

Don't Eat the ChocoPoCs: Trojanised PoCs Hit Researchers

sekoia.com/blog/dont-eat-the-chocopocs-how-vulnerability-researchers-were-repeatedly-targeted-by-trojanised-exploits

Sekoia

July 1, 2026



Key takeaways

- Targeted Supply Chain Attack: Vulnerability researchers and pentesters are being actively targeted via malicious Proof-of-Concept (PoC) repositories on GitHub that contain trojanised Python dependencies.
- ChocoPoC Malware: The infection delivers a stealthy, persistent Python Remote Access Trojan (RAT) known as "ChocoPoC," capable of exfiltrating files, harvesting browser credentials, and executing arbitrary commands on developer machines.

This blogpost is a collaboration between YesWeHack and Sekoia TDR, analysing an undocumented supply chain attack that targets vulnerability researchers and pentesters via lure CVE PoC repositories. Our analysis shows that this vector has already been used in malicious PoCs, seeking to compromise pentesting tools, since late 2025. Because the malware and its C2 infrastructure are still active, we strongly advise against running any of the PoCs code or installing the malicious packages.

In this article, we show how the urgency for vulnerability researchers to quickly produce scan modules for newly disclosed vulnerabilities was exploited by threat actors to target developer environments. The associated malware, dubbed ChocoPoC, is a fully functional Python RAT used to exfiltrate files, execute commands, and harvest secrets. At least 7 fake CVE PoC repositories have been identified, all employing the same *modus operandi*: a malicious PyPI dependency executes a compiled and obfuscated dropper that retrieves a Python RAT payload from a Mapbox dead drop resolver. ChocoPoC RAT is designed to be stealthy and persistent, using timestamping, file-lock mutexes, PEB walking, export hashing, and anti-debugging techniques to evade detection.

YesWeHack is a leading Offensive Security and Exposure Management platform. Its Vulnerability Intelligence team focuses on developing automated security checks for high-profile vulnerabilities, security misconfigurations and subdomain takeovers to provide an up-to-date view of the exploitable attack surface.

Sekoia is a French cybersecurity vendor offering an AI SOC platform that combines SIEM, SOAR and CTI capabilities to help organisations detect, investigate and respond to threats at scale. Its Threat Detection & Research (TDR) team produces a high-fidelity CTI by investigating both cybercrime and APT threats, turning intelligence into contextualised and actionable content.

A Colorful Forwarded PoC Story

As a continuous pentest platform, YesWeHack develops scanners for newly disclosed vulnerabilities. When a critical vulnerability affects a large attack surface - such as React2Shell (CVE-2025-55182), FortiWeb (CVE-2025-64446), or Ivanti Sentry (CVE-2026-10520) - every hour counts to produce a scan module before attackers weaponise and mass-exploit the vulnerability. Like most vulnerability scanning vendors, we leverage community-shared proof-of-concepts (PoCs) to build detection quickly. While everyone is warned to never run code from untrusted sources, urgency has a way of challenging good habits. ProjectDiscovery's Nuclei templates are an invaluable community resource, but GitHub issues and pull requests frequently reference PoCs from newly created or otherwise unknown repositories.

Running a PoC safely is harder than it sounds. Time pressure competes with caution, isolated labs require effort to maintain, and validating exploits against real-world configurations often means interacting with Internet-facing systems. Even when performed from sandboxed environments, these tests may expose sensitive engagement data or valuable intelligence about vulnerable targets... information that would be highly attractive to a malicious third party.

A few days ago, YesWeHack started contributing templates and PoCs to the [Nuclei community](#) to share ongoing vulnerability research and increase the visibility of the Vulnerability Intelligence team. Then, on Thursday, 25 June 2026, shortly after publishing a [Joomla JCE template](#), we received an intriguing GitHub notification:

