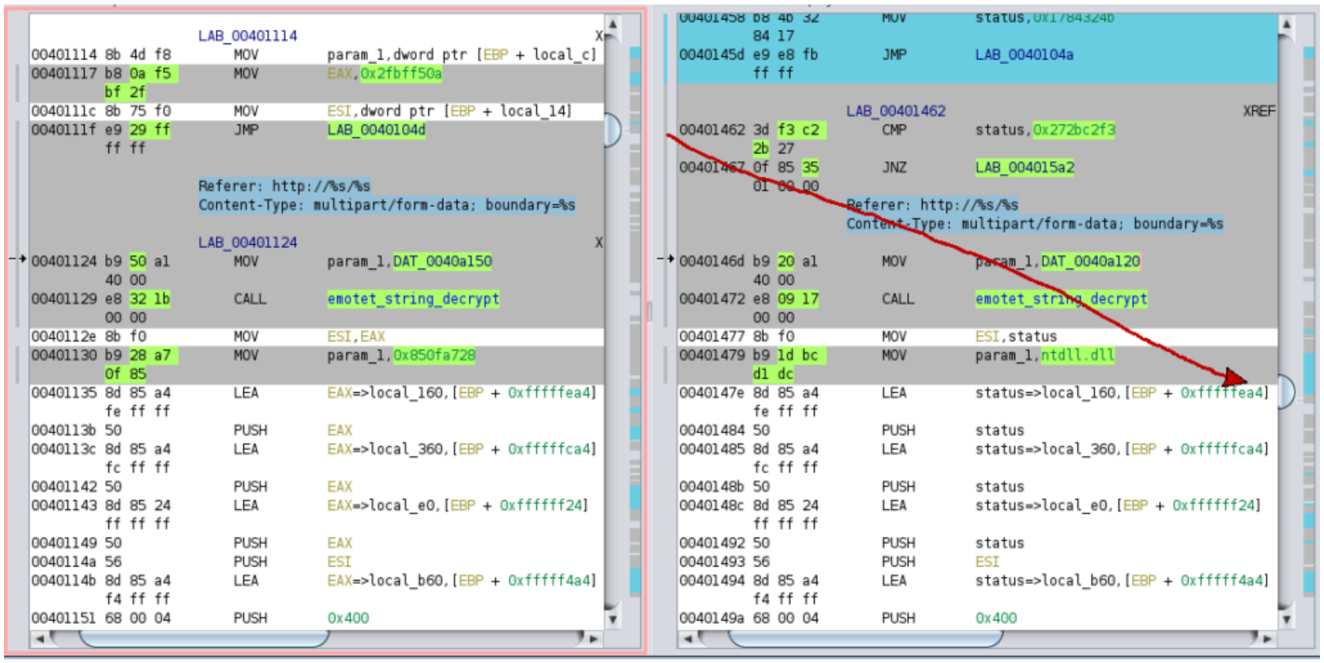
While semantically identical the second code is harder to follow.

In Emotet this looks like:



```

void __fastcall emotet_02_load_libraries(void)

{
  uint tmp_status;
  uint status;

  status = 0x3fcf7ea;
LAB_0040503b:
  do {
    tmp_status = status;
    if (tmp_status < 0x22f0f99d) {
      if (tmp_status == 0x22f0f99c) {
                    /* urlmon.dll */
        emotet_020_load_lib((uint *)&emostr_0xc89ebeb3L_urlmon_dll,4);
        status = 0x3738b43d;
        goto LAB_0040503b;
      }
      status = 0x203cce33;
      if (tmp_status == 0x3fcf7ea) goto LAB_0040503b;
      if (tmp_status == 0x5e45ce9) {
                    /* wtsapi32.dll */
        emotet_020_load_lib((uint *)&emostr_0x74f0daf9L_wtsapi32_dll,7);
        return;
      }
      if (tmp_status == 0xdcaf1cb) {
                    /* shell32.dll */
        emotet_020_load_lib((uint *)&emostr_0x1d902705L_shell32_dll,2);
        status = 0x3775a715;
        goto LAB_0040503b;
      }
      if (tmp_status == 0x203cce33) {
                    /* advapi32.dll */
        emotet_020_load_lib((uint *)&emostr_0x5a3dcc1cL_advapi32_dll,0);
        status = 0x266f36eb;
        goto LAB_0040503b;
      }
    }
    else {
      if (tmp_status == 0x266f36eb) {
                    /* crypt32.dll */
        emotet_020_load_lib((uint *)&emostr_0x36329fc3L_crypt32_dll,1);
        tmp_status = 0xdcaf1cb;
      }
      else {
```

Emotet uses this to move code blocks around, as can be seen in this example, where the code block setting up the headers of the HTTP C2 communication is near the beginning of the function in one binary and near the end of the function in another binary:



```
                           LAB_00401114                                  X
00401114 8b 4d f8          MOV           param_1,dword ptr [EBP + local_c]
00401117 b8 0a f5          MOV           EAX,0x2fbff50a
         bf 2f
0040111c 8b 75 f0          MOV           ESI,dword ptr [EBP + local_14]
0040111f e9 29 ff          JMP           LAB_0040104d
         ff ff

                           Referer: http://%s/%s
                           Content-Type: multipart/form-data; boundary=%s

                           LAB_00401124                                  X
00401124 b9 50 a1          MOV           param_1,DAT_0040a150
         40 00
00401129 e8 32 1b          CALL          emotet_string_decrypt
         00 00
0040112e 8b f0             MOV           ESI,EAX
00401130 b9 28 a7          MOV           param_1,0x850fa728
         0f 85
00401135 8d 85 a4          LEA           EAX=>local_160,[EBP + 0xfffffea4]
         fe ff ff
0040113b 50                PUSH          EAX
0040113c 8d 85 a4          LEA           EAX=>local_360,[EBP + 0xfffffca4]
         fc ff ff
00401142 50                PUSH          EAX
00401143 8d 85 24          LEA           EAX=>local_e0,[EBP + 0xffffff24]
         ff ff ff
00401149 50                PUSH          EAX
0040114a 56                PUSH          ESI
0040114b 8d 85 a4          LEA           EAX=>local_b60,[EBP + 0xfffff4a4]
         f4 ff ff
00401151 68 00 04          PUSH          0x400
```

```
00401458 b8 4b 32          MOV           status,0x1784324b
         84 17
0040145d e9 e8 fb          JMP           LAB_0040104a
         ff ff

                           LAB_00401462                               XREF
00401462 3d f3 c2          CMP           status,0x272bc2f3
         2b 27
00401467 0f 85 35          JNZ           LAB_004015a2
         01 00 00
                           Referer: http://%s/%s
                           Content-Type: multipart/form-data; boundary=%s

0040146d b9 20 a1          MOV           param_1,DAT_0040a120
         40 00
00401472 e8 09 17          CALL          emotet_string_decrypt
         00 00
00401477 8b f0             MOV           ESI,status
00401479 b9 1d bc          MOV           param_1,ntdll.dll
         d1 dc
0040147e 8d 85 a4          LEA           status=>local_160,[EBP + 0xfffffea4]
         fe ff ff
00401484 50                PUSH          status
00401485 8d 85 a4          LEA           status=>local_360,[EBP + 0xfffffca4]
         fc ff ff
0040148b 50                PUSH          status
0040148c 8d 85 24          LEA           status=>local_e0,[EBP + 0xffffff24]
         ff ff ff
00401492 50                PUSH          status
00401493 56                PUSH          ESI
00401494 8d 85 a4          LEA           status=>local_b60,[EBP + 0xfffff4a4]
         f4 ff ff
0040149a 68 00 04          PUSH          0x400
```

Junk Code

Another technique Emotet uses is called junk code. Here useless code that does not change the semantics of a program is added.

