Advanced Cyberchef Techniques - Defeating Nanocore Obfuscation With Math and Flow Control

embeeresearch.io/advanced-cyberchef-techniques-defeating-nanocore-obfuscation-with-math-and-flow-control/

Matthew

September 3, 2024

CyberChef Tutorials

Applying Flow Control and Mathematical operators to deobfuscate a .vbs loader for Nanocore malware.



Introduction

Cyberchef is an incredible tool with powerful features that are rarely documented and can significantly aid an analyst in their efforts to deobfuscate malware.

Today we will be investigating such features and how they apply to defeating the obfuscation of a recent .vbs loader for Nanocore malware.

Our Analysis and Deobfuscation Will Cover...

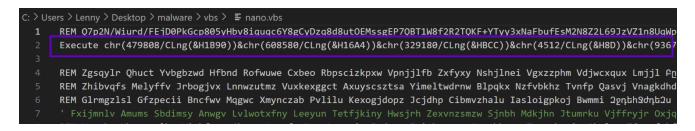
- ASCII Charcodes and Character Conversions
- Alternating Decimal and Hex Values
- Alternating Mathematical Operations (Addition/Division)
- Flow Control and Isolation of Values Using Subsections.
- Lots of regex!

SHA256:c6092b1788722f82280d3dca79784556df6b8203f4d8f271c327582dd9dcf6e1

Initial Analysis and Overview of Obfuscation

The sample in it's initial state contains ~160 lines of code. The majority of this consists of comments that don't contribute to the functionality of the code.

The primary piece of code exists on line 2 and can be seen below. Our analysis will focus only on this line of code.

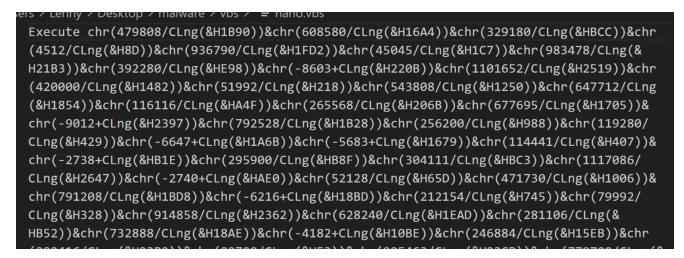


Since our focus is going to be on line 2, we can ignore the remainder of the initial script and remove them using a regular expression.

The goal of the regular expression is identify lines that begin with ${\tt REM}$ or ', and to capture everything on that line that follows .*

^(REM ').*	Aa _ab 📑	2 of 168 个	$\downarrow \equiv \times$
Replace	AB		-
w Vpnjjitb Z	xtyxy Nshjinei Vgxz	zphm Vdjwcxqux	ւլայյ⊥ Բրՙ

Executing the regular expression as a Find/Replace results in the following content. The comments are now removed and we can focus only on line 2 and it's obfuscation tactics.



Intial Review of Obfuscation

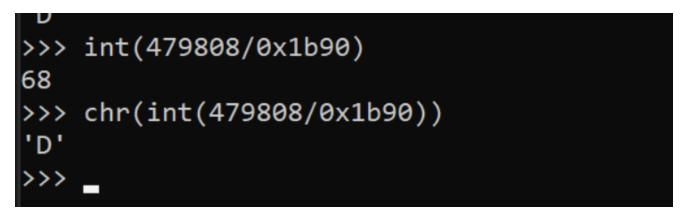
The obfuscation consists of the same pattern repeated over and over again to produce single characters. These characters are concatenated together to form the deobfuscated code.

There are 3 primary pieces of the obfuscation.

- 479808 Large Decimal Value, this will be converted into a smaller number using math operations.
- (&H1b90) This is a vbs representation of the hex value 0x1B90.
- CLng This is the function "Change Long", this <u>converts</u> the hex representation into a numerical value.
- / This divides the numbers 479808 and 0x1b90. Resulting in a value in the ASCII range.
- chr The result of the division is converted into a character which will form part of the resulting script.



The logic is more clear when shown in Python. Here we can see that chr(479808/Clng(&H1B90)) is equal to the character D.



We've now identified the core concept of the obfuscation, so we can go ahead and recreate this in Cyberchef for the entire obfuscated content.

Deobfuscation With Cyberchef

The obfuscation has now been identified, so we can begin to recreate the logic in Cyberchef.

We can begin by isolating the encoded portions with a regular expression chr\([^\)]+.

For the sake of prototyping, we have selected only a small portion of the obfuscated code. This will allow us to get the recipe working before adding the complete script at the end of our analysis.

