The Re-Emergence of Emotet

deepinstinct.com/blog/the-re-emergence-of-emotet

November 30, 2021

Learn more

November 30, 2021 | Ron Ben Yizhak

Emotet, the malware botnet, has resurfaced after almost 10 months. The operation was originally taken down by multiple international law enforcement agencies this past January. These agencies took control of the infrastructure and scheduled an un-installation of the malware on April 25.

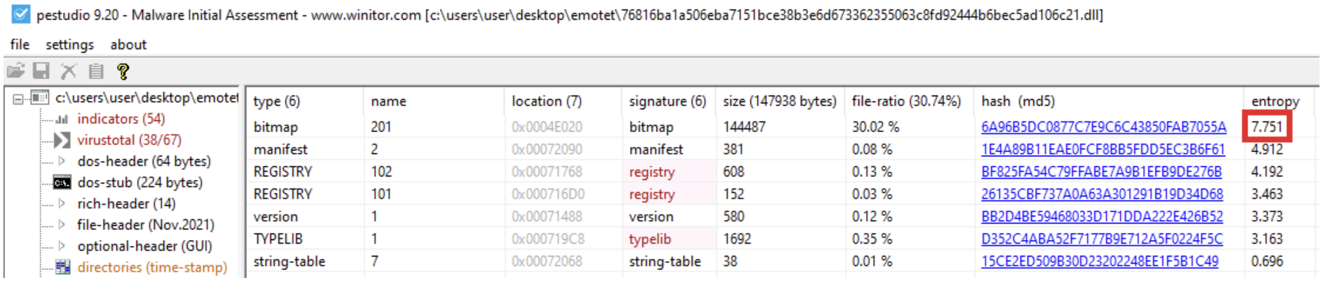
So what does the re-emergence of Emotet mean, and how can cyber professionals prepare for new threats? This blog will analyze the new DLL, break down a new unpacking technique and new features, and review similarities in the new variant with the previous version.

We will also explore a novel tool called “DeMotet” that automates the analysis of Emotet samples on a large scale and includes an unpacker for the latest loader and decryption scripts for the payload. It is used to detect any modification in the malware, and it is now publicly available.

Static Analysis

For this analysis, the following sample will be used:
76816ba1a506eba7151bce38b3e6d673362355063c8fd92444b6bec5ad106c21

As shown in our previous blog posts, the execution flow of Emotet consists of multiple stages that are unpacked in succession. As expected, this is still the case with the new variant. The DLL that is written to disk isn’t the actual payload that communicates with the C2 servers. This can be seen by a review of the static information of the resources.



pestudio 9.20 - Malware Initial Assessment - www.winitor.com [c:\users\user\desktop\emotet\76816ba1a506eba7151bce38b3e6d673362355063c8fd92444b6bec5ad106c21.dll]

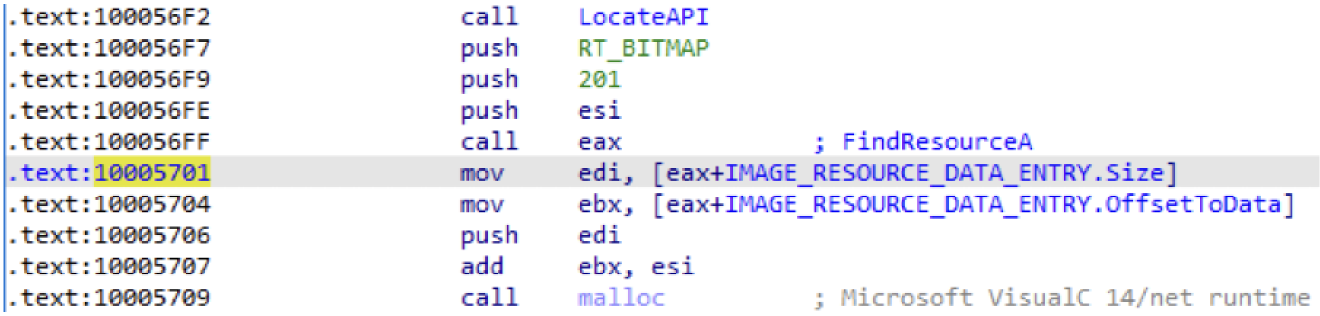
type (6)	name	location (7)	signature (6)	size (147938 bytes)	file-ratio (30.74%)	hash (md5)	entropy
bitmap	201	0x0004E020	bitmap	144487	30.02 %	6A96B5DC0877C7E9C6C43850FAB7055A	7.751
manifest	2	0x00072090	manifest	381	0.08 %	1E4A89B11EAE0FCF8BB5FDD5EC3B6F61	4.912
REGISTRY	102	0x00071768	registry	608	0.13 %	BF825FA54C79FFABE7A9B1EFB9DE276B	4.192
REGISTRY	101	0x000716D0	registry	152	0.03 %	26135CBF737A0A63A301291B19D34D68	3.463
version	1	0x00071488	version	580	0.12 %	BB2D4BE59468033D171DDA222E426B52	3.373
TYPELIB	1	0x000719C8	typelib	1692	0.35 %	D352C4ABA52F7177B9E712A5F0224F5C	3.163
string-table	7	0x00072068	string-table	38	0.01 %	15CE2ED509B30D23202248EE1F5B1C49	0.696