The code sequence:

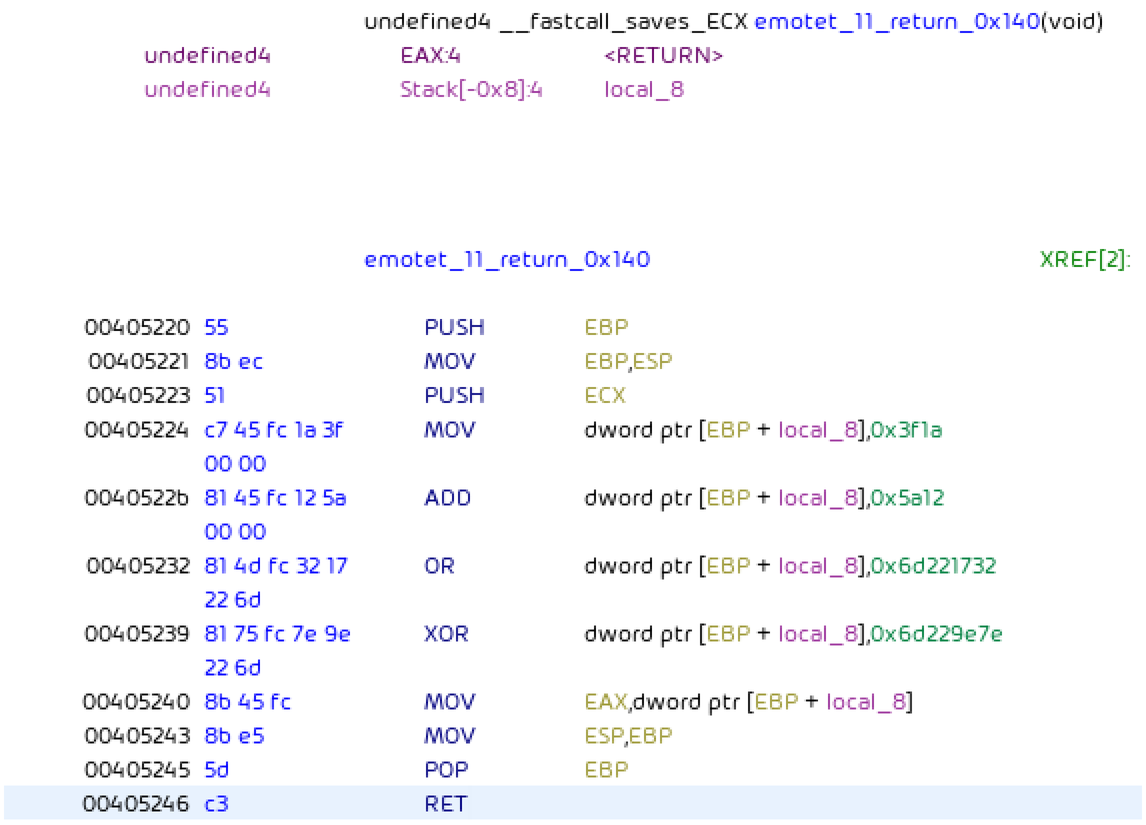
```
unsigned int function(unsigned int n)
{
    if (n == 0)
        return 1;
    return n * function(n - 1);
}
```

is turned into:

```
unsigned int function(unsigned int n)
{
    unsigned int a = 1234;
    unsigned int b = 4321;
    a = n + b;
    b = n + a;
    if (n == 0 && a > 10 && b > 20)
    {
        b = n - a;
        n = b;
        return 1;
    }
    a = b + 42;
    return n * function(n - 1);
}
```

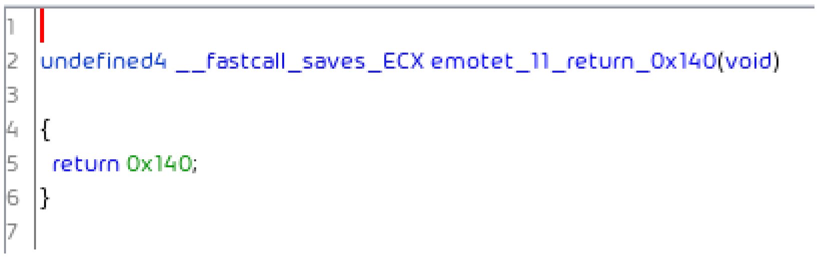
Here the calculations on the variables `a` and `b` are irrelevant code. They do not influence the result of the code at all. However, an analyst does not know which calculations are important and which are not. Hence, while semantically identical the second code is harder to understand.

Luckily modern analysis software is able to simplify artificially complicated code. So a complex looking function:



```
                        undefined4 __fastcall_saves_ECX emotet_11_return_0x140(void)
      undefined4          EAX:4           <RETURN>
      undefined4          Stack[-0x8]:4   local_8



                        emotet_11_return_0x140                               XREF[2]:

    00405220 55                PUSH      EBP
    00405221 8b ec             MOV       EBP,ESP
    00405223 51                PUSH      ECX
    00405224 c7 45 fc 1a 3f    MOV       dword ptr [EBP + local_8],0x3f1a
             00 00
    0040522b 81 45 fc 12 5a    ADD       dword ptr [EBP + local_8],0x5a12
             00 00
    00405232 81 4d fc 32 17    OR        dword ptr [EBP + local_8],0x6d221732
             22 6d
    00405239 81 75 fc 7e 9e    XOR       dword ptr [EBP + local_8],0x6d229e7e
             22 6d
    00405240 8b 45 fc          MOV       EAX,dword ptr [EBP + local_8]
    00405243 8b e5             MOV       ESP,EBP
    00405245 5d                POP       EBP
    00405246 c3                RET
```

is automatically reduced to a return of a static value:



```

undefined4 __fastcall_saves_ECX emotet_11_return_0x140(void)

{
  return 0x140;
}

```

This is used in the next obfuscation method.

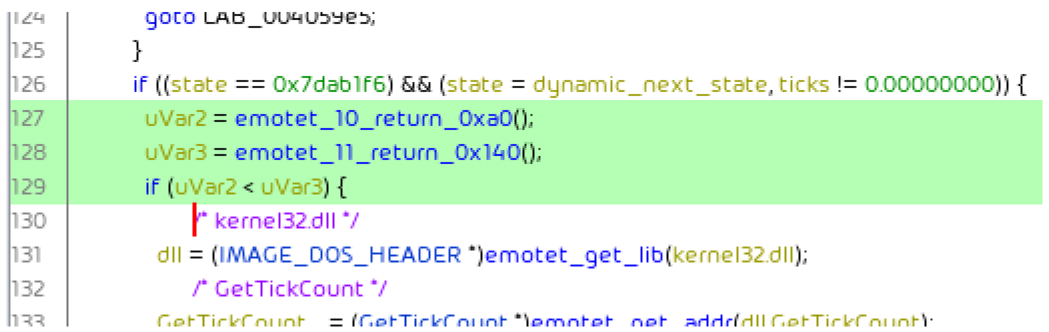
Opaque Predicates

Opaque predicates are branch conditions for which the outcome is already known, but which still need to be evaluated at runtime. An example would be a code that gets the current time twice. Because time never goes backwards the first value will not be greater than the second:

```
time_t a = time(NULL);
time_t b = time(NULL);
if ( a <= b )
    CODE;
else
    JUNK_CODE;
```

While to a human this is logical, a machine would still need to evaluate what values `a` and `b` have to determine whether to take the jump or not.

