

Emotet Malware 0x01

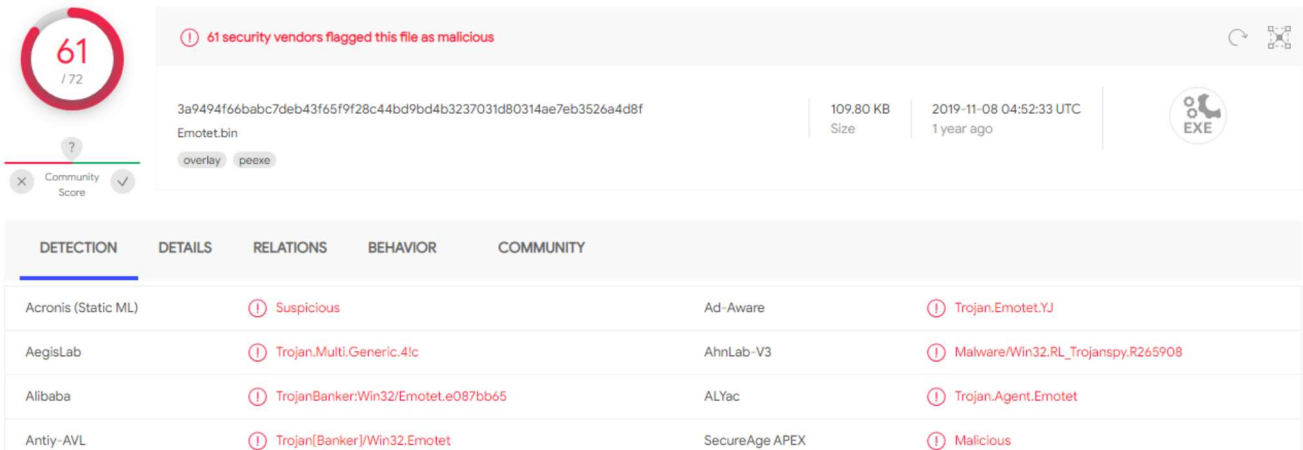
🕒 5 minute read

Introduction

The Emotet banking Trojan was first identified by security researchers in 2014. Emotet was originally designed as a banking malware that attempted to sneak onto your computer and steal sensitive and private information. Later versions of the software saw the addition of spamming and malware delivery services—including other banking Trojans.

virustotal

when scanning malware using VirusTotal website we can see that the malware is detected by 61 out of 70 security vendors as Emotet malware and we can see the results in the next figure.

