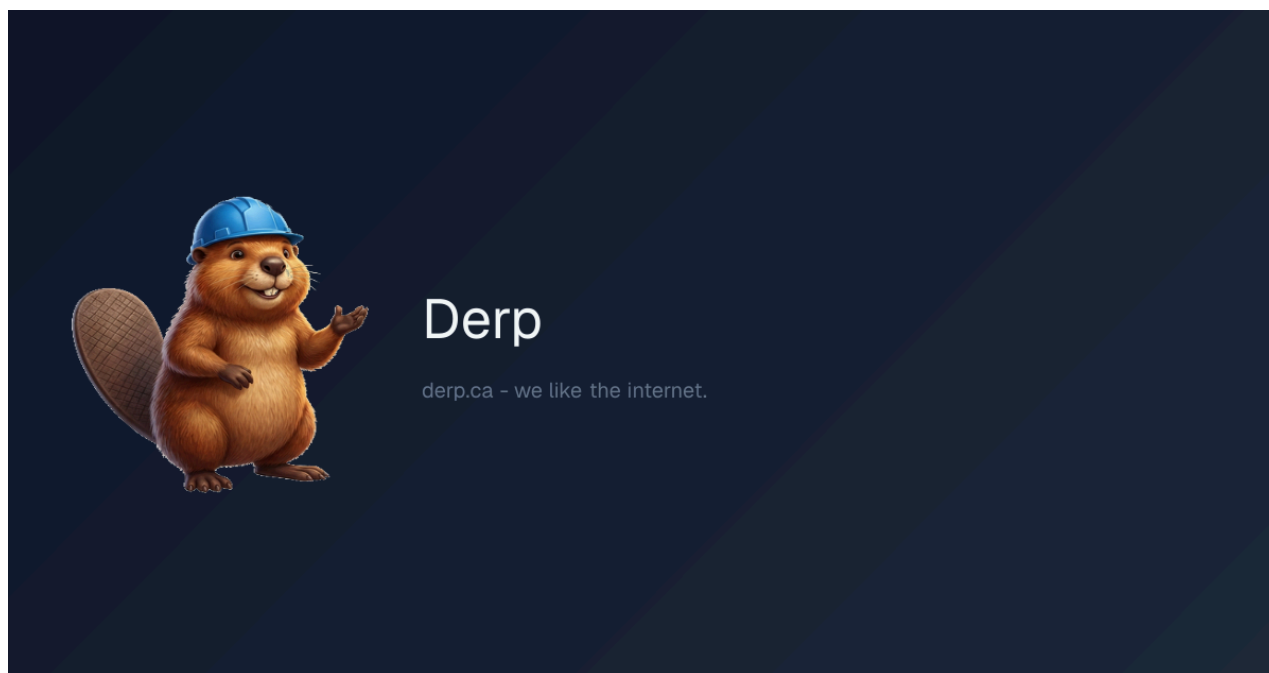


Breaking Aura: five obfuscation layers & hates sandboxes

 derp.ca/research/aura-stealer-reverse-engineering

Kirk

March 28, 2026



malware stealer reverse-engineering obfuscation encryption

► On this page

Aura is a native C++ info-stealer sold as a service since July 2025. It appeared as a post-Lumma replacement, advertised to the same buyer base after the Lumma takedown. 104 unique samples have passed through [Triage](#) since November 2025, all scoring 10/10 and classified as `aura_stealer`. Every submission ends in a crash. VirusTotal detection sits at 53/75. The anti-sandbox code runs in global constructors before `WinMain`, and most public sandboxes capture only the crash.

The binary ships with five layers of code obfuscation: a patched ImageBase that breaks static IAT analysis, a Heaven's Gate shellcode block that transitions to 64-bit mode for Chrome credential theft, a generic 32-to-64-bit trampoline, control flow flattening over an FNV-1a API hash table, and per-build AES-256-CBC config encryption. On top of that, a separate SSE XOR layer encrypts every string argument passed to the resolved APIs. The stealing targets themselves are not in the binary at all. They are pushed from the C2 server at runtime.