Emotet uses functions with a static return value – see previous junk code example above – and uses their return values as a branch condition:



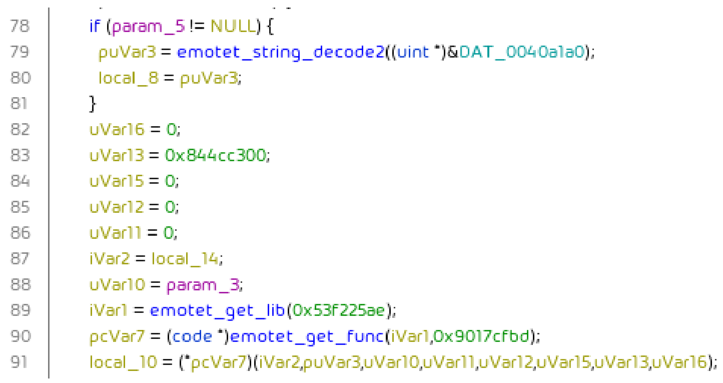
```
      goto LAB_004059e5;
    }
    if ((state == 0x7dab1f6) && (state = dynamic_next_state,ticks != 0.00000000)) {
      uVar2 = emotet_10_return_0xa0();
      uVar3 = emotet_11_return_0x140();
      if (uVar2 < uVar3) {
          /* kernel32.dll */
        dll = (IMAGE_DOS_HEADER *)emotet_get_lib(kernel32.dll);
          /* GetTickCount */
        GetTickCount_ = (GetTickCount *)emotet_get_addr(dll,GetTickCount);
```

The branches used for control flow flattening also use opaque predicates, as the `state` variable changes predictable but must be evaluated at runtime.

Dynamic Library and Function Resolution

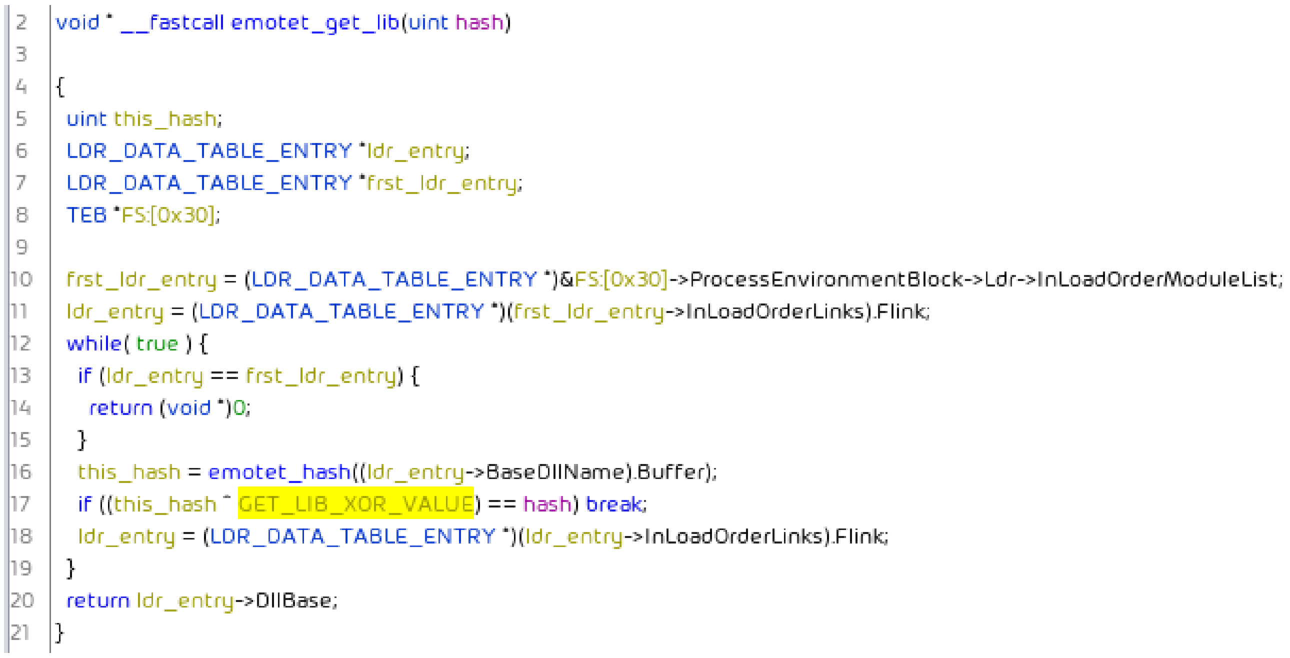
All library calls are dynamically resolved by hash. Previous versions stored the resolved functions. Now they are just-in-time resolved before **every** call.

For each library call, first the library is obtained (`emotet_get_lib()`). Then the address of the desired function is obtained (`emotet_get_func`):



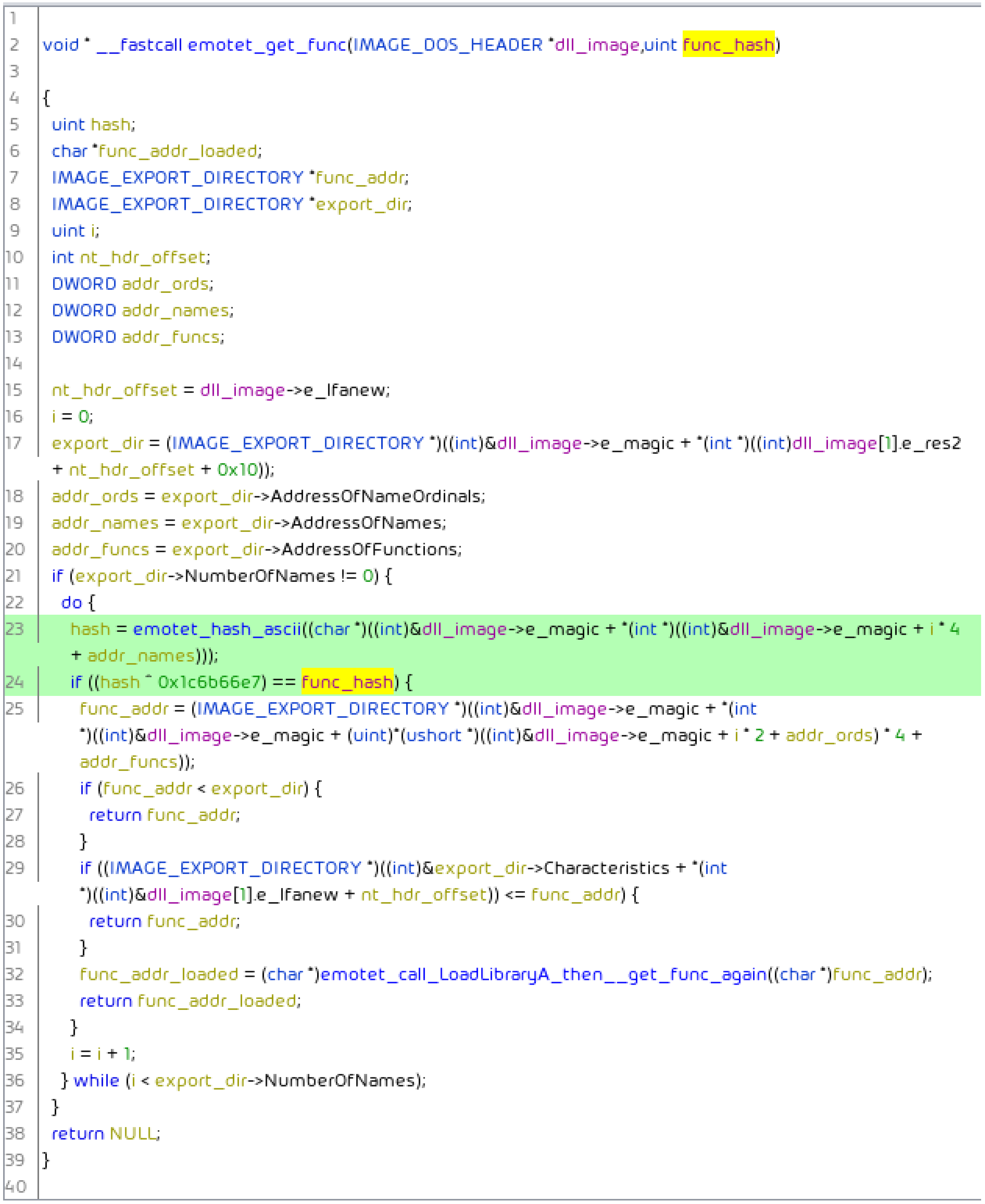
```
  if (param_5 != NULL) {
    puVar3 = emotet_string_decode2((uint *)&DAT_0040a1a0);
    local_8 = puVar3;
  }
  uVar16 = 0;
  uVar13 = 0x844cc300;
  uVar15 = 0;
  uVar12 = 0;
  uVar11 = 0;
  iVar2 = local_14;
  uVar10 = param_3;
  iVar1 = emotet_get_lib(0x53f225ae);
  pcVar7 = (code *)emotet_get_func(iVar1,0x9017cfbd);
  local_10 = (*pcVar7)(iVar2,puVar3,uVar10,uVar11,uVar12,uVar15,uVar13,uVar16);
```