SP Page Builder Unauthenticated RCE #16467

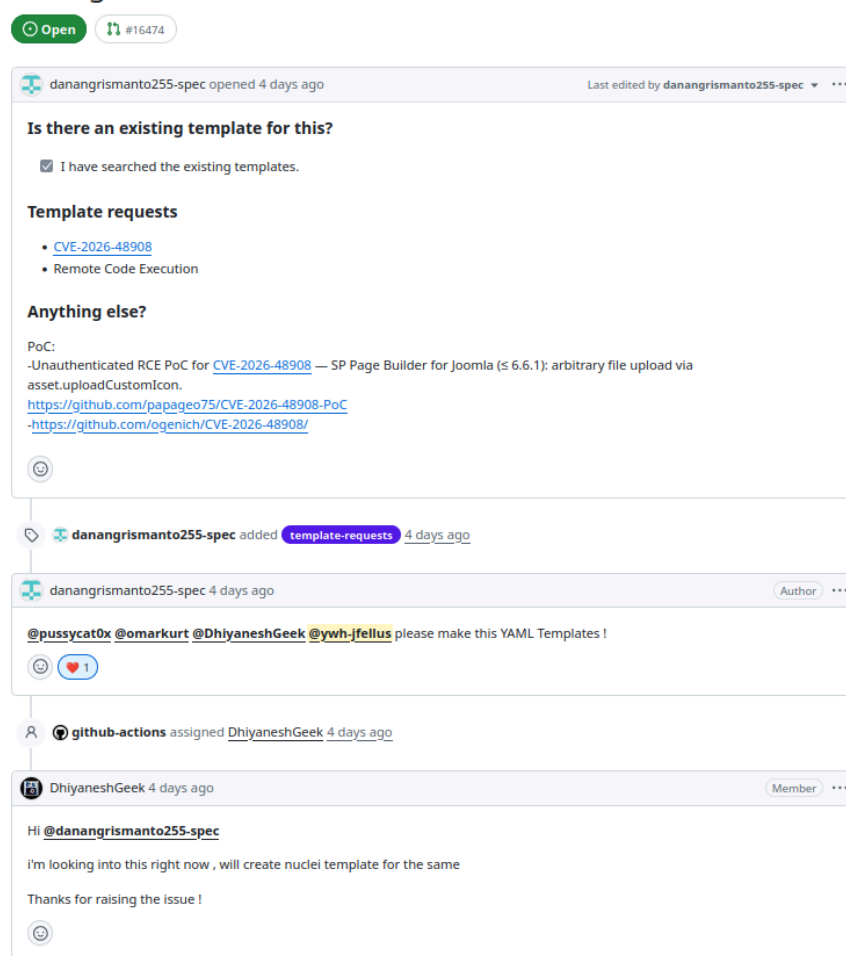
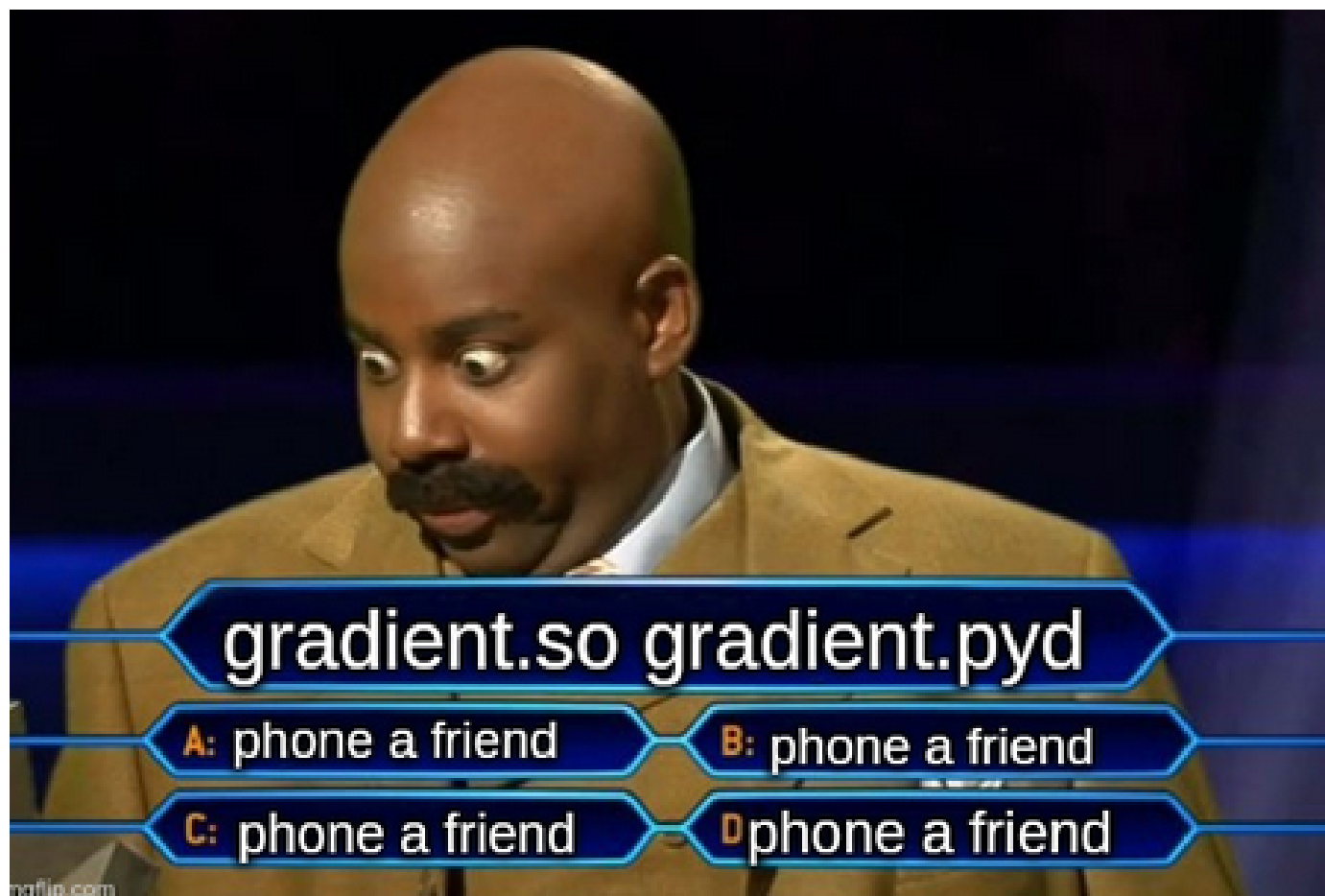


Figure 1. GitHub issue that share the infected PoC repo

Someone suggested two PoCs for a critical unauthenticated RCE affecting a Joomla plugin and asked whether we could implement a matching Nuclei template. As usual, we started by reviewing the code, its authors, and every dependency before executing anything. Precaution paid off. The second PoC (<https://github.com/ogenich/CVE-2026-48908>, now deleted) depended on an unfamiliar package named "frint". The repository owner, 'ogenich', had created their GitHub account only two weeks earlier, with all activity set to private. "frint" itself had appeared on PyPI just a month before, and, even more curiously, depended on another newly published package named "skytext".