We broke all five code layers through static analysis, then emulated the binary past the sandbox wall and onto a live C2 server. The transport encryption turned out to be AES-256-CBC with the key and IV prepended in plaintext to every message. TLS is the only wire protection. We replayed the client handshake against the one surviving C2 domain (`glider[.]cfd`), got a 200 response, and decrypted 42 KB of server-pushed configuration: 212 stealing tasks across 170 targets, including 99 Chromium browsers, 13 Gecko browsers, 40+ crypto wallets, VPN clients, password managers, and messaging apps.

Sample overview

Field	Value
SHA256	90a1fb5ef34cc6abee75e7b39166b3cbb97d5545496251ea69c4d4372aa4c3fe
Type	PE32 (GUI) Intel 80386, MSVC 14.0
Size	725,472 bytes
Compiled	2025-10-25
Version	1.5.2
Build ID	a05e095d-3b2d-4912-9174-a1ce01e86dda
imphash	1cb7010fa29c1205e95faaaf21e5a21c
VT	53/75 (AVG/Avast: Win32:AURAStealer-D, BitDefender: Gen:Variant.Fragtor.907392)
Triage	10/10 aura_stealer, crashes in sandbox (WerFault.exe)
First seen	2025-11-20 (VirusTotal)
C2 domains	magicupdate[.]cfd, searchagent[.]cfd, mscloud[.]cfd

The PE header has ASLR, DEP, and CFG disabled. The Rich header is stripped. No resources, no TLS callbacks, no overlay. The builder strips metadata from the output binary and patches the ImageBase post-compilation from the original `0x750000` to a random value. All code references use the compile-time base; the `.reloc` section fixes them up at load.

Five samples were analysed across two versions (v1.5.1 and v1.5.2), including one UPX-packed build.

Code obfuscation: five layers

Layer 1: static IAT

The import table has 105 functions across three DLLs (KERNEL32, ADVAPI32, gdiplus). All are called via `FF 15` (call [mem32]) using `0x750000`-based addresses. The builder's ImageBase patching means these calls only resolve after `.reloc` fixups apply. The original base can be recovered by scanning `FF 15` call targets against

candidate base addresses until the IAT entries align.

Layer 2: Heaven's Gate x64 shellcode

A 3.5 KB block of x64 code sits in the `.data` section. It executes through a `push 0x33; retf` transition that switches the CPU from 32-bit to 64-bit mode (CS selector 0x33). This is the Heaven's Gate technique.

The shellcode walks the PEB in 64-bit mode to find `kernel32.dll` (the third entry in `InMemoryOrderModuleList`, not `ntdll`). It resolves `GetProcAddress` and `LoadLibraryA` by walking the PE export table and computing DJB2 hashes (seed `0x1505`) with a per-build XOR key.

A 32-byte XOR key, stored in register assignments to an `[r15]` buffer, decrypts 16 API name strings via SSE `xorps`. The key registers vary by version: `rbp/rdi` in v1.5.2, `r14/rsi` in v1.5.1. The decrypted APIs are file I/O (8), COM/WMI (6), and OLE automation (2). Their purpose: read Chrome's App-Bound encryption key (written to a file with an "APPB" magic header) and fingerprint the machine via WMI queries.

The WMI queries are dormant in all v1.5.x builds. The control value is the string "BROWSERTYPE", which causes the WMI code path to be skipped.

Layer 3: Heaven's Gate trampoline

A 137-byte trampoline in `.rdata` acts as a generic 32-to-64-bit function caller. It copies itself to `VirtualAlloc`'d executable memory during init. The calling sequence:

```
push 0x33          ; x64 code segment
call $+5
add [esp], 5
retf               ; far return -> switches to 64-bit mode
```

In 64-bit mode, it loads arguments from the 32-bit stack into the x64 calling convention (`rcx/rdx/r8/r9` + stack), calls the target function, stores the result, and returns to 32-bit mode.