Libraries are resolved via the `InLoadOrderModuleList` reachable from the Process Environment Block (PEB) (`FS:[0x30]`):


```
void * __fastcall emotet_get_lib(uint hash)

{
  uint this_hash;
  LDR_DATA_TABLE_ENTRY *ldr_entry;
  LDR_DATA_TABLE_ENTRY *frst_ldr_entry;
  TEB *FS:[0x30];

  frst_ldr_entry = (LDR_DATA_TABLE_ENTRY *)&FS:[0x30]->ProcessEnvironmentBlock->Ldr->InLoadOrderModuleList;
  ldr_entry = (LDR_DATA_TABLE_ENTRY *)(frst_ldr_entry->InLoadOrderLinks).Flink;
  while( true ) {
    if (ldr_entry == frst_ldr_entry) {
      return (void *)0;
    }
    this_hash = emotet_hash((ldr_entry->BaseDllName).Buffer);
    if ((this_hash ^ GET_LIB_XOR_VALUE) == hash) break;
    ldr_entry = (LDR_DATA_TABLE_ENTRY *)(ldr_entry->InLoadOrderLinks).Flink;
  }
  return ldr_entry->DllBase;
}
```

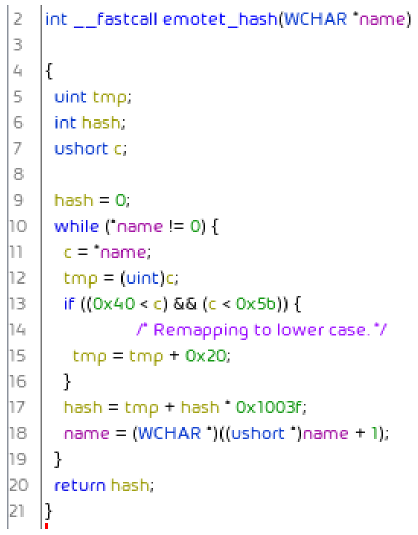
Then the `DllBaseName` for every loaded module is hashed and compared against the current queried hash. If the hash matches the `DllBase` is returned. The `GET_LIB_XOR_VALUE` varies for each binary. This means that library hashes change from sample to sample and can not be pre-calculated. They must be calculated for each sample individually.

Addresses of functions are obtained via manually traversal of in this case the export directory data structure. Each exported function name of the previously resolved DLL's image is iterated over and hashed. If the hash matches the function's address is returned:



```

void * __fastcall emotet_get_func(IMAGE_DOS_HEADER *dll_image,uint func_hash)

{
  uint hash;
  char *func_addr_loaded;
  IMAGE_EXPORT_DIRECTORY *func_addr;
  IMAGE_EXPORT_DIRECTORY *export_dir;
  uint i;
  int nt_hdr_offset;
  DWORD addr_ords;
  DWORD addr_names;
  DWORD addr_funcs;

  nt_hdr_offset = dll_image->e_lfanew;
  i = 0;
  export_dir = (IMAGE_EXPORT_DIRECTORY *)((int)&dll_image->e_magic + *(int *)((int)dll_image[1].e_res2 + nt_hdr_offset + 0x10));
  addr_ords = export_dir->AddressOfNameOrdinals;
  addr_names = export_dir->AddressOfNames;
  addr_funcs = export_dir->AddressOfFunctions;
  if (export_dir->NumberOfNames != 0) {
    do {
      hash = emotet_hash_ascii((char *)((int)&dll_image->e_magic + *(int *)((int)&dll_image->e_magic + i * 4 + addr_names)));
      if ((hash ^ 0x1c6b66e7) == func_hash) {
        func_addr = (IMAGE_EXPORT_DIRECTORY *)((int)&dll_image->e_magic + *(int *)((int)&dll_image->e_magic + (uint)*(ushort *)((int)&dll_image->e_magic + i * 2 + addr_ords) * 4 + addr_funcs));
        if (func_addr < export_dir) {
          return func_addr;
        }
        if ((IMAGE_EXPORT_DIRECTORY *)((int)&export_dir->Characteristics + *(int *)((int)&dll_image[1].e_lfanew + nt_hdr_offset)) <= func_addr) {
          return func_addr;
        }
        func_addr_loaded = (char *)emotet_call_LoadLibraryA_then__get_func_again((char *)func_addr);
        return func_addr_loaded;
      }
      i = i + 1;
    } while (i < export_dir->NumberOfNames);
  }
  return NULL;
}

```

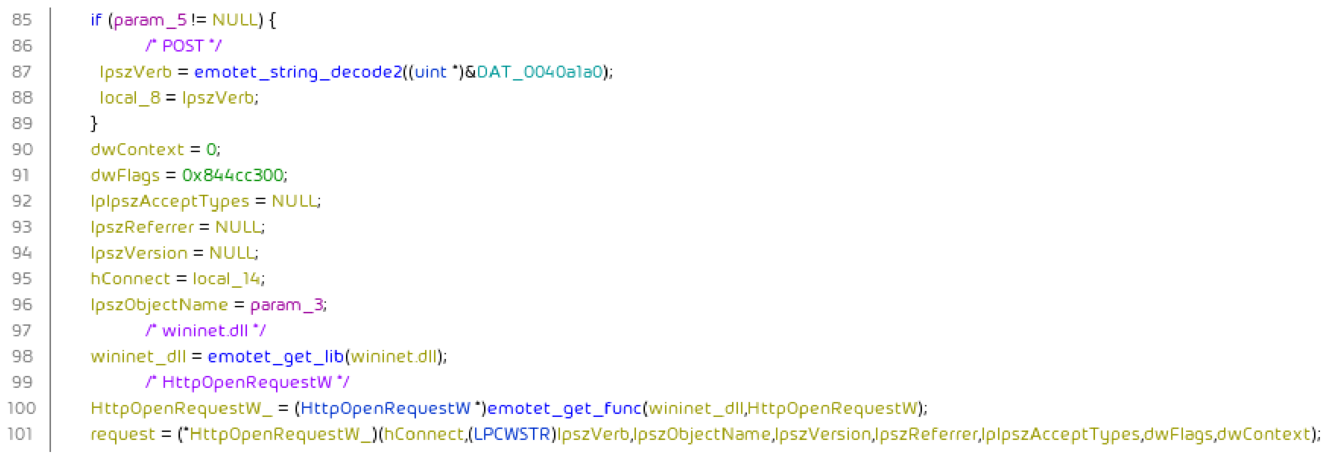
Emotet still uses the same hashing algorithm as previous versions:



```
int __fastcall emotet_hash(WCHAR *name)

{
  uint tmp;
  int hash;
  ushort c;

  hash = 0;
  while (*name != 0) {
    c = *name;
    tmp = (uint)c;
    if ((0x40 < c) && (c < 0x5b)) {
                    /* Remapping to lower case. */
      tmp = tmp + 0x20;
    }
    hash = tmp + hash * 0x1003f;
    name = (WCHAR *)((ushort *)name + 1);
  }
  return hash;
}
```