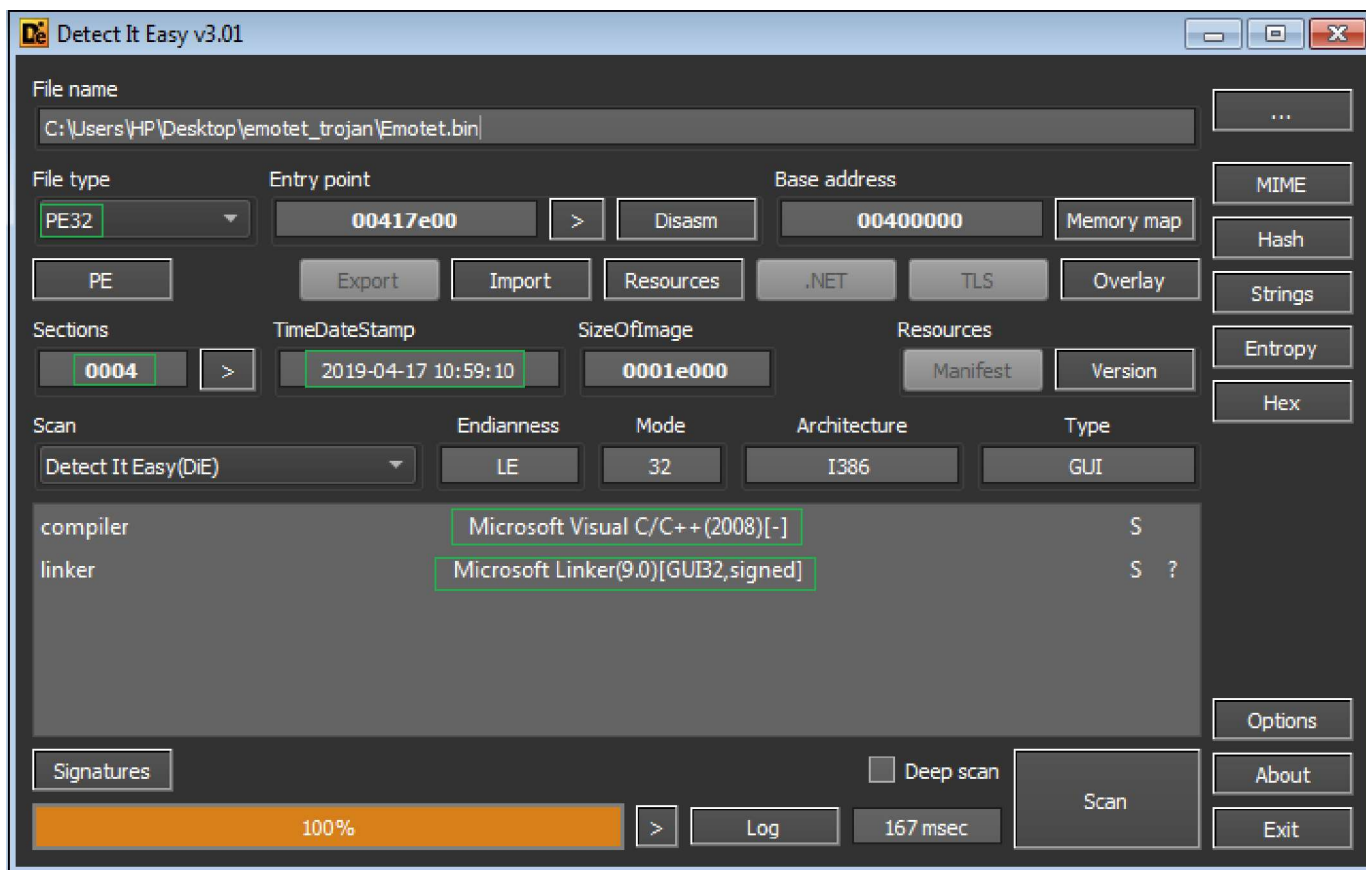


DETECTION	DETAILS	RELATIONS	BEHAVIOR	COMMUNITY
Acronis (Static ML)		⚠ Suspicious	Ad-Aware	⚠ Trojan.Emotet.YJ
AegisLab		⚠ Trojan.Multi.Generic.41c	AhnLab-V3	⚠ Malware/Win32.RL_Trojanspy.R265908
Alibaba		⚠ TrojanBanker:Win32/Emotet.e087bb65	ALYac	⚠ Trojan.Agent.Emotet
Antiy-AVL		⚠ Trojan(Banker)/Win32.Emotet	SecureAge APEX	⚠ Malicious

Examine Emotet with DIE

Basic information

I will open malware with Detect It Easy tool to see some information about Emotet malware like compiler, type of packer (if the malware author used a commercial packer to pack it), type of linker and more information We can see the results in the next figure.



Entropy

From the previous figure, we can identify some information about Emotet malware such as:

- Compiler and linker
- The time that malware has been compiled on (the malware author can fake the time)
- The number of the sections found in the malware. Now I will examine the entropy to see whether the malware is packed or not. Entropy is a good indicator used by malware analysts to detect packing and some other techniques and we can see the results in the next figure.

The screenshot shows the Entropy tool interface with the following details:

- Type:** PE32
- Total Entropy:** 7.15835
- Status:** packed
- Offset:** 00000000
- Size:** 1b738
- Regions Table:**

Name	Offset	Size	Entropy	Status
PE Header	00000000	00000400	2.23745	not packed
Section(0)['.text']	00000400	00017400	7.20283	packed
Section(1)['.rdata']	00017800	00002800	5.85374	not packed
Section(2)['.data']	0001a000	00000200	4.71047	not packed
Section(3)['.rsrc']	0001a200	00000800	3.88502	not packed

From the previous figure we can see the four sections and packed section called ".text" and see the entropy value "7.15835" which gives us a good indicator that this Emotet malware is packed.