"skytext" shipped only precompiled wheels containing native extensions (gradient.so on Linux and gradient.pyd on Windows), and promised "Beautiful & Blazing Fast Terminal Colors for Python". We do love colors and speed, and after VirusTotal reported 0 detections, curiosity got the better of us. We thus opened the binary in Ghidra.

Things soon got truly colorful: the `PyInit_gradient` entrypoint decompiled into an obfuscated function, featuring hash-based module check, PEB walking and several compressed, XOR-encrypted blobs hidden in the `.rodata` section. At that point, our appreciation for "beautiful terminal colors" reached a whole new level, so we decided:



Together with our friends at Sekoia TDR, we dug deeper into this surprisingly sophisticated dependency chain. What we eventually uncovered was a fully functional - and almost completely undocumented - Python remote access trojan (RAT).

Infection Chain

The investigation stems from a dependency confusion tactic, where a malicious Python package was added into the repository's `requirements.txt`, effectively trojanising the distributed PoC. The infection is triggered when the victim executes the `pip install` command to download and install the PoC external requirements.

The infection chain is as follows:

1. **Package Download:** The Python package `frint` is downloaded to the target system.
2. **Dependency Installation:** The `skytext` package, a dependency of `frint`, is transitively installed during the initial setup.
3. **Library Loading:** A specific native extension library (`.so` for Linux or `.pyd` for Windows) is loaded directly into system memory.
4. **Library Execution:** The corrupted native library executes immediately when the Proof of Concept (PoC) Python file runs.
5. **Script Decryption:** The execution process decrypts five small Python scripts using a custom algorithm and a unique key, requiring the runtime context of the specific PoC environment to succeed.
6. **Payload Retrieval:** One of the decrypted scripts functions as a downloader, retrieving a subsequent Python script from `api.mapbox[.]com`.

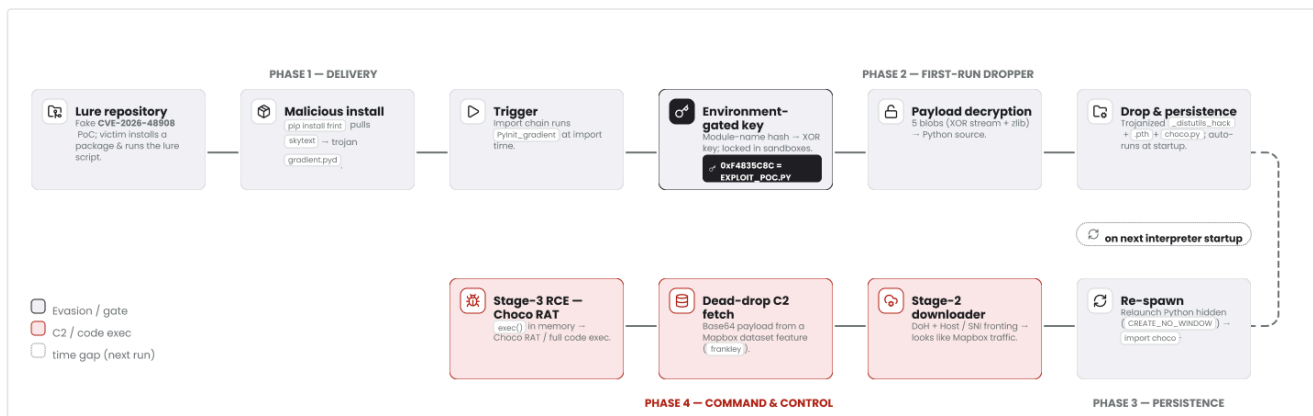


Figure 2. ChocoPoC infection chain

A Word About Python Native Extension Loading

Python's package loading mechanism in the presence of compiled extension libraries deserves special attention to understand the infection chain used by "skytext" package:

Python handles the loading of compiled extension modules - represented as .so (Shared Object) files on POSIX systems and .pyd (Python DLL) files on Windows - via the importlib subsystem's ExtensionFileLoader. When an import statement targets a binary module, Python searches the directories specified in sys.path. If both a compiled extension (e.g. gradient.so) and a Python source module (gradient.py) with the same name are present in the selected location, the compiled extension takes precedence and shadows the source file during import. Upon locating the file, it leverages native operating system APIs, specifically dlopen on Linux/macOS and LoadLibrary on Windows, to dynamically link and load the binary into the current process's memory space. Once mapped, Python invokes a standardised initialisation function (e.g. PyInit_<module_name> in the current infection chain it is PyInit_Gradient) defined via the Python/C API, which bootstraps the module and registers its native functions into sys.modules.