The `emotet_get_func()` function uses a variation without mapping the uppercase characters to lowercase, i.e., its hash is case-sensitive, and ingesting a `char` string instead of a `wchar` string. The `emotet_hash()` function is also heavily laced with junk code, which luckily the decompiler already simplified and/or discarded.

Using the Ghidra analysis scripts `emotet_lib_imports.py` and `emotet_func_imports.py` the called library and function names can be reconstructed from their hashes:



```
    if (param_5 != NULL) {
                    /* POST */
      lpszVerb = emotet_string_decode2((uint *)&DAT_0040a1a0);
      local_8 = lpszVerb;
    }
    dwContext = 0;
    dwFlags = 0x844cc300;
    lplpszAcceptTypes = NULL;
    lpszReferrer = NULL;
    lpszVersion = NULL;
    hConnect = local_14;
    lpszObjectName = param_3;
                    /* wininet.dll */
    wininet_dll = emotet_get_lib(wininet.dll);
                    /* HttpOpenRequestW */
    HttpOpenRequestW_ = (HttpOpenRequestW *)emotet_get_func(wininet_dll,HttpOpenRequestW);
    request = (*HttpOpenRequestW_)(hConnect,(LPCWSTR)lpszVerb,lpszObjectName,lpszVersion,lpszReferrer,lplpszAcceptTypes,dwFlags,dwContext);
```

The following libraries that are usually not loaded into a process by default are loaded via `LoadLibraryW`:

```
shell32.dll
userenv.dll
urlmon.dll
wininet.dll
wtsapi32.dll
advapi32.dll
crypt32.dll
shlwapi.dll
```

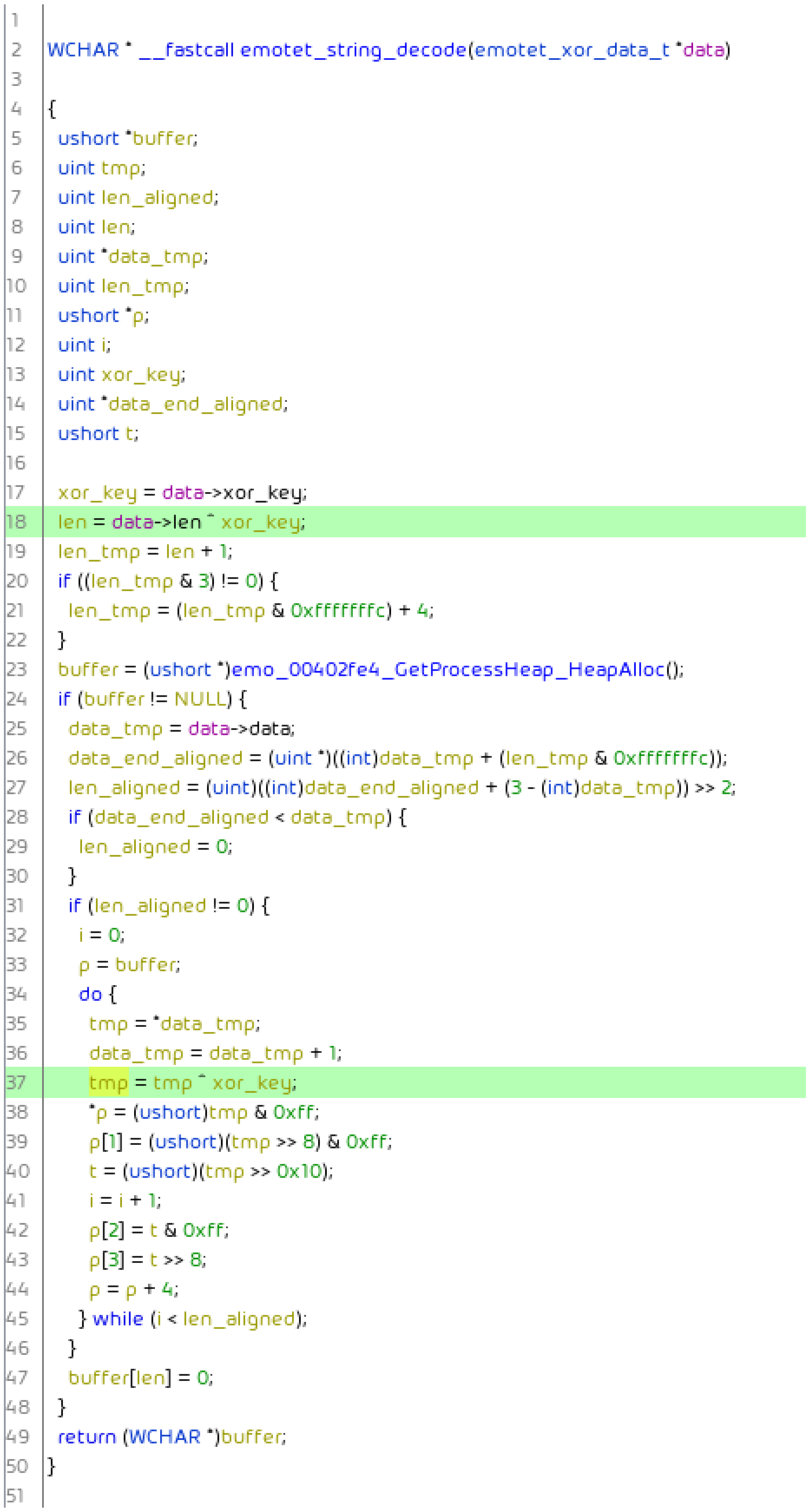
Handles to the libraries are stored to allocated memory. But never used. We have found other functions and code paths that are never used. These is likely leftover code that was not removed during code updates.

