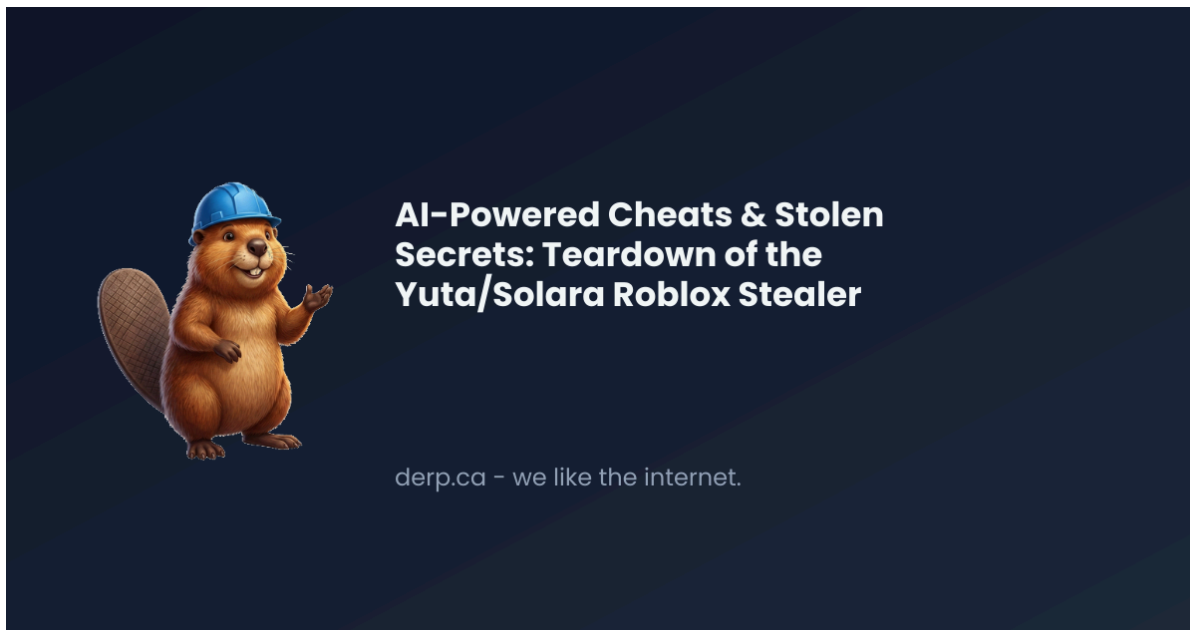


AI-Powered Cheats & Stolen Secrets: Teardown of the Yuta/Solara Roblox Stealer

 derp.ca/research/yuta-solara-roblox-python-rat

Kirk

June 4, 2026



Roblox exploit developers and game cheat users operate in a highly adversarial landscape where security alerts are the baseline norm. Because Roblox executors must hook game processes, inject dynamic link libraries (DLLs), and patch system memory, Windows Defender and third-party antivirus software frequently flag these tools as malicious. This persistent wall of false-positives creates a unique vulnerability: exploiters are primed to ignore security warnings, manually disable their antivirus software, and whitelist entire installation paths. Threat actors exploit this brand recognition and "security fatigue" to distribute highly evasive trojanized payloads disguised as legitimate execution tools.

What starts as a script kid's quest to cheat in Roblox ends in a masterclass of evasion. In this campaign, the initial stage is a .NET 4.8 Windows Presentation Foundation (WPF) application that presents a functional executor GUI complete with live DeepSeek AI integration. However, before the user interface is even displayed, a secondary payload is silently staged and executed.

This payload is a PyInstaller-compiled Python 3.12 binary containing a sophisticated port of the **Glove Stealer** family, complete with active memory patching, persistent C2 communication, and a multi-tiered bypass matrix targeting Google Chrome's App-Bound Encryption (ABE). The inspiration for sample acquisition was based on a [Joe Sandbox Analysis Report \(opens in new tab\)](#) and related community flags, after which we obtained the binary and performed a complete local teardown of both execution stages to reconstruct their capabilities.