The execution of the PyInit_gradient entry point marks the onset of malicious activity. Analysis indicates that the earlier stages of the installation process function as a decoy, leveraging standard Python mechanism to divert suspicion and conceal the delivery of the compiled binary payload.

Compiled Library

This report only details the analysis of the Windows library (gradient.pyd). Linux/macOS implementation is almost identical: the dynamic API resolution part does not apply to POSIX systems but overall mechanisms are the same.

The library minimises its footprint by eschewing standard import tables, instead dynamically resolving the Python C-API and kernel32.dll functions via Process Environment Block (PEB) walking the export-hashing.

Code 1. Python equivalent of the Windows API resolution primitive

Before proceeding, it executes anti-analysis routines, by calling CheckRemoteDebuggerPresent and inspecting debugging registers Dr0 to Dr1 via GetThreadContext to detect hardware breakpoints.

Code 2. Decompiled code excerpt showing dynamic API resolution coupled with debugger identification

Besides, in the init of the export function (PyInit_gradient), the malware also leverages PEB walking to search for Python C-API functions used to read the interpreter state. These states are obfuscated using basic XOR instruction with hardcoded keys, the deobfuscated Python strings are api_version, modules or __file__. Then the malware enumerates sys.modules and collects each module's __file__.

To ensure it only executes in the intended context, the payload employs environmental key gating; it hashes the basenames of all loaded Python modules via their __file__ attributes, proceeding only if a module matches the hash 0xF4835C9C (corresponding to uppcased EXPLOIT_POC.py). The same characteristic hash-based gating algorithm appears in related campaigns (see next sections), using either uppcased exploit.py or exploit_poc.py. This explains why isolated detonation doesn't trigger any malicious behavior nor detection in a typical sandbox: malware execution is conditioned to the full lure PoC runtime.

```

hash = 0x261C; // seed
while ( current_char )
{
    current_char_upcase = toupper(current_char);
    current_char = *v15++;
    hash = ((4 * hash * current_char_upcase) ^ hash) + 1;
}
unknown_libname_11(v22);
if ( hash == 0x7A93452A )
    break;

```

Figure 3. Hashing algorithm

Once validated, the malware decrypts five embedded payloads and achieves persistence by dropping a trojanised `_distutils_hack` package and malicious `.pth` files into the Python `site-packages` directory, subsequently timestamping the files to evade forensic detection. The payloads are stored obfuscated and compressed using the Zlib algorithm. In the annex, we provide a python helper to deobfuscate and inflate next stage payload, statically. The execution of the next stage is done asynchronously when the user starts a new Python interpreter: the `_distutils_hack` being solicited, it triggers the `add_shim` function which subsequently executes the next stage.

Finally, the dropped `.pth` file spawns a hidden Python process to import `choco` (`choco.py` is the name given by the attacker to their final stage), a stage 2 downloader which fetches subsequent malicious code from a Mapbox dataset acting as a dead-drop C2 point. Execution is simply and directly made via `exec`.

Dropped as	Purpose
<code>distutils-precedence.pth</code>	setuptools <code>.pth</code> legit-looking one-liner hook, enabled with <code>__import__('_distutils_hack').add_shim()</code> . Auto-executed on every interpreter start. Loads the trojanised <code>_distutils_hack</code> package
<code>_distutils_hack/override.py</code>	setuptools <code>override.py</code> one-liner: <code>__import__('_distutils_hack').do_override()</code> .
<code>_distutils_hack/__init__.py</code>	Trojanised package init: setuptools <code>_distutils_hack</code> code with the malicious re-spawner (the E810 block) appended to its tail which runs on every import
appended into <code>__init__.py</code>	The malicious re-spawner bootstrap itself: uses env markers to avoid recursion, re-launches a hidden python (<code>CREATE_NO_WINDOW</code>), then runs the downloader.
<code>choco.py</code>	Next stage dead-drop downloader: DoH-resolves (AliDNS/Cloudflare), Host-header/SNI-fronts <code>api.mapbox[.]com</code> , GET the Mapbox dataset feature, base64-decodes a hardcoded properties field's value and executes it.

ChocoPoC Downloader

Upon execution, the malware's Python downloader named `choco.py` decodes some Base64 strings. Then it setup a HTTP session wrapper leveraging DoH (DNS-over-HTTPS) to retrieve the C2 address. By tunnelling DNS resolution inside ordinary HTTPS requests via DoH queries, the malware evades blocking and detection of plain UDP/53 DNS queries by local resolver, corporate DNS, EDR or DNS sinkholes. Using function `get_dns_ip`, the downloader then resolves `api.mapbox[.]com`'s IP using public DoH resolvers - `dns.alidns[.]com` with `cloudflare-dns[.]com` as the referenced alternate - and pulls the "A" record straight out of the JSON response. After obtaining its C2's IP, it opens an HTTPS connection to it (rather than to the hostname), using a custom `HostHeaderSSLAdapter` that forces TLS SNI / `assert_hostname` to `api.mapbox[.]com` and a monkey-patched `Session.request` that rewrites the URL's "netloc" to the IP while keeping the "Host: `api.mapbox[.]com`" header. This is a domain-fronting-style technique: the TLS handshake and certificate validation still look like legitimate Mapbox traffic, so the C2 channel blends into normal API usage and survives IP-pinning. It then GETs a particular Mapbox dataset feature at `hxxps://api.mapbox[.]com/datasets/v1/frankley/cm0r0tcxf008i1mmpd7apt903/features/dm370543acmdopk296nahbtua` (the dead-drop) using a public Mapbox token, Base64-decodes the blob from `properties['dm370543acmdopk296nahbtua']` to Python source, and `exec()`s it, delivering the final stage. All the C2 strings (domain, URL, feature id) are base64-encoded in the source, and a plain fallback request to the full `api.mapbox[.]com` URL is tried if the DoH path fails.