XOR Obfuscation

Strings and the RSA key are (as in previous versions) XOR obfuscated. The use the following structure:

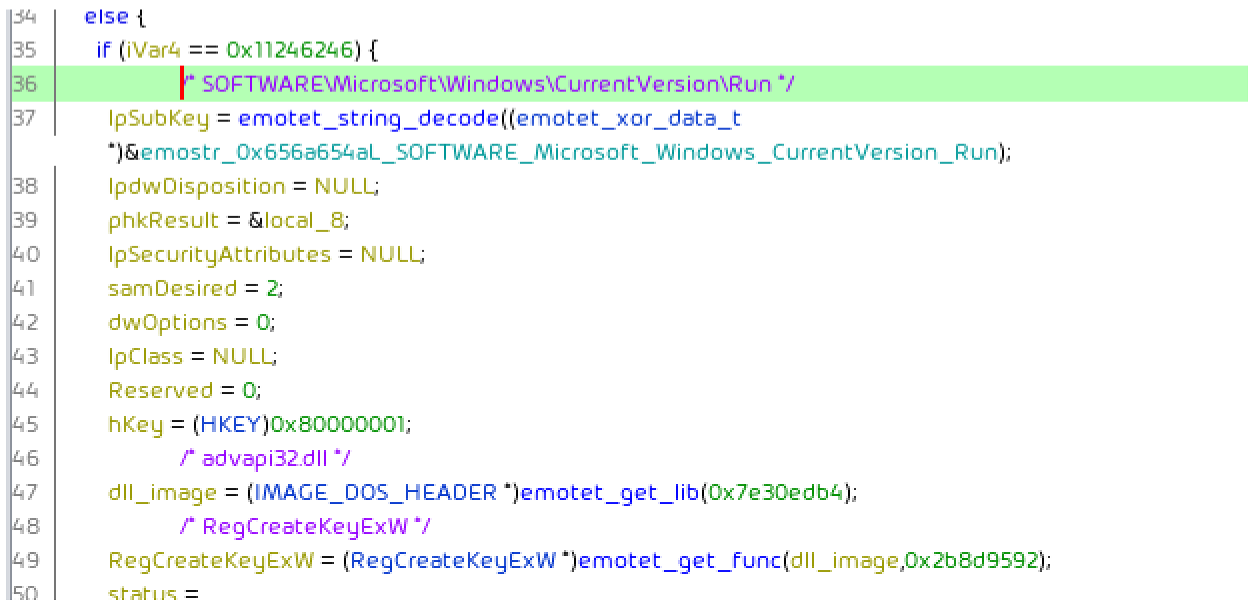
```
typedef emotet_xor_data_t {
    uint32_t xor_key;
    uint32_t len;
    uint32_t data[];
} emotet_xor_data_t;
```

They are decrypted by first XOR'ing `len` with `xor_key`. Then XOR'ing decoded and to 4-byte boundaries aligned `len` bytes of `data` with `xor_key`:



```

WCHAR * __fastcall emotet_string_decode(emotet_xor_data_t *data)

{
  ushort *buffer;
  uint tmp;
  uint len_aligned;
  uint len;
  uint *data_tmp;
  uint len_tmp;
  ushort *p;
  uint i;
  uint xor_key;
  uint *data_end_aligned;
  ushort t;

  xor_key = data->xor_key;
  len = data->len ^ xor_key;
  len_tmp = len + 1;
  if ((len_tmp & 3) != 0) {
    len_tmp = (len_tmp & 0xfffffffc) + 4;
  }
  buffer = (ushort *)emo_00402fe4_GetProcessHeap_HeapAlloc();
  if (buffer != NULL) {
    data_tmp = data->data;
    data_end_aligned = (uint *)((int)data_tmp + (len_tmp & 0xfffffffc));
    len_aligned = (uint)((int)data_end_aligned + (3 - (int)data_tmp)) >> 2;
    if (data_end_aligned < data_tmp) {
      len_aligned = 0;
    }
    if (len_aligned != 0) {
      i = 0;
      p = buffer;
      do {
        tmp = *data_tmp;
        data_tmp = data_tmp + 1;
        tmp = tmp ^ xor_key;
        *p = (ushort)tmp & 0xff;
        p[1] = (ushort)(tmp >> 8) & 0xff;
        t = (ushort)(tmp >> 0x10);
        i = i + 1;
        p[2] = t & 0xff;
        p[3] = t >> 8;
        p = p + 4;
      } while (i < len_aligned);
    }
    buffer[len] = 0;
  }
  return (WCHAR *)buffer;
}

```

The XOR key again changes from binary to binary. But still the decoding process can be automated, e.g. the `emotet_string_decode.py` script can decode used strings:



```
else {
 if (iVar4 == 0x11246246) {
            /* SOFTWARE\Microsoft\Windows\CurrentVersion\Run */
  lpSubKey = emotet_string_decode((emotet_xor_data_t
  *)&emostr_0x656a654aL_SOFTWARE_Microsoft_Windows_CurrentVersion_Run);
  lpdwDisposition = NULL;
  phkResult = &local_8;
  lpSecurityAttributes = NULL;
  samDesired = 2;
  dwOptions = 0;
  lpClass = NULL;
  Reserved = 0;
  hKey = (HKEY)0x80000001;
            /* advapi32.dll */
  dll_image = (IMAGE_DOS_HEADER *)emotet_get_lib(0x7e30edb4);
            /* RegCreateKeyExW */
  RegCreateKeyExW = (RegCreateKeyExW *)emotet_get_func(dll_image,0x2b8d9592);
  status =
```

They are also just-in-time decrypted on demand to a new memory allocation. The memory allocation holding the decrypted string is deleted again after use for every use of the string.

Static Analysis Tutorial

With the basics of the new obfuscation techniques covered we quickly outline how Emotet can still be analyzed.

1. Unpack your Emotet sample. E.g., using the free open source community developed CAPE sandbox [CAPE].
2. Import into Ghidra [GHIDRA].
3. Run Auto Analysis.
4. Run `emotet_lib_imports.py` with `currentAddress` in 2nd function called in `entry` function (this is what we refer to as the `emotet_get_lib` function).
5. Run `emotet_func_imports.py` (selecting `func_names.txt`) with `currentAddress` in 3rd function called in `entry` function (this is what we refer to as the `emotet_get_func` function).
6. Run `emotet_string_decode.py` with `currentAddress` in `emo_*_LoadLibraryW` function.
7. Run `emotet_string_decode.py` with `currentAddress` in 1st function called in `emo_*_LoadLibraryW` function.

Then you have:

- Comments for library and function resolution.
- Two enum types `emotet_{lib,func}_hash` with enums for the library and function hashes, which you can optionally apply to the `emotet_get_{lib,func}` functions.
- Emotet strings decrypted and set as comments, labels as well as searchable bookmarks.

C2 communication is found by searching for usage of the `HttpSendRequestW` function. The type of request and request headers can be found by backtracking via the first parameter (`hRequest`) to `HttpSendRequestW`. Alternatively usage of the `HttpOpenRequestW` function, the `InternetConnect` function, the `POST` string, or the `Referer: http://%s/%s ...` string will yield the relevant code.

C2 IP storage can be found by searching for the `%u.%u.%u.%u` string. These octets are filled from data in a allocation we named `emotet_c2_data`. This allocation is filled from a data location we named `emotet_c2_list`. If you run `emotet_ip_decode.py` with `currentAddress` set to the address of `emotet_c2_list` the C2 list is decoded and annotated as a comment.

The C2 HTTP request template looks like:

```
POST
Referer: http://%s/%s
Content-Type: multipart/form-data; boundary=%s

--%S
Content-Disposition: form-data; name="%s"; filename="%s"
Content-Type: application/octet-stream
```

RSA key is found by searching the source of the 3rd parameter (`pbEncoded`) to the `CryptDecodeObjectEx` function. It is XOR encoded like the strings. You can run `emotet_data_decode.py` with `currentAddress` on the data location passed to the function that returns the pointer that is then passed as the 3rd parameter to the `CryptDecodeObjectEx` function. You can use `openssl asn1parse -inform DER -in emotet_key.bin` to check if the decoded `emotet_key_bin` is valid. The key must be the exact length without any appended bytes beyond the ASN1 DER sequence stream.