The entropy of the bitmap resource is high, and it most likely contains encrypted information. To unpack the next stage, the decryption routine needs to be found.

This resource will be accessed before decryption starts. A breakpoint can be set on the “FindResourceA” function to reach this point.

Extracting the Next Stage

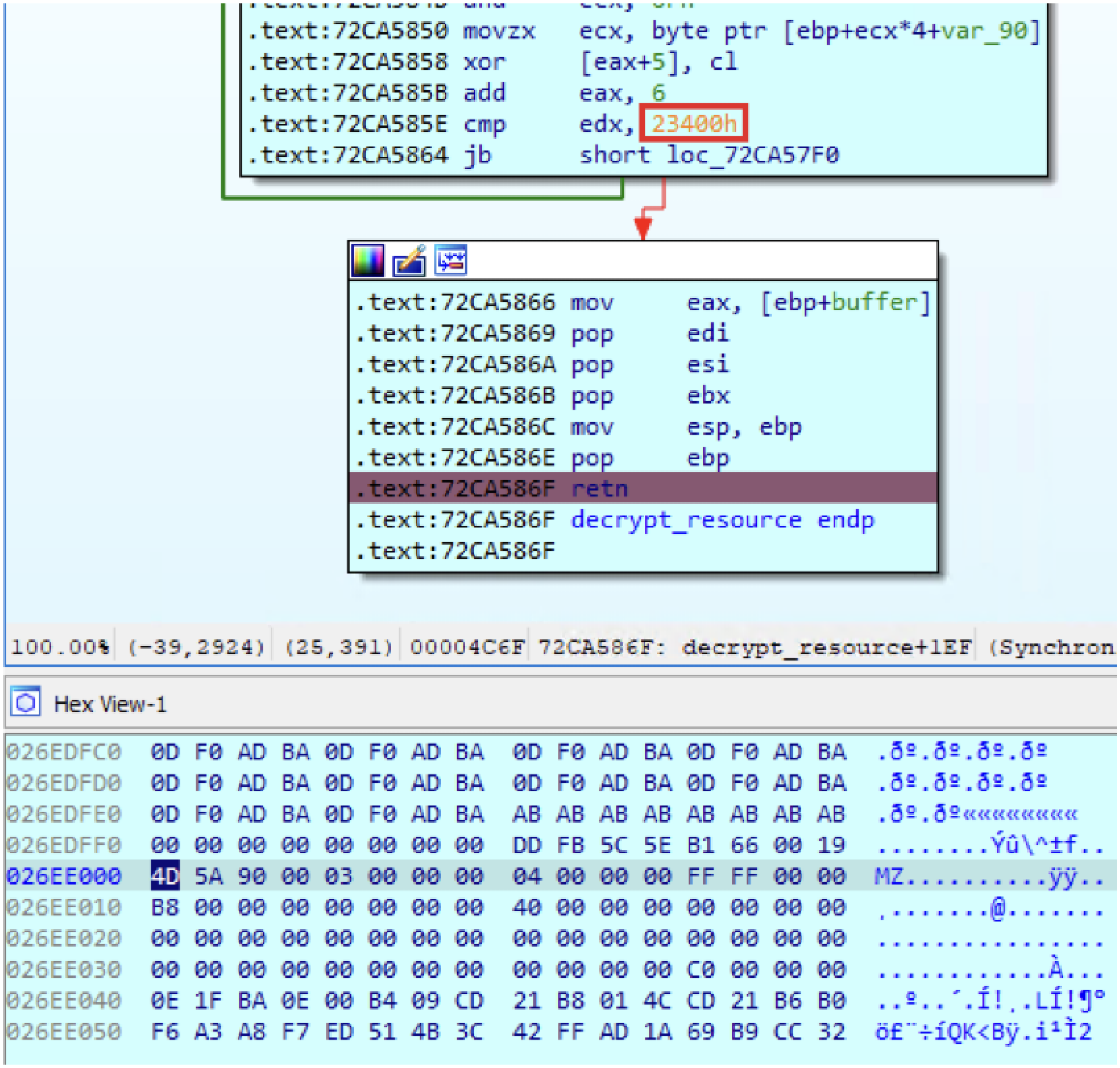
The malware reaches the breakpoint and then returns to the address 0x10005701.



```
.text:100056F2                 call    LocateAPI
.text:100056F7                 push    RT_BITMAP
.text:100056F9                 push    201
.text:100056FE                 push    esi
.text:100056FF                 call    eax             ; FindResourceA
.text:10005701                 mov     edi, [eax+IMAGE_RESOURCE_DATA_ENTRY.Size]
.text:10005704                 mov     ebx, [eax+IMAGE_RESOURCE_DATA_ENTRY.OffsetToData]
.text:10005706                 push    edi
.text:10005707                 add     ebx, esi
.text:10005709                 call    malloc          ; Microsoft VisualC 14/net runtime
```