ChocoPoC RAT

The final stage is a Python script that functions as a RAT and an information stealer. A notable characteristic of this threat is its use of legitimate Mapbox API as a covert C2 mechanism, utilising Mapbox datasets and features to retrieve instructions and drop status updates. For heavier data exfiltration, it falls back to a dedicated external IP address that hosts an HTTP server.

The malware primarily focuses on harvesting user credentials and valuable local information:

- **Browser Data:** It targets Google Chrome, Brave, Microsoft Edge, and Mozilla Firefox to extract stored credentials (passwords), cookies, autofill data, and web history.
- **System Files:** It scans user directories for specific file types, looking for text file (.txt), markdown documentation (.md), and database files (data.db, local-store.db).
- **System Intelligence:** It attempts to gather shell histories (like .bash_history or .zsh_history), active network configuration, and running process lists to understand the victim's environment.

The core architecture relies on a continuous loop that polls the C2 channel for encrypted strings, executing specific actions based on the prefix of the command received:

- **ho!a (System Reconnaissance):** Triggers a comprehensive reconnaissance routine, running native system commands (such as ipconfig, tasklist, uname, and netstat) to gather and exfiltrate basic system information.
- **cmd <command> (Arbitrary Command Execution):** Executes arbitrary shell commands directly on the host operating system.
- **python <base64_code> (Dynamic Code Execution):** Decodes and executes arbitrary Python code on the fly using Python's native exec() function to provide extensible functionality.
- **get <path> / get <path>/* (Data Staging and Exfiltration):** Targets specific local files or folders, compressing them into standard archives before uploading them.
- **browserdata (Credential and Profile Harvesting):** Triggers automated decryption and collection routines for local web browser profile folders.
- **dormir (Beacon Interval Adjustment):** Alters the beaconing interval (sleep timer) to speed up or slow down communication with the C2 server.

Several commands, python functions and variables are named in spanish (*dormir*, *hola*, *dataset_usuario*, *obtener_id*, *devolver*, *nombre*, *actualizacion*, etc) and we denoted a few python coding mistakes such as function calling typos, checks against undefined variables, suggesting that the RAT isn't fully tested nor AI-generated.

A Persistent Supply Chain Threat Against CVE PoCs

Multiple PoCs Distribute the Same Python RAT

When investigating this supply chain campaign, which stems from a weaponised Proof of Concept (PoC) to exploit CVE-2026-48908 (Joomla RCE), we discovered additional PoC repositories on GitHub implementing highly similar infection chains to deliver the same Python RAT. To systematically chase those lures, we tried to reverse-identify all GitHub projects that include "frint" or "skytext" in their requirements.txt or pyproject.toml files. After various unsuccessful GitHub dorks, we uncovered two additional repositories (github.com/bolubey/CVE-2026-0257 and github.com/bolubey/CVE-2026-50751) by reverse-indexing the entire github.com/nomi-sec/PoC-in-GitHub archive for 2025 and 2026 and systematically retrieving every requirements.txt file. Those two repositories, created by user *bolubey*, included the same "frint" / "skytext" dependency chain. This account was banned at the time of writing and we were not able to collect the GitHub committer's email address.

We also discovered sibling malicious PoCs, which were previously analysed on 26 March 2026 by the Chinese cybersecurity researcher "Ch1ngg" from "White Hat 100 Security Attack and Defense Laboratory", who published a technical analysis of a [trojanized MongoBleed PoC shipped into MDUT Extend](#) (a Chinese community DB pentest tool). While the later campaign used different malicious Python packages, namely, "slogsec" and "logcrypt.cryptography", their source code are very similar and implement identical capabilities:

- Mapbox abused both as a staging server for a next-stage Python RAT and as an exfiltration channel alongside a specific IP address.
- Assets show spanish-looking strings (pozoz.py, variables and function naming)
- Perfectly matching environmental gates, function naming, anti-recursion env vars and Mapbox feature ID

We clearly observe a similar pattern where attackers repeatedly target vulnerability researchers who may incorporate backdoored PoCs to larger pentesting and scanning frameworks (MDUT, Nuclei, etc) to expand into a double supply chain attack, wherein the primary package-based compromise directly facilitates a second framework-level supply chain infection.