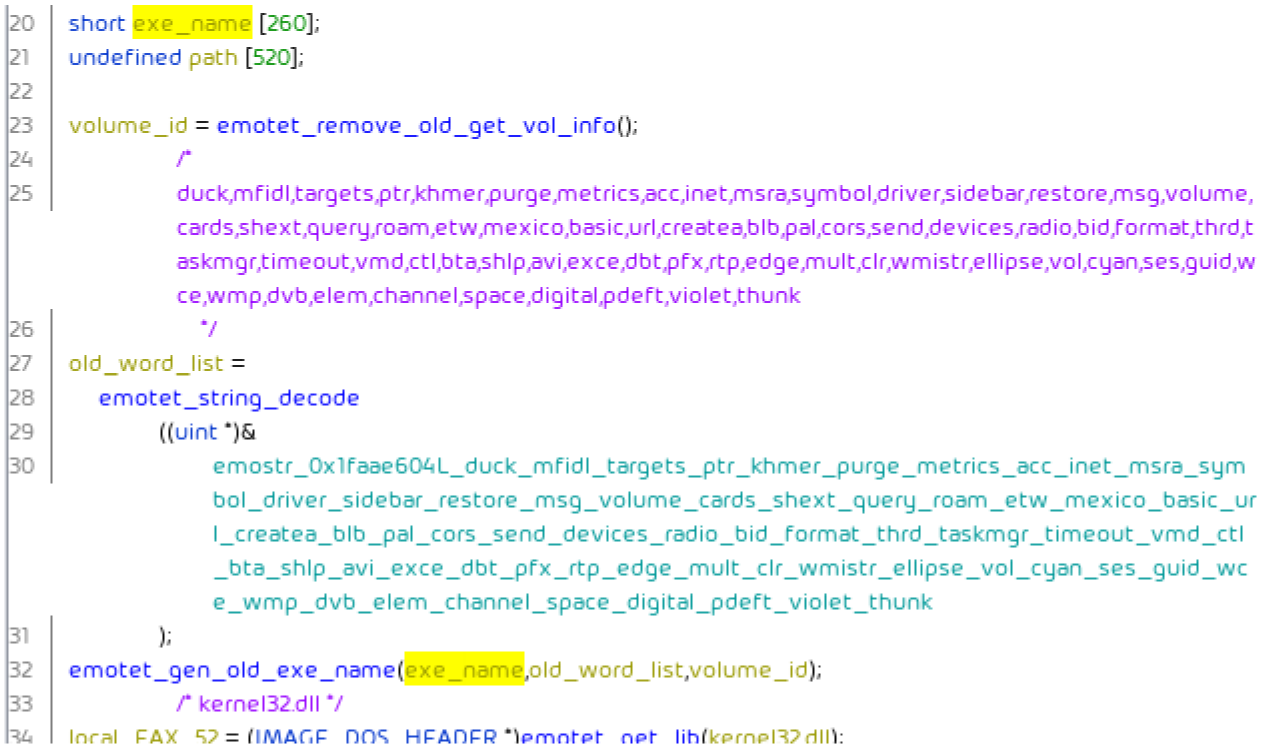
Search for functions using the `OpenSCManagerW` function. One function will contain an assignment to a data location. Emotet uses `OpenSCManagerW` to check whether it has admin rights or not. This is where what we call the `is_admin` flag is set. Use “Auto Create Structure” on the data reference and name it `emotet_data` and name the field assigned the value `1` `is_admin`.

Search for functions using the string `%s\%s.exe`. This is where the executable path is constructed. In its vicinity the `emotet_data` data location is accessed. Determine by their usage as parameters to the `_snwprintf` function which of the two fields in the `emotet_data` structure is the `exe_path` and which is the `exe_name`.

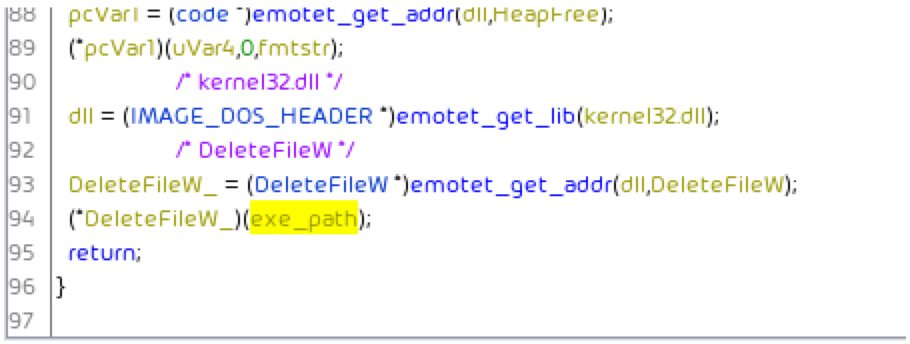
Then use “References -> Find use of emotet_data_t.exe_name” to find out how the `exe_name` is generated. Do the same for the `exe_path`. The `exe_name` will also be used to generate the service name. Relevant code is found by searching for `CreateServiceW`. Persistence via run keys is found via `RegCreateKeyExW` and/or the `SOFTWARE\Microsoft\Windows\CurrentVersion\Run` string.

Searching for usage of the `%s:Zone.Identifier` string finds where Emotet deletes its Zone.Identifier ADS, which on Windows is used to mark files downloaded from external sites as potentially unsafe.

As an interesting side note older new versions contained code removing old Emotet binaries which names were generated from word lists:

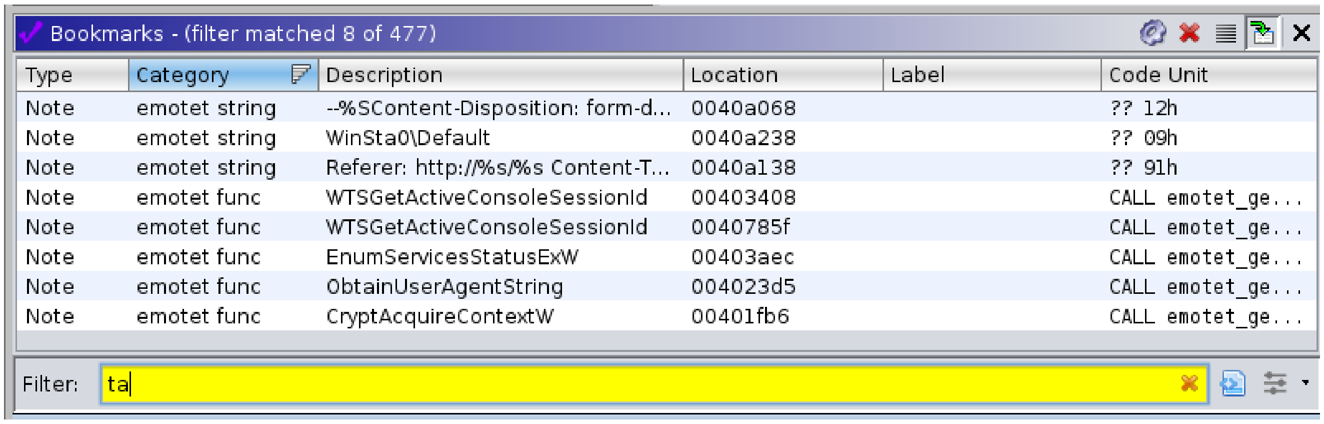


```
short exe_name [260];
undefined path [520];

volume_id = emotet_remove_old_get_vol_info();
            /*
            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,t
            askmgr,timeout,vmd,ctl,bta,shlp,avi,exce,dbt,pfx,rtp,edge,mult,clr,wmistr,ellipse,vol,cyan,ses,guid,w
            ce,wmp,dvb,elem,channel,space,digital,pdeft,violet,thunk
              */
old_word_list =
    emotet_string_decode
            ((uint *)&
                 emostr_0x1faae604L_duck_mfidl_targets_ptr_khmer_purge_metrics_acc_inet_msra_sym
                 bol_driver_sidebar_restore_msg_volume_cards_shext_query_roam_etw_mexico_basic_ur
                 l_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_wc
                 e_wmp_dvb_elem_channel_space_digital_pdeft_violet_thunk
            );
emotet_gen_old_exe_name(exe_name,old_word_list,volume_id);
            /* kernel32.dll */
local_EAX_52 = (IMAGE_DOS_HEADER *)emotet_get_lib(kernel32.dll);
```



```
pcVar1 = (code *)emotet_get_addr(dll,HeapFree);
(*pcVar1)(uVar4,0,fmtstr);
            /* kernel32.dll */
dll = (IMAGE_DOS_HEADER *)emotet_get_lib(kernel32.dll);
            /* DeleteFileW */
DeleteFileW_ = (DeleteFileW *)emotet_get_addr(dll,DeleteFileW);
(*DeleteFileW_)(exe_path);
return;
}
```