		.,		······································
Recipe		2 🖿 🕯	Input	+ 🗅 🖻 🛢 📰
Regular expression 🛇 11		chr(479808/CLng(&H1B90))&chr(608580/CLng(&H16A4))&chr(329180/CLng(&HBCC))&chr(4512 /CLng(&H8D))&chr(936790/CLng(&H1FD2))&chr(45045/CLng(&H1C7))&chr(983478/CLng(&H21B3))&		
Built in regexes User defined			/CLng(&HE98))&chr(-8603+CLng(&H220B))&chr(110	1652/CLng(&H2519))&chr(420000/CLng(&H1482))
Regex chr\([^\)]+				
		11.	ABC 266 📻 1	Tt Raw Bytes ↔
Case insensitive	✓ ^ and \$ ma	atch at newlines	Output chr(479808/CLng(&H1890))&chr(608580/CLng(&H166	
Dot matches all	//		/CLng(&H8B))&Chr(936/90/CLng(&H1FD2))&Chr(450 /CLng(&HE98))&Chr(-8603+CLng(&H220B))&Chr(110	<mark>45/CLng(&H1C7</mark>))&chr(983478/CLng(&H21B3))& <mark>chr(392280</mark> 1652/CLng(&H2519))&chr(420000/CLng(&H1482))
Astral support	Display total	Output format Highlight matc		

Isolating Values With Regular Expressions and Capture Groups

Once the regex is matching as intended using "Highlight Matches", we can change to "List Capture Groups".

This will list out the encoded portions on their own individual lines.

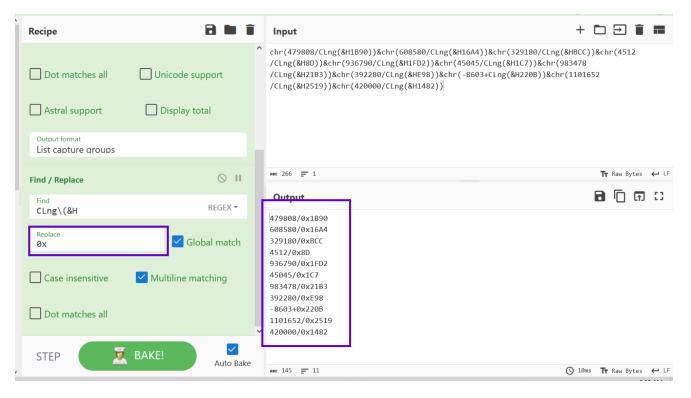
Recipe	2 🖿 🕯	Input	+ 🗅 🖯 î 🖬
Regular expression	⊘ 11	chr(479808/CLng(&H1B90))&chr(608580/CLng(&H16A4))&chr(329180/CLng(&HBCC))&c /CLng(&H8D))&chr(936790/CLng(&H1FD2))&chr(45045/CLng(&H1C7))&chr(983478/CLn	g(&H21B3))&chr(392280
Built in regexes User defined		/CLng(&HE98))&chr(-8603+CLng(&H220B))&chr(1101652/CLng(&H2519))&chr(420000/	CLng(&H1482))
Regex chr\(([^\)]+)			
		ABC 266 = 1	Tt Raw Bytes 🔶 L
Case insensitive	✓ ^ and \$ match at newlines	Output	809:
Dot matches all	 And \$ match at newlines Unicode support Astral support 	479808/CLng(&H1B90 608580/CLng(&H16A4 329180/CLng(&HBCC 4512/CLng(&HBD 936799/CLng(&H1FD2	
Display total	Output format List capture groups	45045/CLng(&H1C7 983478/CLng(&H2183 392280/CLng(&HE98	
		-8603-CLng(&H220B 1101652/CLng(&H2519 420000/CLng(&H1482	

Normalising Hexadecimal Content

We now want to clean up the second half of each line by removing the references to CLng(&H.

The original code is in a format that Visual Basic understands. We want to be in a format that can be understood by Cyberchef. Our primary goal is to make sure that Cyberchef knows the difference between the hex and decimal numbers.

We can do this with a Find/Replace operation, which will replace the CLng(&H with a 0x.



Here is where things start getting more complicated....

As we saw before, the decimal and hex values are separated by mathematical operators. The operators are mostly division / but occasionally are addition + as well.

If we apply a division operator, it will break the lines that require addition. And vice versa. This means we need to separate the lines of code that require different mathematical operators.

