Don't Judge a PNG by Its Header: PURELOGS Infostealer Analysis

By Louis Schürmann

Published: 2026-01-19 · Archived: 2026-04-05 20:30:10 UTC

Swiss Post Cybersecurity traced a suspicious JavaScript file back to a stealthy PURELOGS campaign hiding its payload within a PNG.

Louis Schürmann, Security Analyst of Swiss Post Cybersecurity identified and analyzed a previously unnoticed PURELOGS stealer campaign. In our blog article, he describes the complete attack chain, from the initial use of legitimate infrastructure to the final data exfiltration.

Phishing triage is usually predictable, but this sample was an exception. Masquerading as a pharmaceutical invoice, the loader immediately reached out to archive.org to fetch a PNG image. This caught the attention of our Security Analyst. After four layers of deobfuscation, he discovered PURELOGS. While the malware itself is a known commodity, the staging infrastructure in this campaign offers a valuable case study in evasion.

Initial Access: The JScript Dropper

The infection starts with a phishing email masquerading as a pharmaceutical invoice. Inside the ZIP archive is a file with a .js extension. Most users see JavaScript and think "browser stuff," but this isn't browser JavaScript.

It's a Windows Script Host (WSH) JScript file, meaning it executes with full OS-level privileges through the Windows scripting engine. It gets direct access to COM objects like WScript.Shell and Win32_Process, so it can create files, launch processes, and interact with the system however it wants.

At first glance, the file is unreadable:

It's packed with non-ASCII characters that break static analysis and mess with signature-based detection. Strip out the junk characters though, and the script's logic is pretty straightforward: it builds a PowerShell command with a Base64-encoded payload and fires it off using WMI.

It launches a hidden PowerShell process and runs the decoded payload in memory with Invoke-Expression. No file hits disk, so basic file-based AV doesn't see it. Standard fileless execution. What caught our Security Analyst's attention was what the PowerShell payload did next: it started looking for a PNG file on archive.org.

Stage 1: The Polyglot PNG

The decoded PowerShell script acts as the first stage downloader. Instead of fetching an executable from some disposable domain, it downloads a PNG image from archive.org, a legitimate and well-known website. When analysts review network logs and see traffic to archive.org, it typically doesn’t typically raise flags. The attackers are using the site's reputation as cover.

But this isn't actually a standard PNG. Well it is, but with extras. The attackers embedded a Base64-encoded payload after the IEND chunk of the PNG, which marks the official end of the image data. The file still renders as a valid image in any viewer. The actual malware sits between two custom markers, BaseStart- and -BaseEnd.

The PowerShell script uses regex to extract the payload between these markers:

```
$imageData -match 'BaseStart-(.+?)-BaseEnd'
$valor = $matches[1]
```

Notice that it uses DownloadString() instead of DownloadFile(). The "image" never touches the disk in its original form. It only exists in memory as a string variable. File based security controls never get a chance to inspect it.

Once extracted, the script Base64-decodes the payload and loads it directly into memory using .NET Reflection:

This is where fileless execution really matters. Traditional antivirus scans files on disk, calculate hashes, and match signatures. However, when an assembly loads directly from a byte array into memory, it bypasses all of that. The malware only exists in the PowerShell process's memory space.

The script then passes several arguments to the loaded assembly, including another encoded URL, and invokes a method using Reflection. We've moved from a simple dropper to a configurable loader, and the next stage is where things get more complex.

Stage 2: VMDetectLoader Configuration

The .NET assembly loaded by the PowerShell stager is what IBM X-Force researchers call "VMDetectLoader". This modular loader is responsible for persistence options, environment checks, and injecting the next stage. Its behavior is entirely dictated by the arguments passed from Stage 1.

In this campaign, the attackers made some specific configuration choices:

No Persistence: The loader supports multiple persistence mechanisms (VBScript with Run registry keys, scheduled tasks) but none were enabled here.

VM Detection Enabled: A standard sandbox evasion check. If the loader detects that it's running in a virtual machine, it terminates. Most automated malware analysis relies on VMs, this helps the malware avoid early detection.