Examine Emotet with PeStudio

Sections

Now we will examine the section with pestudio tool to get an overview about sections and some information about every section. So, we can see four sections in the malware and we have one packed section called ".text" that has a high entropy value. We can also see that ".text" section is executable and we can see the results in the next figure.

property	value	value	value	value
name	.text	.rdata	.data	.rsrc
md5	0A0FFF2ED109F8C2235796...	DE5646110B1324F5F57F7B69...	346430862913E3664DDA37F...	2CE248B02CF6EA19407DFB3...
entropy	7.203	5.854	4.708	3.884
file-ratio (96.08%)	84.70 %	9.11 %	0.46 %	1.82 %
raw-address	0x00000400	0x00017800	0x0001A000	0x0001A200
raw-size (108032 bytes)	0x00017400 (95232 bytes)	0x00002800 (10240 bytes)	0x00000200 (512 bytes)	0x00000800 (2048 bytes)
virtual-address	0x00401000	0x00419000	0x0041C000	0x0041D000
virtual-size (107050 bytes)	0x00017318 (95000 bytes)	0x00002606 (9734 bytes)	0x0000023C (572 bytes)	0x000006D0 (1744 bytes)
entry-point	0x00017E00	-	-	-
characteristics	0x60000020	0x40000040	0xC0000040	0x40000040
writable	-	-	x	-
executable	x	-	-	-
shareable	-	-	-	-
discardable	-	-	-	-
initialized-data	-	x	x	x
uninitialized-data	-	-	-	-
unreadable	-	-	-	-
self-modifying	-	-	-	-
virtualized	-	-	-	-
file	n/a	n/a	n/a	n/a

sha256: 3A9494F66BABC7DEB43F65F9F28C448D9BD4B3237031D80314AE7EB3526A4D8F cpu: 32-bit file-type: executable subsystem: GUI entry-point: 0x00017E00

Strings

If we examine the strings of malware, we can identify a lot of malicious strings that give us information about the behavior of the malware and what it tries to do on a system and we can see the blacklist in the next figure.

type (2)	size (bytes)	file-offset	blacklist (70)	hint (11)	group (18)	value (947)
ascii	17	0x0001976E	x	-	windowing	EnumThreadWindows
ascii	12	0x00019783	x	-	windowing	GetClassLong
ascii	22	0x00019022	x	-	system-information	GetTimeZoneInformation
ascii	18	0x00019759	x	-	system-information	EnumDisplayDevices
ascii	17	0x00019AC3	x	-	system-information	SHQueryRecycleBin
ascii	17	0x00019AD9	x	-	system-information	SHQueryRecycleBin
ascii	22	0x00019B25	x	-	system-information	SHGetSpecialFolderPath
ascii	19	0x00018EC4	x	-	synchronization	GetOverlappedResult
ascii	19	0x00019417	x	-	storage	GetCurrentDirectory
ascii	14	0x00019C00	x	-	shell	SHChangeNotify
ascii	21	0x000197D2	x	-	security	GetUserObjectSecurity
ascii	24	0x0001995A	x	-	security	AllocateAndInitializeSid
ascii	7	0x00019976	x	-	security	CopySid
ascii	8	0x00019980	x	-	security	EqualSid
ascii	12	0x0001998C	x	-	security	GetLengthSid
ascii	25	0x00019A1E	x	-	security	SetSecurityDescriptorDacl
ascii	26	0x00019A3A	x	-	security	SetSecurityDescriptorOwner
ascii	14	0x00019595	x	-	resource	UpdateResource
ascii	12	0x000199DB	x	-	registry	RegCreateKey
ascii	13	0x00019A0D	x	-	registry	RegSetValueEx