The parameters for “FindResourceA” match the suspicious resource. This API is called through a register because it isn’t imported. The address is located in runtime to hinder static analysis. The function then allocates memory based on the size of the resource and goes through some decryption loops.

The return value is the next stage. The size of the file is specified in the code, which makes the extraction from memory even easier.



```
.text:72CA5850 movzx   ecx, byte ptr [ebp+ecx*4+var_90]
.text:72CA5858 xor     [eax+5], cl
.text:72CA585B add     eax, 6
.text:72CA585E cmp     edx, 23400h
.text:72CA5864 jb      short loc_72CA57F0
```

```
.text:72CA5866 mov     eax, [ebp+buffer]
.text:72CA5869 pop     edi
.text:72CA586A pop     esi
.text:72CA586B pop     ebx
.text:72CA586C mov     esp, ebp
.text:72CA586E pop     ebp
.text:72CA586F retn
.text:72CA586F decrypt_resource endp
.text:72CA586F
```

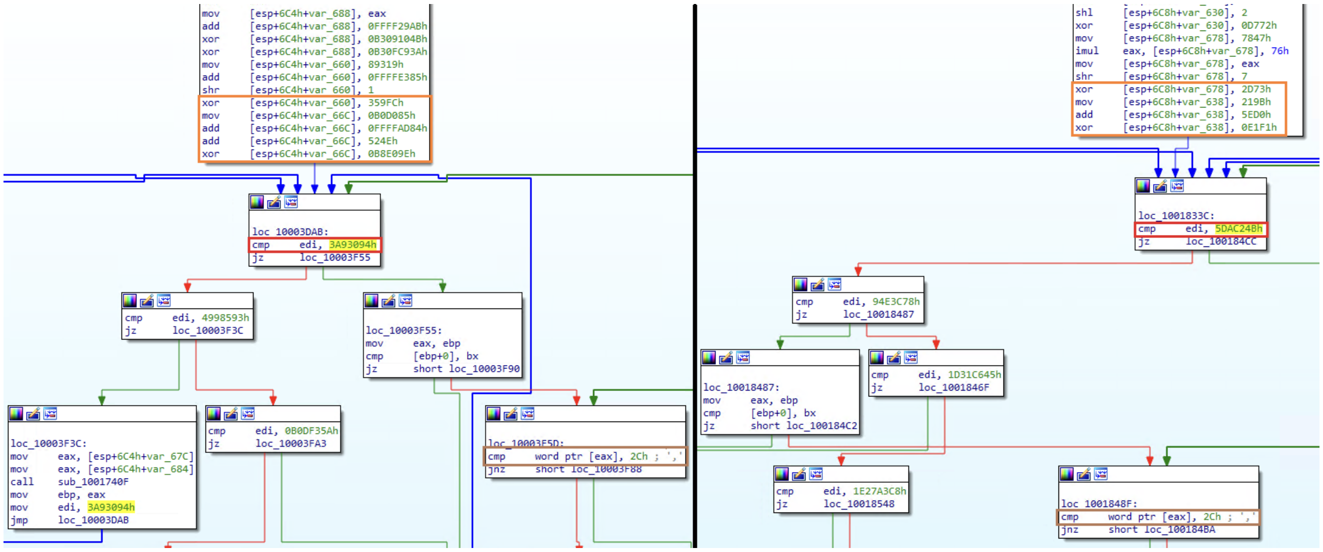
```
100.00% (-39,2924) (25,391) 00004C6F 72CA586F: decrypt_resource+1EF (Synchron
```

```
026EDFC0  0D F0 AD BA 0D F0 AD BA  0D F0 AD BA 0D F0 AD BA  .ð º.ð º.ð º.ð º
026EDFD0  0D F0 AD BA 0D F0 AD BA  0D F0 AD BA 0D F0 AD BA  .ð º.ð º.ð º.ð º
026EDFE0  0D F0 AD BA 0D F0 AD BA  AB AB AB AB AB AB AB AB  .ð º.ð º««««««««
026EDFF0  00 00 00 00 00 00 00 00  DD FB 5C 5E B1 66 00 19  ........Ýû\^±f..
026EE000  4D 5A 90 00 03 00 00 00  04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
026EE010  B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  ¸.......@.......
026EE020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
026EE030  00 00 00 00 00 00 00 00  00 00 00 00 C0 00 00 00  ............À...
026EE040  0E 1F BA 0E 00 B4 09 CD  21 B8 01 4C CD 21 B6 B0  ..º..´.Í!¸.LÍ!¶°
026EE050  F6 A3 A8 F7 ED 51 4B 3C  42 FF AD 1A 69 B9 CC 32  ö£¨÷íQK<Bÿ .i¹Ì2
```