Target Process: CasPol.exe: This is the host process that the loader will use for injection. CasPol.exe is a legitimate .NET Framework tool (Code Access Security Policy Tool), which makes it the perfect cover. Security tools see it as a trusted Microsoft utility.

One interesting detail: many of the internal variable names are in Portuguese (like “nativo” for "native"). IBM X-Force attributes VMDetectLoader to Hive0131, a South American threat group. However, since VMDetectLoader is the only indicator linking this PURELOGS campaign to that threat actor, the attribution is considered as low confidence.

The loader's main job is to fetch the next stage payload. It takes one of the arguments from Stage 1 (stored in the $olinia variable), reverses it, and Base64-decodes it to get a URL pointing to a .txt file.

This file contains yet another encoded PE, which the loader fetches, decodes, and prepares for injection using a technique called process injection.

Process Injection via RunPE

Once the next stage payload is decoded and in memory, the loader's final task is to execute it. To accomplish this, the loader employs a classic, well-documented process injection technique known as *RunPE* , or *process hollowing*.

It launches the legitimate .NET Framework utility CasPol.exe in a suspended state, removes its original code from memory, and replaces it with the decoded payload. By hijacking the main thread and redirecting the instruction pointer to the payload's entry point, the malware effectively masquerades as a trusted Microsoft process. From the perspective of the operating system and many security tools, CasPol.exe is simply running as expected, allowing the next stage to begin its work as a “trusted” process.

Stage 3: The Secondary Unpacker

The payload injected via process hollowing is not the final PURELOGS stealer. Instead, it's a .NET unpacker obfuscated with .NET-Reactor. Its sole purpose is to decrypt, decompress, and execute the final payload entirely within memory.

An Event-Driven Unpacking Pipeline

The unpacker's code initially appears disorganized. The ExecutionEngine class constructor, however, reveals the actual architecture: each method is subscribed to a specific event, forming a sequential pipeline where completing one step triggers the next in the chain.

This architecture breaks the unpacking process into five distinct, sequential stages. The Main method simply kicks off this chain by creating an ExecutionEngine instance and triggering the first event.

Step 1: Decryption with Legacy 3DES

The first operational step is decryption with DecompressDispatcher. This function takes the encrypted payload, a key, and an initialization vector (IV) from the unpacker's embedded resources.

The choice of cryptography here is noteworthy. Instead of a modern standard like AES, the malware uses the Triple DES (3DES) algorithm. 3DES is a legacy cipher that has been officially deprecated by NIST for being too slow and inefficient. Its use here is likely a deliberate choice to evade automated detection systems. Many security tools are configured to look for the cryptographic constants and signatures associated with AES, using an older, less common algorithm can help the malware fly under the radar.

Step 2: Decompression with GZip

After decryption, the resulting data is still not a valid PE file. It is a GZip-compressed archive. The next event in the chain triggers the decompression process: EncryptEfficientDecryptor.

This two-layer approach of encryption followed by compression is a common and effective technique for protecting a payload. It ensures that the final assembly is never exposed until the last possible moment, shielding it from static analysis tools that might otherwise flag it.

Step 3 + 4: Assembly Loading and Final Handover via Reflection

With the final PURELOGS assembly now fully decompressed in memory, the last two steps of the pipeline are dedicated to executing it. This is accomplished using .NET Reflection to achieve a complete fileless handover.

1. **Load Assembly:** The *LoadAssemblyFromBytes* method is triggered, which calls *Assembly.Load()* on the raw byte array of the PURELOGS payload. The assembly is now loaded into the unpacker's memory space.
2. **Invoke Entry Point:** The final event triggers the *InvokeEntryPoint* method. This method uses reflection to find and execute the entry point of the newly loaded assembly.

Stage 4: PURELOGS at last

After four stages of unpacking and injection, the PURELOGS stealer is now running inside the hollowed *CasPol.exe* process.

Meet PURELOGS

PURELOGS is a commodity .NET infostealer that first appeared for sale on various underground forums in 2022. It is developed and sold by a developer known as PureCoder, who also offers a suite of other malicious tools, such as PureRAT, BlueLoader, and PureCrypter.