By pivoting on GitHub accounts' repositories and raw committer email addresses, we list the following PoCs trojanised to distribute the same final ChocoPoC RAT for several high-profile CVEs:

GitHub PoC repo	CVE	Publication date	GitHub committer's email address	Malicious Python dependencies
lincemorado97/CVE-2025-64446_CVE-2025-58034	CVE-2025-64446 FortiWeb path traversal	2025-11-18	21104040041[.]@student.uin-suka.ac[.]id	slogsec, logcrypt.cryptography
lincemorado97/CVE-2025-55182_CVE-2025-66478	CVE-2025-55182 React2Shell	2025-12-08	21104040041[.]@student.uin-suka.ac[.]id	slogsec, logcrypt.cryptography
lincemorado97/CVE-2025-14847	CVE-2025-14847 MongoBleed	2025-12-31	21104040041[.]@student.uin-suka.ac[.]id	slogsec, logcrypt.cryptography
bolubey/CVE-2026-0257	CVE-2026-0257 PAN-OS Auth bypass	2026-05-29		frint, skytext
ogenich/CVE-2026-10520	CVE-2026-10520 Ivanti Sentry OS command injection	2026-06-10	200111085[.]@ogrenci.ibu.edu[.]tr	frint, skytext
bolubey/CVE-2026-50751	CVE-2026-50751 Checkpoint VPN Auth bypass	2025-06-16		frint, skytext
ogenich/CVE-2026-48908	CVE-2026-48908 Joomla SP Page builder RCE	2026-06-24	200111085[.]@ogrenci.ibu.edu[.]tr	frint, skytext

While this exhibits two distinct 2025 and 2026 campaigns based on the GitHub accounts deploying the malicious PoC and the specific dependency packages used, we assess with high confidence that a single threat actor conducted all campaigns. Indeed, the cybercriminal likely rotated GitHub, pip, and Mapbox accounts to prevent new infection chains from being detected by security solutions and avoid relying on unique, abused accounts susceptible to being banned. Most notably, the reuse of a same mapbox feature ID, hash-based environment gate and anti-recursion environment variables are markers of a shared opsec arsenal.

Abuse of Mapbox Datasets

As detailed in the infection chain analysis above, the ChocoPoC downloader retrieves the ChocoPoC RAT from a Mapbox dataset, and the RAT subsequently exfiltrates harvested data to another Mapbox dataset. For that, the Python code embeds Mapbox usernames and API keys controlled by the attacker. The values of these credentials differ between the 2025 and 2026 campaigns, which aligns with the account rotation pattern identified for GitHub and PyPi services.

Analysing Mapbox API keys also reveals interesting patterns. Those JWT tokens follow a three-part structure separated by dots:

- A prefix indicating the token type (pk for public keys, sk for secret keys, and tk for temporary tokens).
- A payload (secret), consisting of a Base64-encoded JSON containing the username in the field `u` and the token ID in the `a` field.
- A cryptographic signature encoded in Base64 HMAC.

For example, the public key used by the ChocoPoC downloader during the 2026 campaign is:

```
pk.eyJ1IjoiaZnJhbmtzZXkiLCJhIjoieY21vNzFzaXNzMDJrMjJxcHJqY3JscnlpYSJ9.fuMfMgsxl0GxRy44A-y0WQ
```

The payload section decodes into:

```
{"u": "frankley", "a": "cmo71siss02k22qprjcrlyia"}
```

Analysis indicates the value in the `"a"` field is a CUID-style identifier containing an 8-character timestamp segment, following the `c` prefix: `mo71siss`. When decoded from Base36, this segment indicates that the *"frankley"* account was highly likely created around 19 April 2026.

Below is an overview of the estimated creation date for each Mapbox token we collected during this investigation:

Mapbox username	Mapbox dataset/feature	Token
mattallahsaed	cmismaye7000s1mp2v8fkn4lp dm370543acmdopk296nahbtua	pk.eyJ1IjoibWF0dGFsbGFoc2FIZCIsImEiOiJjbWlzbWpncWkwNHRmM2ZzMWd1eTBmanQ4InI
rdras	dynamically generated as {username}_{hostname}_{randstr(5)}	sk.eyJ1IjoicmRyYWEiLCJhIjoY21pMmo1bWFrMTJvOTJscjRsenlqOWwydCJ9.vGHDYdHYNEr
frankley	cmor0tcxf008i1mmpd7apt903 dm370543acmdopk296nahbtua	pk.eyJ1IjoiznJhbmtsZXkiLCJhIjoY21vNzFzaXNzMDJrMjJxcHJqY3JscnlpYSJ9.fuMfMgsxIOGxf
james09790	dynamically generated as {username}_{hostname}_{randstr(5)}	sk.eyJ1IjoiamFtZXMwOTc5MCIslmEiOiJjbWoxM3JuNHQwYnh0M2xeWhsMnVyaDZwIn0.ELp

Use of Compromised Accounts

While the GitHub committer's email addresses used by both 2025 and 2026 campaigns are those of two alleged Indonesian and Turkish students, unverified email addresses were used to publish the "*frint*" and "*skytext*" packages (leechuun[.]gmail[.]com and faberhun[.]gmail[.]com). Credentials for two of these four addresses appeared in leak databases, and one highly likely originates from an infostealer compromise.