Comparing Variants

In the past, there was a middle stage between the loader and the payload. Based on our previous analysis of the Emotet payload, the file extracted is the payload itself and not a middle stage.

1. The PE file has no imported API functions.
2. There are barely any strings present.
3. The malware utilizes the same code obfuscation techniques.



```
mov     [esp+6C4h+var_688], eax
add     [esp+6C4h+var_688], 0FFFF29ABh
xor     [esp+6C4h+var_688], 0B309104Bh
xor     [esp+6C4h+var_688], 0B30FC93Ah
mov     [esp+6C4h+var_660], 89319h
add     [esp+6C4h+var_660], 0FFFFE385h
shr     [esp+6C4h+var_660], 1
xor     [esp+6C4h+var_660], 359FCh
mov     [esp+6C4h+var_66C], 0B0D085h
add     [esp+6C4h+var_66C], 0FFFFAD84h
add     [esp+6C4h+var_66C], 524Eh
xor     [esp+6C4h+var_66C], 0B8E09Eh
```

```
loc_10003DAB:
cmp     edi, 3A93094h
jz      loc_10003F55
```

```
cmp     edi, 4998593h
jz      loc_10003F3C
```

```
loc_10003F55:
mov     eax, ebp
cmp     [ebp+0], bx
jz      short loc_10003F90
```

```
loc_10003F3C:
mov     eax, [esp+6C4h+var_67C]
mov     eax, [esp+6C4h+var_684]
call    sub_1001740F
mov     ebp, eax
mov     edi, 3A93094h
jmp     loc_10003DAB
```

```
cmp     edi, 0B0DF35Ah
jz      loc_10003FA3
```

```
loc_10003F5D:
cmp     word ptr [eax], 2Ch ; ','
jnz     short loc_10003F88
```

```
shl     [esp+6C8h+var_630], 2
xor     [esp+6C8h+var_630], 0D772h
mov     [esp+6C8h+var_678], 7847h
imul    eax, [esp+6C8h+var_678], 76h
mov     [esp+6C8h+var_678], eax
shr     [esp+6C8h+var_678], 7
xor     [esp+6C8h+var_678], 2D73h
mov     [esp+6C8h+var_638], 219Bh
add     [esp+6C8h+var_638], 5ED0h
xor     [esp+6C8h+var_638], 0E1F1h
```

```
loc_1001833C:
cmp     edi, 5DAC24Bh
jz      loc_100184CC
```

```
cmp     edi, 94E3C78h
jz      loc_10018487
```

```
loc_10018487:
mov     eax, ebp
cmp     [ebp+0], bx
jz      short loc_100184C2
```

```
cmp     edi, 1D31C645h
jz      loc_1001846F
```

```
cmp     edi, 1E27A3C8h
jz      loc_10018548
```

```
loc_1001848F:
cmp     word ptr [eax], 2Ch ; ','
jnz     short loc_100184BA
```