As most infostealers nowadays PURELOGS operates as a Malware-as-a-Service (MaaS), making it accessible to a wide range of threat actors, regardless of their technical skill level. The stealer is advertised on clearnet and darknet sites, with sales handled through a fully automated Telegram bot. For as little as $150 a month, anyone can purchase a subscription and deploy a sophisticated infostealer in minutes.

The malware itself is engineered to be modular and stealthy, with features that appeal to both low-sophistication operators and more capable threat actors. These features include in-memory .NET loaders, flexible command-and-control (C2) channels, and a plugin system for harvesting a wide variety of data from compromised systems.

Architecture and Obfuscation

The PURELOGS stub itself is obfuscated using a combination of ConfuserEx, .NET Reactor, and custom virtualization techniques. This results in mangled class and method names, virtualized control flow, and encrypted strings. This makes analysis more difficult but not impossible.

At its core, the stealer's execution flow is straightforward:

1. **Decrypt Configuration:** It starts by decrypting its C2 configuration from an embedded resource.
2. **Execute Modules:** It runs its various stealer modules according to the feature flags set in the configuration.
3. **Exfiltrate Data:** It packages and sends the stolen data back to the C2 server.

Configuration

The C2 configuration is stored as a Protobuf-serialized, XOR-encrypted blob embedded in the malware's resources. To access its configuration, the malware first applies a custom XOR decryption routine to the resource blob. This results in a Protobuf message that contains another layer of encryption: the actual configuration data is encrypted with 3DES.

The code snippet below shows the final 3DES decryption routine. It takes the encrypted data and a Base64-encoded string (the key) as input.

First, it decodes the Base64 string and computes its MD5 hash to derive the actual 3DES key.
Then, it initializes a *TripleDESCryptoServiceProvider* with the derived key and sets the mode to Electronic Codebook (ECB).
Finally, it calls *CreateDecryptor().TransformFinalBlock()* to decrypt the data.

Once decrypted, the configuration reveals the following:

- **C2 Server Detail**s**:** The IP address and port number for command-and-control communication.
- **3DES Encryption Key:** A Base64-encoded key used to encrypt all outgoing data.
- **Build ID:** A unique identifier for the malware campaign.
- **Feature Flags:** A series of boolean values that enable or disable individual stealer modules.

Modules

As mentioned earlier, PURELOGS is modular, meaning it includes numerous modules, each targeting different applications and credentials. Given the large number of modules available, not all of them are examined in detail here. Instead, the focus is on the two most important modules. A complete list is provided at the end.

Chromium: DPAPI be gone

The stealer's primary data harvesting module targets browsers built on the open-source project, which uses the Blink rendering engine. This allows it to attack a wide range of popular browsers like *Google Chrome, Microsoft Edge, and Opera* with a single codebase.

PURELOGS starts by reading the Local State file to find the encrypted master key. This key is protected by the Windows Data Protection API (DPAPI). However, DPAPI's protection is context-dependent and offers no real defense in this scenario. Since the malware is executing within the victim's user session, a call to the native CryptUnprotectData function is sufficient for the operating system to transparently decrypt the master key. This renders the protection useless against an attacker who has already achieved code execution.

Read about alternative Chromium credential theft techniques in our previous blog article on The ClickFix Deception.

With the plaintext AES-256-GCM master key in hand, the stealer accesses the SQLite databases, Login Data, Cookies, and Web Data, to decrypt and extract stored credentials, session cookies, autofill information, history and credit cards .
Finally, all the harvested information is collected, serialized, and packed into a Protobuf data structure, ready for exfiltration.

Crypto-Currency: Cash Grab

The malware employs a dual-pronged approach to stealing cryptocurrency, targeting both traditional desktop wallet applications and modern browser-based wallet extensions.
For desktop wallets, the WalletStealer class maintains an extensive hardcoded list of over 30 wallet applications including Bitcoin Core, Electrum, Exodus, and Monero. It systematically checks the default installation paths (typically within %APPDATA%) and registry keys for these applications. If a wallet file (like wallet.dat or default_wallet) is found, it is queued for exfiltration.