According to these findings, we assess with high confidence that the attacker primarily employed compromised accounts to publish malicious PyPI packages and PoCs. This TTP has been observed consistently over time, and the use of new accounts for each Python library or for malicious PoC campaigns enables the threat actor to make it harder to identify and remove all malicious code.

Package Download Bumps Reveal Past Waves

According to pepy.tech and pypistats.org, the "*skytext*" package alone has accumulated roughly 2,400 downloads across both Linux and Windows environments, with Linux systems accounting for the majority. While download statistics cannot prove successful compromises, they strongly suggest that researchers and pentesters installed these packages while executing the associated PoCs. More interestingly, download trends for "*frint*" and "*skytext*" closely follow the disclosure of FortiSandbox, Ivanti Sentry and Joomla SP Page builder vulnerabilities, along with the publication of their corresponding PoCs. However, both "*frint*" and "*skytext*" show a bump of downloads on May 5, indicating a successful prior infection wave. May 5 precisely matches the date where Linux Copy Fail top-tier vulnerability has been added to CISA Known Exploited Vulnerabilities (KEV) catalog and the massive spread of public exploits for it. We may hypothesise that vulnerability researchers interested in such vulnerabilities were also targeted at that time. Similarly, "*slogsec*" package shows a narrow bump on 25 March, which tightly matches a critical Langflow RCE's disclosure and addition to KEV.

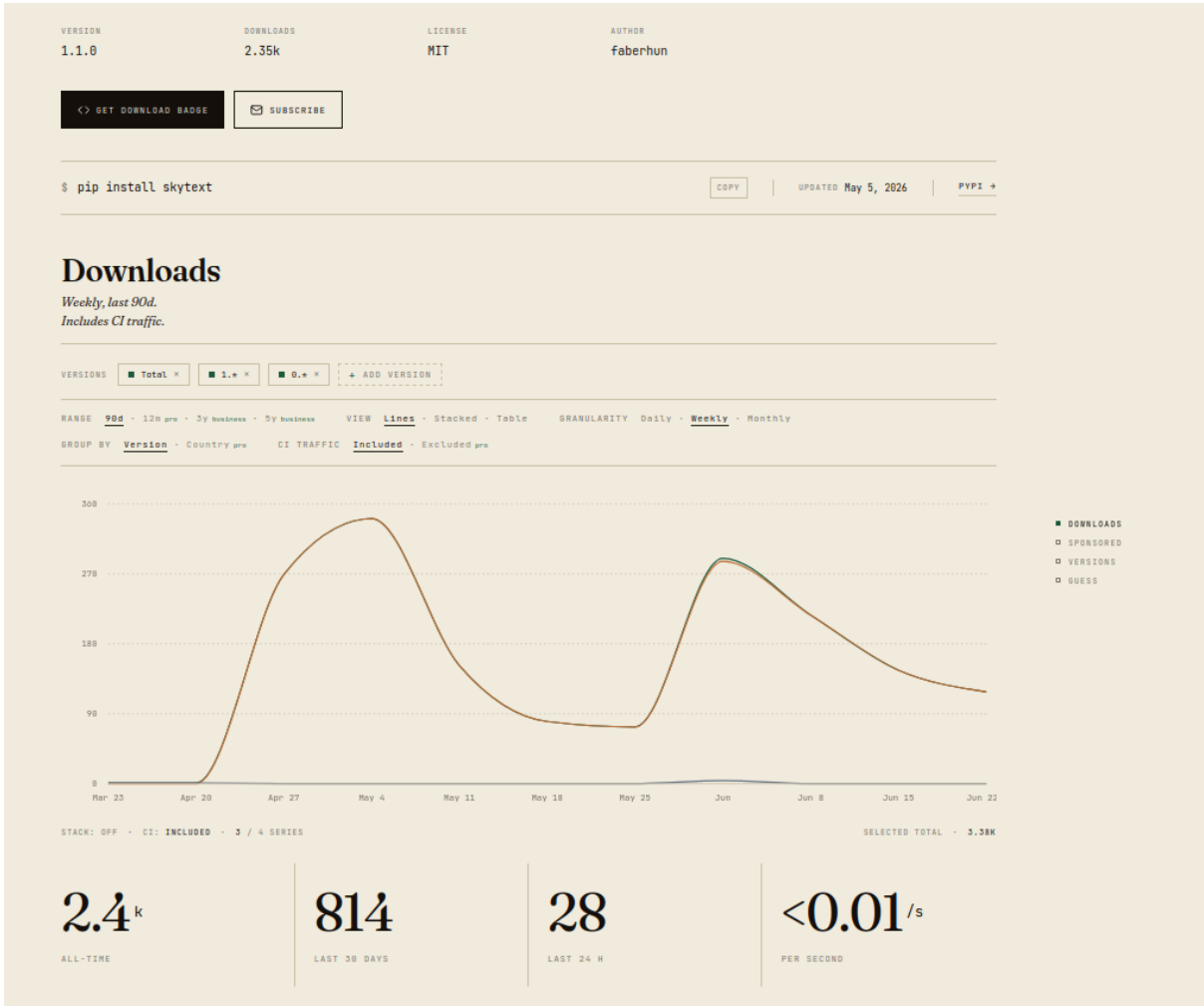


Figure 4. Skytext PyPi repository download statistics

Conclusion

Although this investigation started with a suspicious PoC, it quickly became clear that this was not an isolated incident. The same supply chain technique had already been used successfully to compromise existing offensive security tooling. Rather than inventing a new attack vector, the operators simply adapted a proven one to target the vulnerability research ecosystem. Repeated focus on top-tier hot vulnerabilities and operational security also characterise a deliberate campaign. Disposable GitHub and PyPI accounts, apparently created from confirmed leaked email addresses, were used to publish both the malicious packages and the lure PoCs.