Layer 4: control flow flattening + FNV-1a hash table

Two CFF variants protect most API calls:

Pattern A (multi-level chain): 3-4 pointer dereferences through 17 pre-initialised `.data` base pointers, summed to produce a computed jump target.

Pattern B (constant + pointer): a compile-time constant added to a `.data` value produces the target directly.

Both patterns resolve to the same FNV-1a hash table wrapper at `0x7F4D68`. The hash table uses seed `0x811C9DC5` and prime `0x01000193`. It is populated during init via `GetModuleHandleA("ntdll.dll")` and `LdrGetProcedureAddress`, with API addresses stored XOR-encrypted (`[entry+0xC] ^ [entry+0x10]`).

Nine lookup keys are identified. Each is a builder-assigned compile-time constant, XOR-obfuscated per call site. A 99-call function pointer at `0x903B8` points to a `ret 0` stub, a no-op placeholder disabled in v1.5.x.

Layer 5: config encryption (AES-256-CBC)

The build config is encrypted in the `.data` section after the Heaven's Gate shellcode. Layout:

```
[32-byte AES key][16-byte IV][0-8 byte gap][ciphertext]
```

Each build has a unique key and IV generated by the builder panel. The gap between key/IV and ciphertext varies per build (0, 7, or 8 bytes observed). Decrypted, the config is JSON:

```
{
  "conf": {
    "hosts": ["https://magicupdate.cfd", "https://searchagent.cfd", "https://mscloud.cfd"],
    "anti_vm": false,
    "anti_dbg": true,
    "self_del": true,
    "run_delay": 0,
    "useragents": [""],
    "human_check": false
  },
  "build": {
    "ver": "1.5.2",
    "build_id": "a05e095d-3b2d-4912-9174-a1ce01e86dda"
  }
}
```

The encrypted config can be located in any build by scanning for the PEB walk signature (the Heaven's Gate anchor) and reading the key, IV, and ciphertext relative to that anchor. No hardcoded offsets are needed.

Data obfuscation: XOR string encryption

A separate obfuscation layer protects the string arguments passed to the resolved APIs: credential paths, browser profile names, registry keys, process blacklists. The strings are encrypted inline via SSE `xorps` using per-block XOR keys derived from `.data` base pointers.

252 XOR sites across four function blocks produce 235+ unique decoded strings. The decoded content includes:

Category	Count	Examples
Chrome file targets	8	Local State, Login Data, Network\Cookies, History, Bookmarks

Category	Count	Examples
Firefox file targets	16	key4.db, logins.json, cookies.sqlite, places.sqlite, cert9.db
Chrome ABE strings	7	os_crypt, encrypted_key, app_bound_encrypted_key
Process blacklist	37	ollydbg.exe, ida64.exe, x64dbg.exe, Wireshark.exe, windbg.exe, Fiddler.exe
DLL/driver blacklist	23	apimonitor-x86.dll, apimonitor-psn-x86.sys, sbiedll.dll, cuckoomon.dll
Sandbox usernames	2	JohnDoe, HAL9TH
CIS locale exclusion	3	AM, BY, GE
C2 protocol strings	~30	/api/conf, /api/send, multipart/form-data, pow=
Self-delete commands	7	cmd.exe, taskkill, del /f /, fsutil f, timeout
DLL inventory	14	kernel32.dll, ntdll.dll, winhttp.dll, crypt32.dll, ws2_32.dll
System fingerprint	155	AURA self-ID, HWID, [System Info], [Hardware], IsWow64Process