However, the more potent threat lies in its targeting of browser extensions, implemented within the ChromiumBrowserStealer module. This component targets over 70 different browser-based Web3 wallets, such as *MetaMask*, *Phantom*, *Trust Wallet*, and *Binance Chain Wallet*. It works by enumerating the Local Extension Settings directory of compromised browsers. It compares directory names against a dictionary of known wallet extension IDs (e.g., nkbihfbeogaeaoehlefnkodbefgpgknn for MetaMask). When a match is found, the malware compresses the entire extension data folder, which contains the encrypted seed phrases and private keys, into a ZIP archive. Crucially, because these extensions often use the browser's storage mechanisms, the attacker can potentially decrypt this data using the same master key stolen from the browser profile.

Remaining Modules

The full list of targeted applications and services includes:

Firefox / Gecko Browsers, Opera Browser, Yandex Browser, FileZilla, WinSCP, Outlook, Thunderbird, Foxmail, Mailbird, MailMaster, Telegram, Signal, Pidgin, Steam, OpenVPN, ProtonVPN, Ngrok, OBS Studio, Internet Download Manager (IDM)

Conclusion

PURELOGS demonstrates how commodity malware has adapted to the modern threat landscape, not through groundbreaking innovation, but through practical evasion layered at every stage.
The key insight here isn't sophistication. It's volume economics. These campaigns don't target specific industries or high-value entities. They're spray-and-pray operations where success is measured by the number of infections, not the impact per victim. For $150 per month, threat actors can deploy this tool and cast a wide net across thousands of potential targets. More infections mean more stolen credentials, more crypto wallets, more data to monetize.

This creates an interesting dynamic: the attacks don't need cutting-edge EDR bypasses because the targets largely don't have EDR. Home users, small businesses, and freelancers are operating with Windows Defender and basic antivirus at best. The four-stage delivery chain isn't overkill, it's right-sized for that threat model. It defeats consumer-grade defenses while remaining cheap and scalable.

But here's what matters for organizations: these campaigns don't discriminate. Corporate employees working from home, contractors on personal devices, and partners in your supply chain are all in the blast radius. Once credentials get compromised, they get tested against corporate VPNs, cloud services, and SaaS platforms. An infostealer hitting a remote employee's personal laptop can quickly become an enterprise security incident. The defense isn't complex. Memory-based execution leaves behavioral traces. Process hollowing creates anomalies. Reflection abuse generates telemetry. But you need visibility to see it, and you need monitoring to act on it.

The malware may be cheap, but the risk isn't.

The only system which is truly secure is one which is switched off and unplugged locked in a titanium lined safe, buried in a concrete bunker, and is surrounded by nerve gas and very highly paid armed guards. Even then, I wouldn't stake my life on it.
- Gene Spafford, Director, Computer Operations, Audit, and Security Technology (COAST) Project, Purdue University

IOC’s

IOC	Type	Name
c3857a086bdac485a5e65fc88828cb0c4c831be7a1f63e2dab32a47f97b36289	SHA256	PO 4501054441 Luan Pharm.js
c208d8d0493c60f14172acb4549dcb394d2b92d30bcae4880e66df3c3a7100e4	SHA256	Microsoft.Win32.TaskSchedule
3050a5206d0847d5cfa16e79944ce348db688294e311db4d7b6045ffbe337450	SHA256	Qgwwal.exe
bb723217f9c2932116c9e1313d558a7baddb921886eaa3beca95f7b3c5b848b0	SHA256	ClassLibrary4.dll
08a5d0d8ec398acc707bb26cb3d8ee2187f8c33a3cbdee641262cfc3aed1e91d	SHA256	optimized_MSI.png
hxxps[://]archive[.]org/download/optimized_msi_20250904/optimized_MSI[.]png	url	
hxxps[://]ia902909[.]us[.]archive[.]org/16/items/optimized_msi_20250904/optimized_MSI[.]png	url	
hxxp[://]lineclearexpress[.]wuaze[.]com/arquivo_20250908023227[.]txt	url	
185.27.134.206	IP	
45.137.70.55:5888	IP:Port	

References

https://www.ibm.com/think/x-force/dcrat-presence-growing-in-latin-america
https://github.com/SychicBoy/NETReactorSlayer
https://hackforums.net/showthread.php?tid=5926879
https://www.netresec.com/?page=Blog&month=2025-07&post=PureLogs-Forensics

---