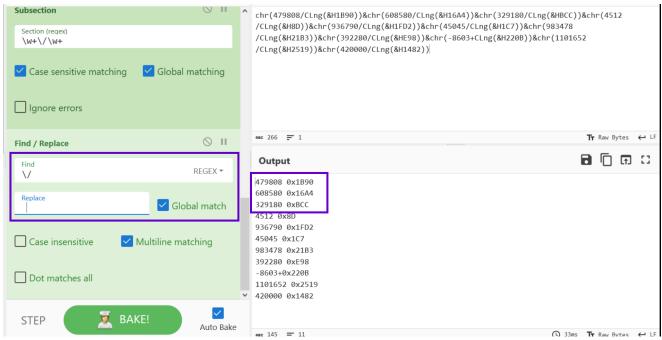
We can do this with Regular Expressions and a Subsection operation. A subsection will apply future operations only to lines that match the provided regex.

Below we can see the regular expression of w+//w+, this will isolate the lines of code that contain a division operator.

Recipe	Î	Input +	۵ 3		=
Øx ✓ Global match □ Case insensitive ✓ Multiline matching □ Dot matches all ✓	^	chr(479808/CLng(&H1B90))&chr(608580/CLng(&H16A4))&chr(329180/CLng(&HBCC /CLng(&HBD))&chr(936790/CLng(&H1FD2))&chr(45045/CLng(&H1C7))&chr(983478 /CLng(&H21B3))&chr(392280/CLng(&HE98))&chr(-8603+CLng(&H220B))&chr(1101 /CLng(&H2519))&chr(420000/CLng(&H1482))	3	512	
Split delimiter Merge delimiter		nec 266 = 1	Tr Ra	/ Bytes	← LF
\n \n Ignore errors		Output] ि	::
Subsection O II		479808/0x1B90 608580/0x16A4			
Section (reqex) \w+\/\w+		329180/0xBCC 4512/0xBD			
Case sensitive matching Global matching		936790/0x1FD2 45045/0x1C7 983478/0x21B3 392280/0xE98			
Ignore errors	×	-5603+0x2208 1101652/0x2519 420000/0x1482			
STEP Z BAKE!	e				

Before applying a division operation, we need to add a delimiter to our divided values. Most math operations in Cyberchef require a "list" of values rather than an equation.

The TLDR here is that we need to turn the / into spaces. Luckily we can do this with a simple Find/Replace.



Now that we have applied spacing on the division lines, we can apply a Divide operation and specify a space delimiter.

We can see that this converts the division lines into their repective ASCII charcodes.

Find / Replace	⊘ 11		
Find \/	REGEX -		
Replace	Global match	AND 266 = 1	Tr Raw Bytes ↔ LF
		Output	C 🖬 🗍 🖬
Case insensitive	✓ Multiline matching	68 105	
Dot matches all		109 32 115	
Divide	0 11	99 114 105	
Delimiter Space		-8603+0x220B 116	
		× 80	
STEP	BAKE!	e sec 48 = 11	🕓 17ms 🏾 🏹 Raw Bytes 🛩 LF

With the Charcodes ready, we can apply a simple "From Decimal" to produce the relevant ASCII character.

Now we can see the beginning of the decoded script.

Case insensitive Vultiline matching	
Dot matches all	nuc 266 = 1 Output
Divide 🛇 II	D i
Delimiter Space	m s
From Decimal 🚫 🔢	c r i
Delimiter Space Support signed values	-8603+0x220B t ▼ P
STEP Z BAKE! Auto Bake	явс 32 🚍 11

Now we need to deal with the lines of code containing addition + operators.

Since we previously applied a subsection, we need to leave the subsection and change it to focus on the addition lines.

To leave a subsection, we can apply a Merge operation. We should also uncheck "Merge All" as there is only a single subsection that we want to leave.

Diride			
Delimiter		RBC 266 = 1	Tr Raw Bytes 🔶 LF
Space			
Space		Output	🖬 🗍 🖬 🖸
From Decimal	0 11	þ	
Delimiter Space	Support signed values	m S	
Merge	0 11	c r i	
Merge All	~	-8603+0x220B t P	
STEP	Z BAKE!		() 47ar 12 0ar 0.000 del 10

Subsections and Isolating Specific Lines of Content

After leaving the Subsection for division, we can create a new Subsection specifically for Addition.

We can do this with another regular expression $-?\w+\+\w+$. This regular expression accounts for the negative values which may be present.

Merge 🚫 II			
Merge All		Nec 266 = 1	
		Output	
Subsection 🛇 II		þ	
Section (reaex) - ? \\\+\\\+		i m	
Case sensitive matching Case sensitive matching		s c r	
Ignore errors	×	1 -8603+0x220B t p	
STEP Z BAKE! Auto Bake		мас 32 📻 11	S
			8 6

Similar to the division operation, we need to remove the + operators and turn the lines into a list separated by a space.