(Methodology Note: The findings in this report are based on deep static analysis, PyInstaller extraction, and manual bytecode disassembly via xdis/pydisasm. Because no live detonation trace was utilized, the runtime success of specific mechanisms - such as the UAC bypass, Defender tampering, and WMI persistence - is inferred directly from the recovered execution logic and orchestration constants.)

Campaign Indicators

The campaign employs a multi-stage execution model, relying on remote dead-drop configuration points to resolve active payload hosts, exfiltration webhooks, and Discord command-and-control (C2) servers.

Indicator Type	Value / Path	Details
Stage 1 Executable Hash	7ec70af57c96c75da3553bbdf333a8486be1c6ab9e07f6242443ec8db0318a5e	Trojanized WPF UI (.NET 4.8 executable)
Stage 2 Payload Hash	a5538a6dd83446eafbe356dc98016cc2aa62092f524f4db47546fb1552aa53b1	PyInstaller-compiled Python 3.12 Stealer
XOR Remote Config	https://pastebin.com/raw/ptXPHhQS	Key: W3P!9kZ#mQ2xR@nL
Configuration Cache	%LOCALAPPDATA%\Microsoft\CLR_v4.0\cache_cfg.dat	Local cache storing base64-encoded loader config
SilentLoader Target Path	%APPDATA%\cache\winupdate.exe	Staging location for the executed Python payload
Webhook Resolver URL	https://pastebin.com/raw/a18wq6rG	Decrypted URL SHA256: c01fa643edea4d1aaa...
C2 Resolver URL	https://pastebin.com/raw/4ubCKBwD	Decrypted Token SHA256: ecb2531b1d2204...
Discord Bot RAT Channel	Channel ID: 1511075836818882591	Discord C2 interactive session management channel

Phase 1: The Lure and User Deception

To maximize infection rates, the threat actors invested heavily in creating a polished front-end. The outer wrapper utilizes Costura.Fody to dynamically load UI dependencies directly from memory, maintaining a small distribution footprint while providing genuine utility to keep the user engaged.

Dependency-Bound Execution & Resilience

While the GUI features legitimate script editing and ScriptBlox search capabilities, it operates as a dependency-bound, semi-functional lure. Actual Roblox injection relies on adjacent native binaries (SLaunch.exe and YutaCore.dll) and a local HTTP broker (<http://localhost:9912/>), none of which are embedded in the submitted sample.

Crucially, the malware is built for resilience. If `SLaunch.exe` is missing, the application gracefully degrades, displaying a customized French-language alert (`SLaunch.exe introuvable - demarrage...`) but loading the GUI anyway to keep the user occupied while the infection chain proceeds in the background.

Live DeepSeek AI & AIReconstructor Integrations

To further validate its legitimacy, the app embeds an "AI Assistant" powered by DeepSeek.

- **API Key Exposure:** The application requires users to enter their personal DeepSeek API key. This key is written directly to disk in plaintext at `%LOCALAPPDATA%\YutaAIKey.txt`, immediately exposing the user's premium LLM developer credentials to the malware's broad file-sweeping modules.
- **AIReconstructor:** This component parses local Roblox game files (`.rbxlx`), extracts embedded source code, and sends it to DeepSeek with a highly specific system prompt requesting deobfuscation and anti-tamper removal. The reconstructed source is written to `*_ai_reconstructed.rbxlx`.

Phase 2: Silent Staging and System Compromise

While the user interacts with the AI features and script menus, the background loader (`SilentLoader.RunAsync()`) executes asynchronously to download and launch the primary stealer.

Defender Exclusion Hooking

To ensure the subsequent payload runs unhindered, the loader immediately attempts to blind Windows Defender for its current directory via `UpdateSplash.EnsureWindowsDefenderExclusion()`.

1. It queries local preferences via PowerShell: `(Get-MpPreference).ExclusionPath -contains '<current_directory_path>'`
2. If not excluded, it executes an administrative invocation: `Add-MpPreference -ExclusionPath "<current_directory_path>"`
3. To trigger the required UAC prompt silently, the loader uses the `runas` verb within the `ProcessStartInfo` configuration. Relying on the fact that cheat executors inherently require admin rights, victims typically click "Yes" without second-guessing.