The system fingerprint block reveals the full exfil report format. The binary builds a structured text document with the self-identification marker **AURA**, followed by labelled sections:

```
AURA
HWID: {MachineGuid}
Launched at: {%Y-%m-%d %H:%M:%S UTC}
Location: {binary path}
Run as Admin: {yes/no}
User in Admins group: {yes/no}
[System Info]
Architecture: {x64/x32}
Language: {locale}
Keyboard Layouts: {list}
Time Zone: {tz}
Computer Name: {NetBIOS}
User Name: {user}
Screen resolution: {WxH}
OS Name: {ProductName}
Edition: {EditionID}
Version: {CurrentBuild}
Build Number: {BuildLab}
[Hardware]
CPU: {ProcessorNameString}
RAM: {total} MB
GPUs: {display devices}
[Processes List]
{running processes}
[Installed Software]
{installed programs}
```

OS version data comes from the registry at `SOFTWARE\Microsoft\Windows NT\CurrentVersion` (ProductName, EditionID, CurrentBuild, BuildLab, CSDVersion, InstallDate) and `HARDWARE\DESCRIPTION\System\CentralProcessor\0` (ProcessorNameString). The binary also calls `RtlGetVersion` from `ntdll` and `IsWow64Process` for architecture detection.

Hardcoded Chrome targets

These file names are in the binary, not server-pushed. They target specific Chrome data files within each profile directory:

File	Purpose
<code>/MasterKey.bin</code>	App-Bound master encryption key (via Heaven's Gate APPBKEYREAD)
<code>/Local State</code>	Contains <code>encrypted_key</code> for cookie/password decryption
<code>Login Data</code>	SQLite DB with saved passwords
<code>Web Data</code>	SQLite DB with autofill and saved credit cards

File	Purpose
Network\\Cookies	SQLite DB with cookies (Chromium 96+ path)
History	SQLite DB with browsing history
Bookmarks	JSON with saved bookmarks
Last Version	Chrome version tracking

The Chrome App-Bound Key theft uses the Heaven's Gate shellcode to read Chrome's App-Bound encryption key in 64-bit mode and write it to a file with an **APPB** magic header. The key fields **os_crypt**, **encrypted_key**, and **app_bound_encrypted_key** are parsed from **Local State** to locate the encrypted credential material.

Runtime-resolved API surface

The binary resolves 100+ APIs at runtime through the FNV-1a hash table and CFF dispatch. The full surface, from **.rdata** strings interleaved with nlohmann JSON error messages:

Category	APIs
Anti-analysis	NtQueryInformationProcess, NtCreateDebugObject, NtQueryObject, NtGetContextThread, NtSetContextThread, GetTickCount
Network	WinHttpOpen/Connect/OpenRequest/SendRequest/ReceiveResponse/ReadData/WriteData + 10 more WinHTTP, WSStartup, getaddrinfo, connect, htons
Credential theft	CryptUnprotectData, CryptBinaryToStringA, CredReadA, RegOpenKeyExW, RegEnumKeyExW, RegQueryValueExW
Screenshot	BitBlt, CreateCompatibleBitmap, GdiCreateBitmapFromHBITMAP, GdiSaveImageToStream, CreateStreamOnHGlobal
Clipboard	OpenClipboard, GetClipboardData, CloseClipboard
System enum	GetComputerNameW, GetUserNameW, GetSystemInfo, EnumDisplayDevicesW, Process32FirstW/NextW, CreateToolhelp32Snapshot
Process control	CreateProcessW, OpenProcess, TerminateProcess, NtResumeProcess, NtSuspendProcess, ShellExecuteExW
File system	NtCreateFile, NtOpenFile, NtReadFile, NtQueryDirectoryFile, ExpandEnvironmentStringsW

The stealing targets (credential paths, wallet extension IDs, browser profile directories) are not in the binary. They are pushed from the C2 server at runtime. We confirmed this by searching all six encrypted blocks against 916 known target strings, including all 274 Avast-captured extension IDs. None matched. The targets come from the server.