We can this again with a simple Find/Replace

Find / Replace	RBC 266 = 1
Find \+ REGEX ~	Output
Replace Global match	D i m
Case insensitive Multiline matching	s c r
Dot matches all	i -8603 0x220B
	у р
STEP Z BAKE! Auto Bake	
	ABC 32 = 11

Now that we have a clean list for our addition lines, we can apply a SUM operation.

This will add the values together and produce an ASCII charcode.

Find / Replace	
Find REGEX *	
Replace Global match	ne 266 = 1
	Output
🗌 Case insensitive 🛛 🗹 Multiline matching	D i
Dot matches all	m S
Sum	
Delimiter Space	112
	✓ P
STEP 🗵 BAKE!	ваке ввс 23 = 11

We can now apply a From Decimal operation to obtain the resulting character.

The obfuscated script now looks much better and no longer contains obfuscated content.

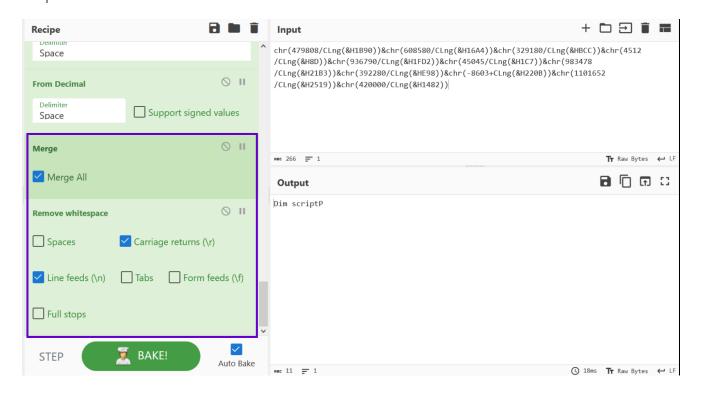
Case insensitive Multiline matching	
Dot matches all	nuc 266 = 1 Output
Sum 🛇 II	Di
Delimiter Space	m S
From Decimal 🚫 🔢	c r
Delimiter Space Space	i p t v
STEP Z BAKE! Auto Bake	auc 21 📻 11

Our deobfuscation prototype is complete, so we can go ahead and remove all subsections and the newlines that separated them.

We can do this with a Merge -> Merge All and Remove Whitespace -> \r + \n

The code output now looks clean and easily readable. So we can go back and add the original obfuscated content.

Note that we could repeat the previous process if there were other mathematical operations. This script only contains Addition and Division



Pasting in the full obfuscated content, we can see the complete deobfuscated result.

Recipe	2 🖬 🕯	i	Input	+		∋	Î	
Delimiter Space		^	<pre>kecute chr(479808/CLng(&H1B90))&chr(608580/CLng(&H16A4))&chr(329180 CLng(&HBCC))&chr(4512/CLng(&H8D))&chr(936790/CLng(&H1FD2))&chr(45045 CLng(&H1C7))&chr(983478/CLng(&H21B3))&chr(392280</pre>					^
From Decimal Delimiter Space	Support signed values		<pre>/CLng(&HE98))&chr(-8603+CLng(&H220B))&chr(1101652/CLng(&H2519))&chr /CLng(&H1482))&chr(51992/CLng(&H218))&chr(543808/CLng(&H1250))&chr /CLng(&H1854))&chr(116116/CLng(&HA4F))&chr(265568/CLng(&H206B))&chr /CLng(&H1705))&chr(-9012+CLng(&H2397))&chr(792528/CLng(&H1828))&chr /CLng(&H988))&chr(119280 /CLng(&H492))&chr(-6647+CLng(&H166B))&chr(-5683+CLng(&H1679))&chr(1 /CLng(&H407))&chr(-2738+CLng(&H181E))&chr(295900/CLng(&H18F))&chr(30)</pre>	5477 (677) (256)	12 695 200 1			
Merge	⊘ 11		/CLng(&HBC3))&chr(1117086/CLng(&H2647))&chr(-2740+CLng(&HAE0))&chr(1		8	• Raw Byt	tes 🗲	- LF
🗹 Merge All			Output		8			3
Remove whitespace	⊘ II ✓ Carriage returns (\r)		<pre>Dim scriptPath, scriptFolder, sourcePath, destinationPath, scriptNameWithoutExtensionDim fso, shellSet fso = CreateObject("Scripting.FileSystemObject")scriptPath = WScript.ScriptFullNamescriptFolder = fso.GetParentFolderName(scriptPath)scriptNameWithoutExtension = Cos GetFileNers(cosithDath(scriptPath)scriptNameWithoutExtension = Cos GetFileNers(cosithDath(scriptPath))</pre>					^
✓ Line feeds (\n) ☐ Full stops	Tabs Form feeds (\f)	~	<pre>fso.GetFileName(scriptPath)destinationPath = scriptFolder & "\" & scriptNameWithoutExtension & ".exe"Dim part1, part2, part3, part4, part8, part9, part10, part11, part12, part13, fullPathpart1 = "%S" part3 = "Root%" part4 = "\Sys" part5 = "WOW" part6 = "64" part7 "wsPow" part9 = "erShell" part10 = "\v1." part11 = "0\pow" part2 = "11.exe"fullPath = part1 & part2 & part3 & part4 & part5 & part6 & """</pre>	oart = " = "e part	2 = " \Wind rshe"	'ystem" lo" par ' part1	rt8 = 13 =	
STEP	Z BAKE!		<pre>part9 & part10 & part11 & part12 & part13Dim expandedPathSet shell = CreateObject("WScript.Shell")expandedPath = shell.ExpandEnvironmentStrings(fullPath)sourcePath = expandedPathDim mec 3011 = 1</pre>			Service Raw Byt		↓ LF
2 🔒 🖪			ĉ 📼	<u>G</u>	€ ¢)) ENG	8:23 A 2/09/20	M 024