On the left: the new payload. On the right: a payload from January 2021

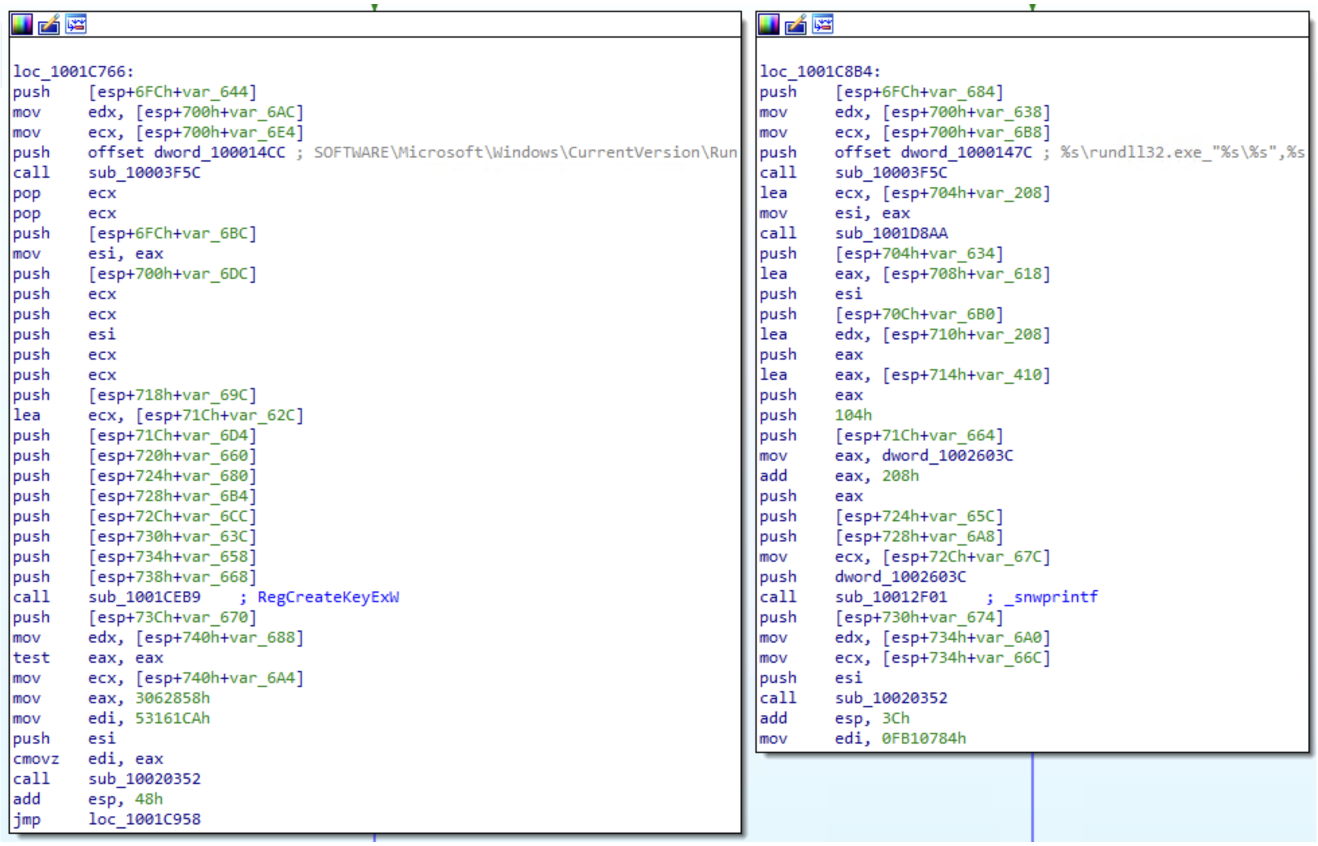
The payload conceals its capabilities by hiding information that is used for static analysis. The names of the API functions are stored in the code after they were hashed. Their address is located in run-time instead of using the Import Address Table. The strings are encrypted inside the file.
The code is obfuscated using Control Flow Flattening, which works as follows:

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

New imported functions from bcrypt.dll were added to the payload. The strings that represent constants for those functions were also added, such as “ECDH_P256” and “Microsoft Primitive Provider.” This is probably due to Emotet changing its communication protocol from HTTP to HTTPS.
The name hashing and strings decryption algorithms were kept. This means that “DeMotet” is still able to discover this information.