Staging Orchestration

1. **Config Retrieval:** The loader retrieves a remote config from Pastebin (`ptXPHhQS`) by decrypting the payload in memory using an XOR loop with the key `W3P!9kZ#mQ2xR@nL`.
2. **MediaFire Scraping:** The JSON config points to a MediaFire storage page. Because MediaFire uses dynamic, time-sensitive download tokens, the loader makes a GET request to the landing page and applies a regex filter to scrape the active direct-download URL from the DOM.
3. **Validation & Hiding:** The loader downloads the binary, verifies the MZ and PE\0\0 headers, and saves it to `%APPDATA%\cache\winupdate.exe`. It applies Hidden and System attributes to the directory and binary.
4. **Execution:** The loader starts the background process and caches the base64-encoded Pastebin config at `%LOCALAPPDATA%\Microsoft\CLR_v4.0\cache_cfg.dat` for version tracking.

Phase 3: Payload Orchestration & Defense Evasion

The Stage 2 binary (`winupdate.exe`) is packaged using PyInstaller, unpacking a compiled Python 3.12 runtime directory containing Python bytecode files (`.pyc`). Modern Python decompilers (`uncompyle6`, `decompyle3`) fail against Python 3.12 due to structural changes introduced in PEP 659 (Specializing Adaptive Interpreter). Our team bypassed

traditional decompilation entirely, relying on xdis and pydisasm to manually map magic numbers and reconstruct control flow.

Parallel Execution Flow

By reconstructing `stage2_stealer.dis`, we mapped the payload's precise automated collection sequence. Following C2 credential resolution and memory patching, the orchestrator triggers collection via thread pools:

1. A **2-worker thread pool** prioritizes `system_info.collect()` and `chrome_abe.recover_key()`.
2. A synchronous run of `browsers.collect(abe_key)` ensures decrypted browser cookies and tokens are immediately available for subsequent modules to reuse.
3. A massive **14-worker thread pool** concurrently executes the remaining modules (wallets, Discord, Telegram, password managers, note scanning).
4. Finally, it starts the Discord bot C2, builds the output tree, exfiltrates the ZIP, and enters a 60-second sleep loop to remain resident.

Anti-Analysis & Defense Evasion

- **Sandbox Evasion:** The `_is_sandbox()` method queries active processes using `psutil` against a blacklist (`idaq.exe`, `x64dbg.exe`, `procmon.exe`, `wireshark.exe`). If detected, the payload enters a 120-240 second sleep cycle and silently terminates. This function relies *entirely* on process-name detection, lacking traditional hardware, VM registry, or CPU timing checks.
- **Active Memory Patching:** It locates `amsi.dll` and overwrites the entry point of `AmsiOpenSession` with `b8 57 00 07 80 c3 (mov eax, 0x80070057; ret)`. This forces a return of `E_INVALIDARG`, tricking Windows into failing "open" and allowing execution. It simultaneously patches `ntdll.dll!EtwEventWrite` with `c3 (ret)` to blind system telemetry.
- **Background Exclusion Maintenance:** A background daemon thread periodically loops a PowerShell command (`Set-MpPreference -DisableRealtimeMonitoring $true...`) to ensure Defender remains crippled even if manually re-enabled by the user.
- **AppLocker Relocation:** To bypass execution policies, the binary checks its own path. If not running from a trusted directory, it relocates to `%WINDIR%\Temp`, `%WINDIR%\SysWOW64`, or `%ProgramFiles%\Common Files`. This relies on environments utilizing Microsoft AppLocker's default allow-rules for standard system directories. It then spawns the relocated instance in a new process tree and terminates the parent.

Phase 4: Targeted Harvesting & App-Bound Encryption Bypasses

This payload contains parsing logic targeting highly specific developer and gaming vaults, confirming it as an evolutionary Python port of the Glove Stealer family rather than a generic grabber.