Dynamic analysis: breaking the sandbox wall

Every Aura sample crashes in the Triage sandbox. The anti-sandbox code runs in global constructors (`_initterm_e`), not in `WinMain`. The `_initterm_e` table at `0x7E0408` contains eight function pointers. One or more of them contain:

- `MapFileAndCheckSumW` binary integrity verification
- Software breakpoint detection on return addresses
- Anti-debug checks (configurable via `anti_dbg` config flag)

These fire before `WinMain` executes. Patching the config is not enough. The constructors must be survived.

We emulated both samples using a custom PE emulator. Both ran to a clean `ExitProcess(0)`, producing ~16.5 million instructions and ~2,800 API calls each. The emulator walked through all eight global constructors without issue.

The runtime behaviour sequence:

1. MSVC CRT init (`_initterm_e`, eight global constructors)
2. `MapFileAndCheckSumW` binary integrity check
3. `Sleep(968)` post-checksum delay
4. `GetUserNameW` (sandbox username check against `JohnDoe/HAL9TH`)
5. AES-256-CBC config decryption
6. `NtCreateDebugObject` + `NtQueryObject` anti-debug check
7. Process blacklist scan via `CreateToolhelp32Snapshot` + `Process32FirstW/Process32NextW`
8. `RegOpenKeyExW("SOFTWARE\\Microsoft\\Cryptography")` + `RegQueryValueExW("MachineGuid")` for HWID
9. `CreateMutexA("Global\\{build_token}")` single-instance guard
10. DNS connectivity check: `ws2_32.socket` + `connect` to port 53
11. Heartbeat loop: six WinHTTP GET requests to `/api/live` (three domains, two attempts each)
12. Build `/api/conf` request payload (multipart with encrypted build ID)
13. `CorExitProcess(0)`

The mutex token is per-build but is not the build UUID. Examples: `LI5joIjAwOpBWBvWHUK2Y0U5fz20` (build `a05e095d`), `1yilNwHd5QDEvAUM0GCI60c` (build `a068cff0`).

C2 protocol

Infrastructure status (as of 2026-03-28)

Domain	Status
<code>glider[.]cfd</code>	Live. Backend active, Cloudflare-proxied.
<code>mushub[.]cfd</code>	Cloudflare phishing interstitial.
<code>searchservice[.]cfd</code>	Cloudflare phishing interstitial.
<code>sakuratea[.]cfd</code>	Cloudflare malware interstitial.
<code>magicupdate[.]cfd</code>	DNS dead.
<code>searchagent[.]cfd</code>	DNS dead.
<code>mscloud[.]cfd</code>	DNS dead.
All <code>.shop</code> domains	DNS dead.

`glider[.]cfd` is the only domain with a functioning backend.

Heartbeat: `/api/live`

```
GET /api/live HTTP/2
Host: {c2_domain}
Connection: Close
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/127.0.0.0 Safari/537.36
```

The server responds with the string `true` (4 bytes). The binary iterates the hosts array from the config, trying each domain in order. Timeout is 0x55730 ms (~350 seconds) for resolve, connect, send, and receive.

Config fetch: `/api/conf`

POST-only (Allow: `POST` on GET). The error page on invalid requests uses the Tabler CSS framework and renders a Russian-language "Домой" (Home) button.

The binary sends a multipart POST with a single field named `data` containing a base64-encoded envelope. We probed the live server and got a 200 response with the encrypted config. Garbage data returns 500. Missing or malformed requests return 302.

Transport encryption

The transport envelope is identical for client-to-server and server-to-client:

```
[32-byte AES-256 key][16-byte IV][8 random bytes][AES-256-CBC ciphertext, PKCS7 padded]
```

Both sides generate a fresh AES-256 key and IV per message, prepend them in plaintext, and encrypt the payload. TLS is the only wire protection. The 8 random bytes between the IV and ciphertext come from `SystemFunction036` (RtlGenRandom) via MT19937.