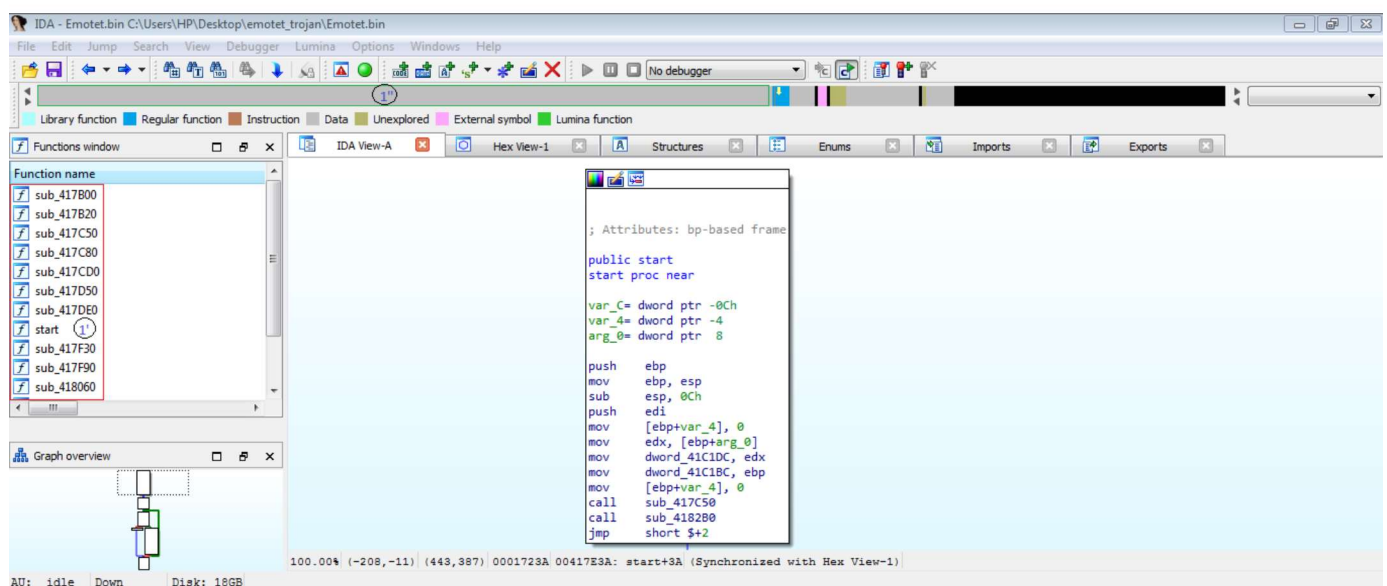
sha256: 3A9494F66BABC7DEB43F65F9F28C448D9BD4B3237031D80314AE7EB3526A4D8F cpu: 32-bit file-type: executable subsystem: GUI entry-point: 0x00017E00

Examine Emotet with ida pro

Identify Packed Emotet

After examining the malware with a static analysis tool, we can use a great tool called "IDA Pro" to help us understand assembly code and functionalities of malware. It is also a great tool to identify packed malware with some techniques and we can see the following techniques.

- There are some of functions which used by the malware
- Most of the above bar has one color only (Navigation bar) and we can see the results in the next figure



from the previous figure, we can see two indicators telling us that this Emotet malware is packed. After examining the malware with IDA Pro, we can get also some common techniques used by malware analysts to identify packed malware.

If a malware is packed, we can search in code for:

- jump with register
- jump with a region of memory

This is because malware authors use custom packers which throw the malware into multiple loops and some encryption routines to decrypt payload in runtime (to avoid antivirus and any security solution tools). So, if we go to start function from **exports** and click on "start" function and examine the code there, we can identify an interesting function that we will deep into to see how the malware author used it to unpack the malware in runtime. We can see the name of function in the next figure from the "start" function.

```

; Attributes: bp-based frame

public start
start proc near

var_C= dword ptr -0Ch
var_4= dword ptr -4
arg_0= dword ptr 8

push    ebp
mov     ebp, esp
sub     esp, 0Ch
push    edi
mov     [ebp+var_4], 0
mov     edx, [ebp+arg_0]
mov     dword_41C1DC, edx
mov     dword_41C1BC, ebp
mov     [ebp+var_4], 0
call   sub_417C50
call   sub_4182B0
jmp     short $+2

```

```

loc_417E30:
call   sub_4180A0
push   2C58h
call   sub_417D50
add    esp, 4
mov    dword_41C1C8, 0
mov    eax, dword_41C1C8

```

Structure of function

After going into the function, we must first talk about the function's prologue and epilogue. Every function in assembly code has three parts. The first part is the "prologue", while the third part is the "epilogue". And we know that the first and third parts are simply a set of instructions that set up the context for the function when it is called or cleaned. The prologue performs such operations as:

- saving any registers that the function uses.
- allocating storage on the stack that the function needs to store the local variables.
- setting up pointers, such as parameters passed to function. The epilogue restores any saved registers, as well as the stack pointer.

syntax of function

we will show the syntax in the function and this syntax depends on the architecture that we work on. So, the syntax can be different from one architecture to another.