Introducing “DeMotet”

Deep Instinct has been closely following Emotet for some time. “DeMotet” was developed to automate the research performed on the malware. The tool is a static unpacker for the latest variant of the Emotet loader. It can extract the encrypted payload from the resource without executing the malware. Python scripts are also included in this tool. They reveal the hidden strings and API calls the payload uses. The first one is a standalone script that can be used to extract this information from a large number of payloads. The second one is an IDA plugin. It adds this information as comments in the code.



```
loc_1001C766:
push    [esp+6FCh+var_644]
mov     edx, [esp+700h+var_6AC]
mov     ecx, [esp+700h+var_6E4]
push    offset dword_100014CC ; SOFTWARE\Microsoft\Windows\CurrentVersion\Run
call    sub_10003F5C
pop     ecx
pop     ecx
push    [esp+6FCh+var_6BC]
mov     esi, eax
push    [esp+700h+var_6DC]
push    ecx
push    ecx
push    esi
push    ecx
push    ecx
push    [esp+718h+var_69C]
lea     ecx, [esp+71Ch+var_62C]
push    [esp+71Ch+var_6D4]
push    [esp+720h+var_660]
push    [esp+724h+var_680]
push    [esp+728h+var_6B4]
push    [esp+72Ch+var_6CC]
push    [esp+730h+var_63C]
push    [esp+734h+var_658]
push    [esp+738h+var_668]
call    sub_1001CEB9    ; RegCreateKeyExW
push    [esp+73Ch+var_670]
mov     edx, [esp+740h+var_688]
test    eax, eax
mov     ecx, [esp+740h+var_6A4]
mov     eax, 3062858h
mov     edi, 53161CAh
push    esi
cmovz   edi, eax
call    sub_10020352
add     esp, 48h
jmp     loc_1001C958
```

```
loc_1001C8B4:
push    [esp+6FCh+var_684]
mov     edx, [esp+700h+var_638]
mov     ecx, [esp+700h+var_6B8]
push    offset dword_1000147C ; %s\rundll32.exe_"%s\%s",%s
call    sub_10003F5C
lea     ecx, [esp+704h+var_208]
mov     esi, eax
call    sub_1001D8AA
push    [esp+704h+var_634]
lea     eax, [esp+708h+var_618]
push    esi
push    [esp+70Ch+var_6B0]
lea     edx, [esp+710h+var_208]
push    eax
lea     eax, [esp+714h+var_410]
push    eax
push    104h
push    [esp+71Ch+var_664]
mov     eax, dword_1002603C
add     eax, 208h
push    eax
push    [esp+724h+var_65C]
push    [esp+728h+var_6A8]
mov     ecx, [esp+72Ch+var_67C]
push    dword_1002603C
call    sub_10012F01    ; _snwprintf
push    [esp+730h+var_674]
mov     edx, [esp+734h+var_6A0]
mov     ecx, [esp+734h+var_66C]
push    esi
call    sub_10020352
add     esp, 3Ch
mov     edi, 0FB10784h
```

"DeMotet” can be used to track new variants of the malware. New samples of Emotet can be downloaded and unpacked regularly. Once the unpacking process of the malware is modified, the tool will fail.

This is an indication of a new variant. The variant will then be manually analyzed to update the tool so the automation can be restored. Finding new strings and imported functions in the payload is also indicative of a new feature.

The tool is available in this GitHub repository.

Summary

The notorious botnet Emotet is back, and we can expect that new tricks and evasion techniques will be implemented in the malware as the operation progresses, perhaps even returning to being a significant global threat. However, by using the techniques and tools presented in this article, the analysis of the malware can be simplified and automated.

Deep Instinct takes a prevention-first approach to stopping ransomware and other malware using the world’s first and only purpose built, deep learning cybersecurity framework. We predict and prevent known, unknown, and zero-day threats in <20 milliseconds, 750X faster than the fastest ransomware can encrypt.

If you’d like to learn more about our malware, ransomware, and zero-day prevention capabilities – including our industry best $3M no-ransomware guarantee – we’d be delighted to give you a demo.

IOC
Loaders SHA256
3b3b65d42e44bcc0df291ddab72f1351784f7e66357a4ec75ee5c982ef556149
6b477d63b3504c6eab3c35057b99d467039995783f5f14714ae6af4f83b9dcb3
442ff2de8a19c3f6cf793f9209ffd21da18aa7eb5b4c4c280222eb9f10a2c68a
9ac36258c63a5edfd29e3ed1882c61487ef2c70637192108cc84eb4ea27f7502
00ceb55abdb43042c6f7fabd327e6e1a6cdefed723dea6c4e90d159b9466518c