The broader lesson is not that proof-of-concepts have suddenly become dangerous. Malicious PoCs have existed for decades. What is changing is the delivery mechanism. By hiding malware in transitive dependencies, attackers can keep the visible PoC almost entirely benign while delegating the malicious behavior to packages that appear harmless in isolation. This significantly reduces the chances of detection by antivirus products, automated sandboxes, and even careful manual reviews, especially when the malicious behavior only emerges through interactions between multiple packages.

Security researchers and penetration testers are particularly attractive targets. They routinely execute untrusted code, often work with elevated privileges, and handle customer credentials, confidential reports, and sensitive infrastructure during engagements. Compromising a single researcher can therefore provide an attacker with access far beyond one workstation.

IoCs & Technical details

Malicious PyPi Packages

Package	Version	SHA-256 (wheel)
skytext	1.1.0	93739477cd379adef95126b22758c0e644282d2028dd297328ce856fa111dd06

Package	Version	SHA-256 (wheel)
frint	0.1.2	17997e9e0256d0f5d5d21a4852c37f16b338e4bb9c2bec09bdfd822b24aa76b4
slogsec	*	5abd45d6f4a1705dca55d882f017d4768888dce9ad99cea40b3da35c23de5cae

Malicious Binaries / Native Python Extensions

File	SHA-256
gradient.pyd (Windows)	40569318e89db751ff3886b2617d990d8a343f0d1d8727b7f978a28129ca36bc
gradient.so (Linux)	320b29844892e3c59bc6fcb07e701b2b3230a37cb4a13176174e9e294ec6d43e

Lure / Distribution Github Repos

- https://github[.]com/lincemorado97/CVE-2025-64446_CVE-2025-58034
- https://github[.]com/lincemorado97/CVE-2025-55182_CVE-2025-66478
- https://github[.]com/lincemorado97/CVE-2025-14847
- https://github[.]com/ogenich/CVE-2026-10520
- https://github[.]com/ogenich/CVE-2026-48908
- https://github[.]com/bolubey/CVE-2026-0257
- https://github[.]com/bolubey/CVE-2026-50751

Mapbox indicators

hxxps://api.mapbox[.]com/datasets/v1/frankley/cmor0tcxf008i1mmpd7apt903/features/dm370543acmdopk296nahbtua?
access_token=pk.eyJ1IjoibW0dGFsbGFoc2FLZCIsImEiOiJjbWlzbWpncWkwNHRmM2ZzMWdl1eTBmanQ4In0.VNFutzqzaSVfDiwQFr7_gQ

hxxps://api.mapbox[.]com/datasets/v1/mattallahsaed/cmismaye7000s1mp2v8fkn4lp/features/dm370543acmdopk296nahbtua?
access_token=pk.eyJ1IjoibW0dGFsbGFoc2FLZCIsImEiOiJjbWlzbWpncWkwNHRmM2ZzMWdl1eTBmanQ4In0.VNFutzqzaSVfDiwQFr7_gQ

Indicator	Type
frankley	Mapbox account
mattallahsaed	Mapbox account
james09790	Mapbox account
cmor0tcxf008i1mmpd7apt903	Mapbox dataset
dm370543acmdopk296nahbtua	Mapbox feature key
pk.eyJ1IjoibW0dGFsbGFoc2FLZCIsImEiOiJjbWlzbWpncWkwNHRmM2ZzMWdl1eTBmanQ4In0.VNFutzqzaSVfDiwQFr7_gQ	Mapbox private token
sk.eyJ1IjoiamFtZXMwOTc5MCIsImEiOiJjbWoxM3JuNHQwYnh0M2xxeWhsMnVyaDZwIn0.ELpete7yVGAeg52Mrmt2DA	Mapbox public token

Host / Behavioral

Indicator	Value
Environment variable	ZEBUWIAKGPHOQAP006=PTsjBGKQUxZorq2
Environment variable	JKHWQVEKRASDF12=JKHKJ23VAS8DF9

Network - Stage 3 Exfiltration Server

Indicator	Value
Chunked upload endpoint	hxxp://91[.]132[.]163[.]78:8001/assets/static/bundle[.]ext[.]min[.]de5b2bc9[.]jjs

Indicator	Value
RSA public key	MIICljANBgqhkiG9w0BAQEFAAOCAg8AMIICGgKCAgEAtctH5Bk1nM/Jnp6UUUsC jymtKzujqWF3ljji12z529AvQjSUrodCc14h4v40zPXZ72SkKiEEaoHhOYA56uaC ...

Annexes

ChocoPoC RAT Spawner

Blob Deobfuscator