Chrome App-Bound Encryption (ABE) Bypass Matrix

To defeat Chrome 127+'s App-Bound Encryption, the payload integrates a 5-tier fallback matrix mimicking techniques pioneered by Glove Stealer and public security researchers:

1. **Debugger Hardware Breakpoint:** Launches Chrome using the `DEBUG_PROCESS` flag, placing a hardware breakpoint (DR0) at the internal decryption routine inside `chrome.dll` to read the decrypted master key directly from the R15 register.
2. **IElevator COM Interface:** Instantiates the Chrome elevation COM service (`IElevator`) and calls `CoSetProxyBlanket` to impersonate the client, forcing the browser's high-privilege service to decrypt the blobs and hand back the plaintext.

3. **Double DPAPI Impersonation:** Enables SeDebugPrivilege, duplicates the SYSTEM token from lsass.exe, and executes double DPAPI calls to decrypt machine-level bound ABE keys.
4. **DLL Injection via Suspended Thread:** Spawns Chrome in a CREATE_SUSPENDED state, using NtCreateThreadEx to map and inject a specialized decryption helper DLL entirely within the trusted process memory space.
5. **UAC Registry Hijack:** If integrity limitations block strategies 2 or 3, it writes a payload launcher command to HKCU\Software\Classes\ms-settings\Shell\Open\command and calls fodhelper.exe or computerdefaults.exe to auto-elevate and retry the LSASS token impersonation.

The Glove Stealer Footprint

Rather than just exfiltrating encrypted blobs for offline C2 cracking, the payload prioritizes aggressive local data processing.

Category	Targets	File Patterns / Details
2FA Databases	Authy, WinAuth	sqlite3.db, LevelDB folders, *.xml configurations
Notes & Vaults	Obsidian, Joplin, Notion	Traverses vault directories, scans text headers
Wallets	MetaMask, Phantom, Trust Wallet, Exodus	LevelDB extensions folder, JSON configurations
Webmail & Payments	PayPal, Amazon, IMAP config	Traverses HTML directories, IMAP settings
Game Launchers	Steam, Epic, Riot, Minecraft	loginusers.vdf, Riot JWT tokens, Lunar Client profiles
Messaging & Comms	Telegram, Signal, Slack, Teams	Telegram tdata copy, Electron token extraction
VPN & FTP	OpenVPN, WireGuard, FileZilla	.ovpn, .conf, sitemanager.xml, WinSCP registry keys
Password Mgrs	Bitwarden, 1Password, KeePass	.kdbx databases, browser session cookies

- **Roblox Asset Harvesting:** Directly targeting its lure demographic, modules.roblox.pyc hunts for .ROBLOSECURITY cookies across browsers and LocalStorage\RobloxCookies.dat. It actively hits internal Roblox APIs (economy.roblox.com, trades.roblox.com) to enrich the stolen profile with Robux balances, Premium features, saved billing credentials, and inbound trade inventory.
- **Deep Note Scanning:** twofa_notes.scan_notes_directory traverses developer note-taking software (Joplin, Obsidian, Notion). It reads the first 4,096 bytes of text and environment files (.env, .md, .py) hunting for high-entropy API tokens, specifically OpenAI keys prefixing with sk-.
- **Local MetaMask Dictionary Attack:** It parses the MetaMask LevelDB vault and executes a local dictionary attack using a COMMON_PASSPHRASES list. It performs PBKDF2-HMAC-SHA256 derivation (10,000 iterations) and AES-GCM decryption. To prevent host freezing, the attack logic is capped at an 8-second timeout. *(Assessment: This is weak-password opportunism against copied vault material, not a cryptographic break of AES-GCM).*

- **Google Account Harvesting:** It extracts SAPISID and __Secure-3PAPISID cookies to forge a SAPISIDHASH authorization header, directly querying the live Gmail API (/gmail/v1/users/me/messages) to scrape the victim's inbox for cryptocurrency password resets and financial receipts.

Phase 5: Exfiltration, C2, and Persistence

In-Memory Compression & Chunking

Rather than writing temporary ZIP files to disk (which routinely triggers EDR file-write heuristics), `exfil.pyc` builds the ZIP archive entirely in memory. It enforces a strict 25MB upload limit (26,214,400 bytes); if exceeded, it slices the archive into sequential byte chunks. These chunks are transmitted to the AES-decrypted Discord Webhook URL (<https://pastebin.com/raw/a18wq6rG>), pausing for exactly 1 second between requests to defeat API rate limiting.