We have now deobfuscated line 2 of the initial script. We won't focus on the remainder of the code, but it effectively executes a powershell command that runs a Nanocore payload.

Of interest is that the Nanocore payload is contained in the comments of the initial script.

Since we initially removed these comments, we would need to restore them to obtain the final payload.

Link To The Sample

The sample can be found on Malware Bazaar with the following SHA256 and Link.

SHA256: c6092b1788722f82280d3dca79784556df6b8203f4d8f271c327582dd9dcf6e1

CyberChef Recipe

The complete Cyberchef recipe can be found below.

```
Regular_expression('User defined','chr\\
(([^\\)]+)',true,true,false,false,false,false,'List capture groups')
Find_/_Replace({'option':'Regex','string':'CLng\\(&H'},'0x',true,false,true,false)
Fork('\\n','\\n',false)
Subsection('\\w+\\/\\w+',true,true,false)
Find_/_Replace({'option':'Regex','string':'\\/'},' ',true,false,true,false)
Divide('Space')
From_Decimal('Space',false)
Merge(false)
Subsection('-?\\w+\\+\\w+',true,true,false)
Find_/_Replace({'option':'Regex','string':'\\+'},' ',true,false,true,false)
Sum('Space')
From_Decimal('Space',false)
Merge(true)
Remove_whitespace(false,true,true,false,false,false)
```

Recipe						^ 🖻 🖿 🖠
Regular expression						^ ⊗ II
Built in regexes User defined	Regex chr\(([^\))]+)			Case insensitive	^ and \$ match at newlines
Dot matches all	🗌 Uni	code support	Astral suppor	t 🗌 Dis	splay total	Output format List capture groups
Find / Replace						^ ⊗ II
Find CLng\(&H	REGEX 🕶	Replace Øx		Global match	Case insensi	tive 🗹 Multiline matching
Dot matches all						
Fork						∧ ⊗ II
Split delimiter ∖n			lerge delimiter N		Ignore errors	
Subsection						∧ ⊗ II
Section (reqex) \W+\/\W+			Case sensitive mate	:hing 🔽	Global matching	Ignore errors
Find / Replace						^ ⊗ II
Find V	REGEX 🕶	Replace		Global match	Case insensi	tive 🔽 Multiline matching
Dot matches all						
Divide						^ ⊗ II
Delimiter Space						
From Decimal						∧ ⊗ II
Delimiter Space				Sup	port signed values	
Merge						^ ⊘ Ⅱ
Merge All						Ī
Subsection						∧ ⊗ II
Section (regex) -?\w+\+\w+			Case sensitive mate	:hing 🔽	Global matching	Ignore errors
Find / Replace						^ ⊗ II
Find \+	REGEX *	Replace		Global match	Case insensi	tive 🗹 Multiline matching
Dot matches all						
Sum						^ ⊗ II
Delimiter						

Space				
From Decimal				^ ⊗ II
Delimiter Space		🗌 Suj	pport signed values	
Merge				^ ⊗ II
Merge All				
Remove whitespace				^ ⊗ II
Spaces Carriage returns (\r)	✓ Line feeds (\n)	Tabs	Form feeds (\f)	Full stops