This functionality has been removed from the latest version, indicating that the migration to the new version has been completed for Epoch 2 and there is no need to delete old binaries anymore:



Bookmarks - (filter matched 8 of 477)

Type	Category	Description	Location	Label	Code Unit
Note	emotet string	--%SContent-Disposition: form-d...	0040a068		?? 12h
Note	emotet string	WinSta0\Default	0040a238		?? 09h
Note	emotet string	Referer: http://%s/%s Content-T...	0040a138		?? 91h
Note	emotet func	WTSGetActiveConsoleSessionId	00403408		CALL emotet_ge...
Note	emotet func	WTSGetActiveConsoleSessionId	0040785f		CALL emotet_ge...
Note	emotet func	EnumServicesStatusExW	00403aec		CALL emotet_ge...
Note	emotet func	ObtainUserAgentString	004023d5		CALL emotet_ge...
Note	emotet func	CryptAcquireContextW	00401fb6		CALL emotet_ge...

Filter: ta

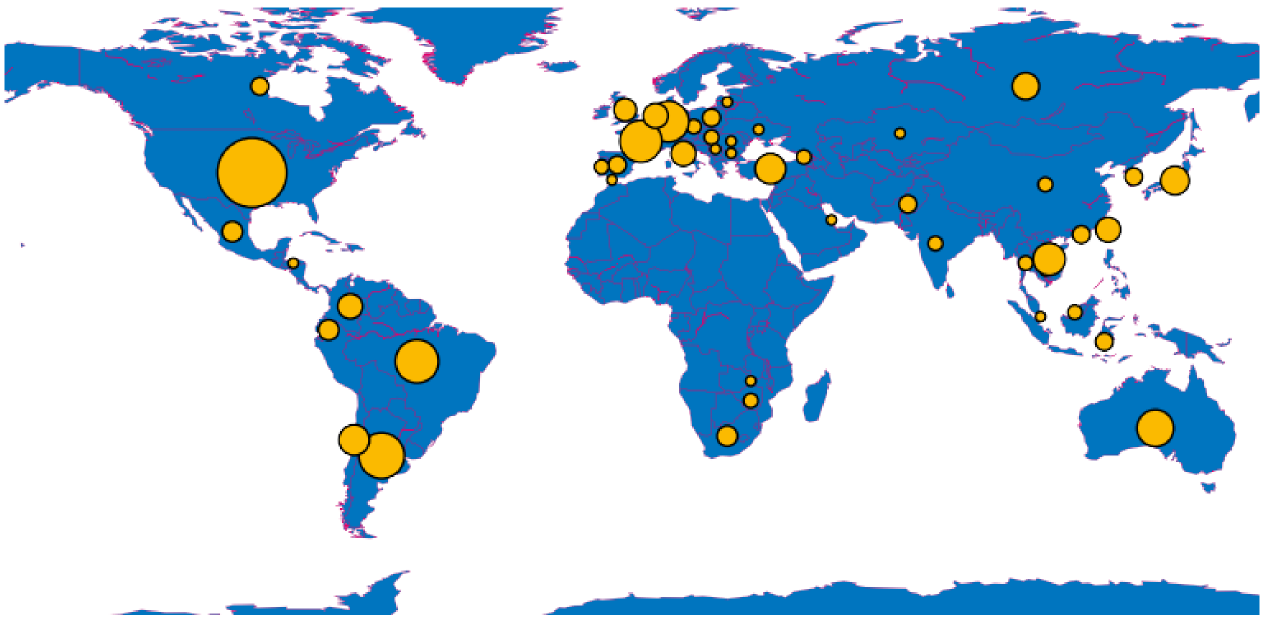
(Either that or the function has moved and has not been picked up by our string deobfuscator script.)

The following video demonstrates how our scripts can be used to jump start your own analysis of the Emotet loader:

Download our scripts on Github.

Tier 1 C2 geolocation

Emotet's tier 1 C2 proxies and servers are geolocated all over the world:



(C2 IP list as observed 2020-04-20.)

Conclusion and Remediation

To protect against Emotet the US CERT recommends to “implement filters at the email gateway to filter out emails with known malspam indicators” [USCERT].

Hornetsecurity’s Email Spam Filtering with the highest detection rates on the market are not impacted by the updates to the Emotet loader (as the loader is never send directly via emails) and thus will (as in the past) block all Emotet malspam indicators, such as macro documents used for infection, but also known Emotet download URLs. Hornetsecurity’s Advanced Threat Protection extends this protection by also detecting yet unknown malicious links by dynamically downloaded and executing the potentially malicious content in a monitored and sandboxed environment. Meaning that even in the event the Emotet loader changes is accompanied in a change in delivery tactics, Hornetsecurity is prepared.

Beyond blocking the incoming Emotet emails defenders can use public available information by the Cryptolaemus team, a voluntary group of IT security people banding together to fight Emotet. They provide new information daily via their website [CryptolaemusWeb]. There you can obtain the latest C2 IP list for finding and/or blocking C2 traffic. For real-time updates you can follow their Twitter account [CryptolaemusTwitter].

We acknowledge that our presented analysis only scratches the surface of the Emotet malware complex, but when Emotet returns so will we, with updated analyses of new malicious documents and/or any other new developments.

References

- [USCERT] https://www.us-cert.gov/ncas/alerts/TA18-201A
- [JRoosen] https://twitter.com/JRoosen/status/1228215329022603267
- [CERTPL] https://www.cert.pl/en/news/single/whats-up-emotet/
- [CAPE] https://capesandbox.com/
- [GHIDRA] https://ghidra-sre.org/
- [Cryptolaemus] https://paste.cryptolaemus.com/
- [CryptolaemusTwitter] https://twitter.com/Cryptolaemus1

Samples Used

Hashes

SHA256	Description
`cc96711da9ef7b63d5f1749d8866b0149f84506f9d53d79de018dac92e9443a0`	Epoch 1 sample (“old” version) 2020-04-20

SHA256	Description
`4c3acc885006faebe59e0bfdd452499056d8fcc9e6a810d7ff93762ce1a061ad`	Epoch 1 sample unpacked payload 2020-04-20
`4250a3ab9044b4a3a5f8319e712306bb06df61b1dee8b608505c026bacba4aa1`	Epoch 2 sample (“new” version) 2020-04-20
`14f8463a86bbae338925fbbcb709953ae7f1bffdb065fee540b6fa88fbbce70e`	Epoch 2 sample unpacked payload 2020-04-20