We can see the following syntax for x86 assembly:

```

1:  push ebp
2:  mov  ebp, esp    ---> prologue
3:  ...
4:
5:  main code to do operation
6:
7:  ...
8:  mov  esp, ebp
9:  pop  ebp        ---> epilogue
10: retn

```

Code analysis

unpacking Function

But when we see the epilogue the function we can see that it is different from normal epilogues, and that the malware authors used it to complicate the process of reverse engineering. We can see the results in the next figure

```

; Attributes: bp-based frame

sub_417D50 proc near

var_14= dword ptr -14h
var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4

push  ebp (1)
mov   ebp, esp
sub   esp, 14h
mov   [ebp+var_4], 40h ; '@'
mov   [ebp+var_C], 0
mov   eax, dword_41C1A4
mov   [ebp+var_14], eax
mov   [ebp+var_8], 0FFFFFFFFh
mov   ecx, ds:VirtualAlloc (1)
mov   dword_41C218, ecx
push  [ebp+var_4]
push  3000h
push  [ebp+var_14] (1)
push  [ebp+var_C]
mov   ecx, dword_41C218
push  offset loc_417D9A
push  ecx
retn  (1)

loc_417D9A:
mov   [ebp+var_10], eax
mov   edx, [ebp+var_10]
mov   dword_41C1E8, edx
mov   eax, dword_41C1A4
mov   dword_41C1A8, eax
mov   dword_41C1B4, 0
mov   ecx, dword_41C1E8
add   ecx, 102F0h
mov   dword_41C1B4, ecx
mov   eax, [ebp+var_10]
mov   esp, ebp
pop   ebp (1)
retn

sub_417D50 endp ; sp-analysis failed

```

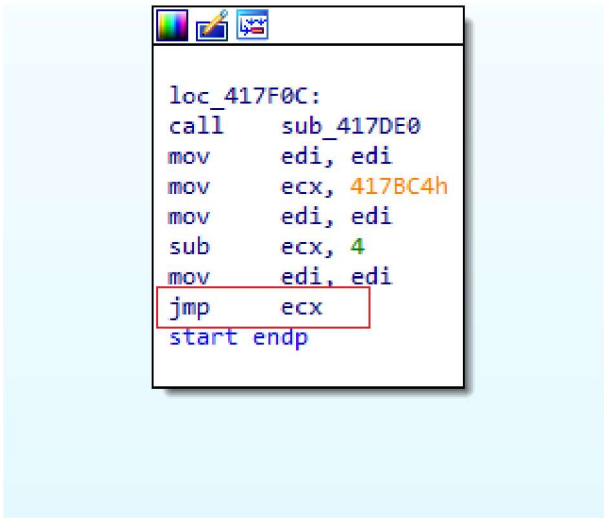
From the previous figure. We can see that:

- VirtualAlloc is moved into ecx
- Then ecx is moved into dword_41C218
- And also dword_41C218 is moved into ecx
- Finally, we can see push ecx followed by ret instruction.
So we can identify that function is used to unpack malware.

jump Instruction

If we return to the start function and dive deep into code, we can see `jump ecx`. This is a good start for debugging the code in x32dbg.

We can see the instruction in the end of the "start" function:



```
loc_417F0C:  
call    sub_417DE0  
mov     edi, edi  
mov     ecx, 417BC4h  
mov     edi, edi  
sub     ecx, 4  
mov     edi, edi  
jmp     ecx  
start endp
```

We will show how to play with the code in the debugger to unpack Emotet malware in the next part.

Categories: Malware-Analysis

Updated: August 1, 2021