We confirmed this structure by comparing the base64 blobs from two builds. Both encode to 120 bytes. The first 72 bytes are identical between builds (same key and IV material, same JSON prefix). The builds diverge at byte 72 where the UUID differs. Decrypting `blob[56:120]` with key `blob[0:32]` and IV `blob[32:48]` in AES-256-CBC yields:

```
{"build":{"build_id":"a05e095d-3b2d-4912-9174-a1ce01e86dda"}}
```

Valid PKCS7 padding (pad=3). The server response follows the same envelope. The server's AES key and IV are in the first 48 bytes of the response.

Decrypted server config

We decrypted the `/api/conf` response from `glider[.]cfd` using the server's key and IV. The result is 42,658 bytes of JSON (config version 1.1.0) defining 212 stealing tasks across 172 unique targets.

Task breakdown

Type	Count	Description
<code>fg-std</code>	98	File grabbers. Steal files matching glob patterns from target directories.
<code>chromium</code>	99	Chromium browser data extraction. Credentials, cookies, history, bookmarks.
<code>gecko</code>	13	Gecko/Firefox browser data extraction.

Type	Count	Description
system-info	1	Machine fingerprint collection.
screenshot	1	Desktop screenshot capture.

File grabber targets (98 tasks)

The file grabber tasks target application data directories under %appdata%, %localappdata%, and %programfiles%. Each task specifies a name, path, file mask pattern, maximum depth, maximum file size, and archive entry path.

Crypto wallets (30+): Anoncoin, Armory, Atomic, BBQCoin, Binance, Bitcoin, Blockstream Green, Bytecoin, Coinomi, Daedalus, DashCore, Devcoin, Digitalcoin, Dogecoin, ElectronCash, Electrum, Electrum-G, Electrum-LTC, Ethereum, Exodus, Florincoin, Franko, Freicoi, GoldCoinGLD, Guarda, IOCoin, Infinitecoin, Ixcoin, Jaxx, Ledger, Litecoin, Megacoin, Mincoin, MultiDoge, Namecoin, Primecoin, Raven, Terracoin, Wasabi Wallet, YACoin, Zcash.

Applications: FileZilla (FTP credentials), AnyDesk (remote access configs), KeePass (password database files), Discord (LevelDB token storage), Telegram (tdata session files), Steam (config + ssfn files), Uplay, OpenVPN/NordVPN/ProtonVPN (VPN profiles), Authy Desktop (2FA tokens), Pidgin/Psi+/qTox (messaging).

Generic: Two tasks scan %appdata% and %localappdata% with broad masks for "important files."

Browser targets

Chromium (99 browsers): Google Chrome (8 variants including Beta, Canary, Dev, SxS, x86), Microsoft Edge, Brave, Opera (Stable, GX, Crypto Developer, Neon), Vivaldi, Yandex (6 variants), 360 (5 variants), CentBrowser, Comodo/Comodo Dragon, CryptoTab, Epic Privacy, Iridium, Maxthon, Naver Whale, SRWare Iron, Slimjet, Torch, UCBrowser, and 70 others.

Gecko (13 browsers): Firefox, Thunderbird, Waterfox (Classic + current), Pale Moon, SeaMonkey, Basilisk, Cyberfox, IceDragon, K-Meleon, SlimBrowser, BitTube, BlackHawk.

Anti-analysis

Process blacklist (37 entries)

ollydbg.exe	ollyice.exe	tcpview.exe	autoruns.exe
autorunsc.exe	filemon.exe	procmon.exe	regmon.exe
proccxp.exe	idaq.exe	idaq64.exe	ida64.exe
radare2.exe	ImmunityDebugger	Wireshark.exe	dumpcap.exe
HookExplorer	ImportREC.exe	PETools.exe	LordPE.exe
SysInspector	proc_analyzer	sysAnalyzer.exe	sniff_hit.exe
windbg.exe	joeboxcontrol	joeboxserver	ResourceHacker
x32dbg.exe	x64dbg.exe	Fiddler.exe	httpdebugger
cheatengine-i386		cheatengine-x86_64	
cheatengine-x86_64-SSE4-AVX2		frida-helper-32	frida-helper-64