Discord Bot RAT Commands

Operating in parallel to exfiltration, the malware resolves a Discord Bot token and Channel ID (1511075836818882591) via AES-CBC decryption. It monitors the channel for interactive Remote Access Trojan (RAT) commands:

Command	Action / Method
webcam	Captures image from default camera using OpenCV / DirectShow and uploads to channel
shell	Spawns a hidden shell process, executes CMD or encoded PowerShell commands, and returns stdout
keylogger	Installs a low-level keyboard hook (WH_KEYBOARD_LL) via SetWindowsHookExW
trace	Launches native system trace (pktmon or netsh trace) to capture local interface packets

6-Layer Persistence Matrix

To guarantee survival across reboots, the stealer installs a highly noisy persistence matrix:

1. **Scheduled Task:** Creates `WindowsDefenderAgent` configured to run at user login with highest privileges.
2. **Registry Run Key:** Inserts a payload path under `HKCU\Software\Microsoft\Windows\CurrentVersion\Run`.
3. **Startup VBS:** Writes `%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\WindowsDefenderAgent.vbs` to execute the binary windowlessly.
4. **COM Hijack:** Hijacks a common CLSID to load the malicious payload upon targeted Windows shell operations.
5. **PowerShell Profile Append:** Appends execution calls directly into `%USERPROFILE%\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1`.
6. **WMI Event Subscriber:** Registers an `ActiveScriptEventConsumer` tied to a `Win32_LogonSession` event filter, ensuring execution via the WMI subsystem upon boot.

Prior Art & MITRE ATT&CK Mapping

To anchor our teardown with the foundational research and threat intel communities that pioneered these bypasses, we trace the following resources:

- **App-Bound Encryption:** Alexander Hagenah's COM elevation and thread hijacking repository ([xaitax/Chrome-App-Bound-Encryption-Decryption \(opens in new tab\)](#)).
- **ChromeKatz Breakpoints:** Hardware break hooking pioneered by Meckazin ([ChromeKatz \(opens in new tab\)](#)).
- **Glove Stealer Origins:** The original .NET variant of the Glove family was first analyzed by Gen Threat Labs ([Avast Decoded: Glove Stealer \(opens in new tab\)](#)).
- **Python 3.12 Structural Changes:** Decompilation failures are due to bytecode shifts detailed in [PEP 659 \(opens in new tab\)](#) and open issues under the [Decompyle++ repository \(opens in new tab\)](#).

ID	Technique	Mitigation / Detection Context
T1204.002	User Execution: Malicious File	Trojanized Roblox executor distributed in game exploit communities.
T1102.001	Web Service: Dead Drop Resolver	Utilizes Pastebin entries to host dynamic C2 configuration and webhook URLs.
T1105	Ingress Tool Transfer	Downloads the secondary payload executable from MediaFire storage.
T1562.001	Impair Defenses: Disable Tools	Memory patches AMSI/ETW functions and runs background Defender disabling commands.
T1548.002	Abuse Elevation Control: Bypass UAC	Registry hijack of ms-settings paired with fodhelper.exe for high integrity elevation.
T1547.001	Boot/Logon Autostart Execution	Installs Registry Run keys, Startup scripts, and Scheduled Tasks for persistence.
T1555.003	Credentials from Web Browsers	Employs multiple App-Bound Encryption bypass strategies to retrieve cookie and credential keys.
T1056.001	Input Capture: Keylogging	Installs low-level keyboard hook (WH_KEYBOARD_LL) for keylogging.
T1546.003	WMI Event Subscription	Registers WMI active script event consumer and filter for persistent execution.
T1041	Exfiltration Over C2 Channel	Transmits files over Discord Webhooks using 25MB chunked, in-memory ZIP archives.

Conclusion

This campaign represents a highly functional lure that provides real Luau exploit generation while silently staging an advanced credential harvester. By combining a Python-based port of the Glove Stealer blueprint with active browser COM interface ABE bypass techniques, the threat actors are capable of capturing high-value developer secrets that standard malware ignores, all controlled through a persistent Discord bot RAT.

Kirk

I like the internet. Want to get in touch?