DLL/driver blacklist (23 entries)

npf.sys	winpcap.dll	npcap.dll	wpcap.dll
packet.dll	windivert.dll	detours.dll	apimonitor-x86.dll
apimonitor-drv-x86	apimonitor-psn-x86	vehdebug-x86_64	sbiedll.dll
dbghelp.dll	dbgeng.dll	api_log.dll	dir_watch.dll
pstorec.dll	cuckoomon.dll	wpespy.dll	printfhelp.dll
cheatengine-i386	frida-helper-32	frida-helper-64	

Sandbox evasion

Username comparison against **JohnDoe** and **HAL9TH**. Locale check excludes CIS country codes **AM**, **BY**, **GE**. Binary integrity verification via **MapFileAndCheckSumW**. All checks run in global constructors before **WinMain**.

Human verification gate

When **human_check** is true in the config, the binary creates a window titled with the first 8 characters of the multipart boundary and displays "Enter {prefix} to continue" with an input field and Ok/Exit buttons. The prefix is derived from the randomly generated boundary string. The binary blocks on **GetMessageW** until the user enters the correct value. A second build (a068cff0) had **human_check: true** and **anti_vm: true** where the primary sample had both set to false.

Self-delete

When **self_del** is true in the config, the binary wipes and deletes itself after exfiltration. The reconstructed command from decoded string fragments:

```
cmd.exe /c taskkill /f /im <process> & fsutil file setZeroData offset=0 length=<size> <path> & del /f /q <path> & timeout /t 5
```

`fsutil file setZeroData` overwrites the file contents with zeros before deletion, defeating forensic file recovery.

IOC summary

Network

IOC	Type	Notes
<code>glider[.]cfd</code>	C2 domain	Live as of 2026-03-28
<code>mushub[.]cfd</code>	C2 domain	Cloudflare-flagged
<code>searchservice[.]cfd</code>	C2 domain	Cloudflare-flagged
<code>sakuratea[.]cfd</code>	C2 domain	Cloudflare-flagged
<code>magicupdate[.]cfd</code>	C2 domain	DNS dead
<code>searchagent[.]cfd</code>	C2 domain	DNS dead
<code>mscloud[.]cfd</code>	C2 domain	DNS dead
<code>opencamping[.]shop</code>	C2 domain	DNS dead
<code>gamedb[.]shop</code>	C2 domain	DNS dead
<code>unknowntool[.]shop</code>	C2 domain	DNS dead
<code>browsertools[.]shop</code>	C2 domain	DNS dead
<code>armydevice[.]shop</code>	C2 domain	DNS dead
<code>glossmagazine[.]shop</code>	C2 domain	DNS dead

Host

IOC	Type
<code>Global\\{random_token}</code>	Mutex (per-build)

IOC	Type
<code>SOFTWARE\\Microsoft\\Cryptography\\MachineGuid</code>	Registry read (HWID)

Behavioural

Behaviour	Detail
User-Agent	<code>Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0 Safari/537.36</code>
Heartbeat	<code>GET /api/live</code>
Config fetch	<code>POST /api/conf</code> (multipart, <code>data</code> field)
Data exfil	<code>POST /api/send</code>
Self-delete	<code>fsutil file setZeroData + del</code>
Integrity check	<code>MapFileAndCheckSumW</code> in global constructor

Crypto constants

Constant	Value	Purpose
FNV-1a seed	<code>0x811C9DC5</code>	API hash table
FNV-1a prime	<code>0x01000193</code>	API hash table
DJB2 seed	<code>0x1505</code>	Heaven's Gate export resolution
MT19937 init	<code>0x6C078965</code>	Transport key generation
AES S-box VA	<code>0x7D9744</code>	Custom AES-256 implementation

The transport encryption sends the AES key and IV in plaintext with every message. The only protection is TLS. Any entity with visibility into the TLS session (the Cloudflare proxy, for example) can decrypt both the build registration and the server-pushed config.

Kirk

I like the internet. Want to get in touch? ·