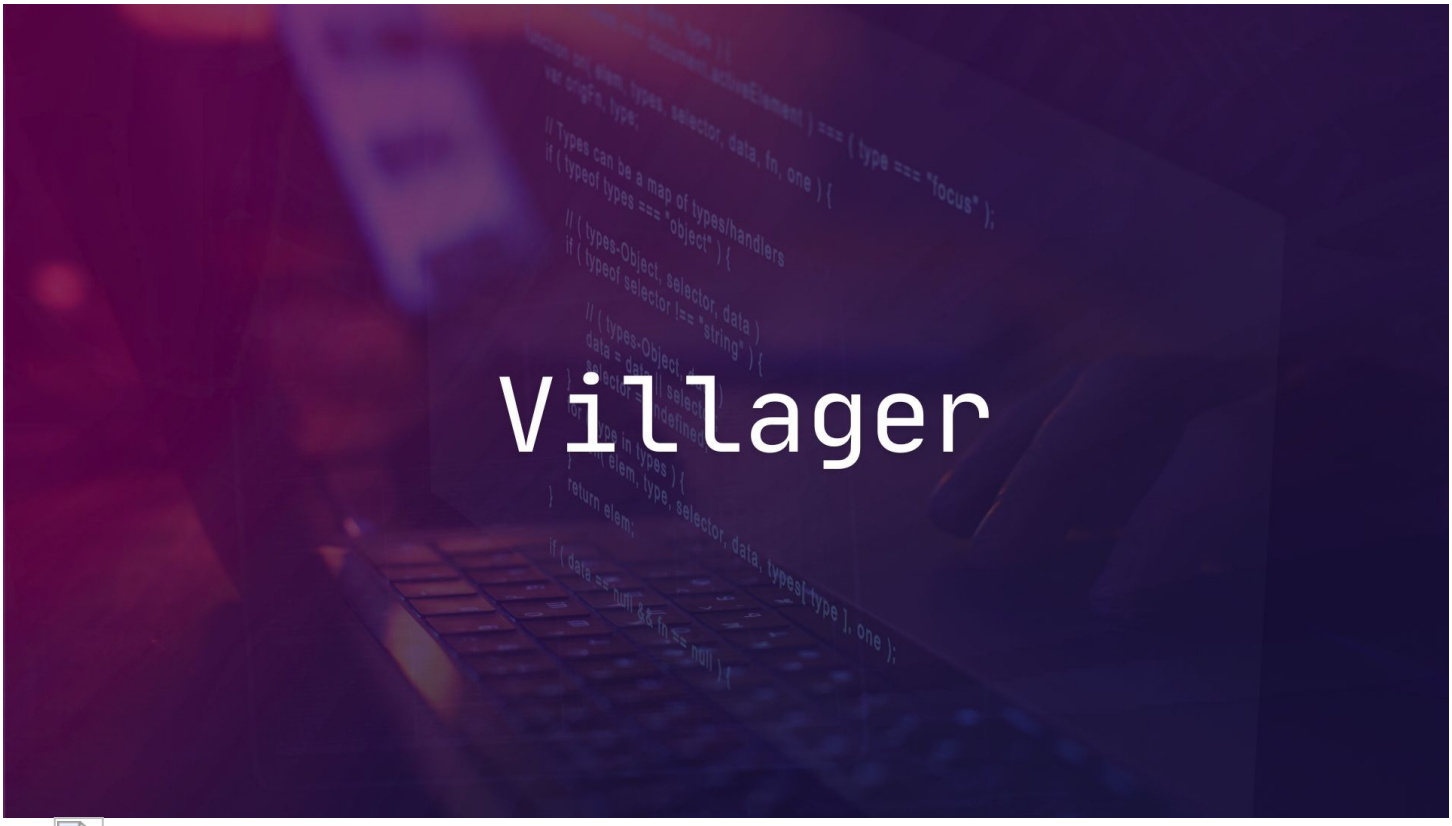
www.secureblink.com /cyber-security-news/from-red-team-to-rogue-villager-threatens-to-become-the-next-cobalt-strike

Unknown Title

⋮ 9/16/2025



AI Penetration Testing

From Red Team to Rogue, Villager Threatens to Become the Next Cobalt Strike

Villager an AI-native penetration-testing framework tied to “Cyberspike.” This dossier sequences discovery, architecture, capabilities, indicators, risk assessm...

16-Sep-2025

12 min read

Villager is an AI-native, multi-container penetration-testing and orchestration framework observed in the wild and published to PyPI; Straiker’s research attributes the project to a group/company using the “Cyberspike” name (domain `cyberspike.top`) and a PyPI maintainer alias `stupidfish001`.

Architecturally, Villager fuses generative AI task planning — implemented as a retrieval-augmented prompt bank and model endpoint — with an MCP-style coordination plane and automated, ephemeral Kali-based

container drivers, enabling programmatic decomposition of high-level objectives into parallelized recon, fingerprinting, and exploitation tasks.

That synthesis materially reduces the skill and time required to discover, prioritise, and operationalise vulnerabilities, raising the likelihood of large-scale, low-effort abuse if the tool is repurposed by malicious actors or inadvertently introduced into CI/dev environments.

This dossier, therefore, documents the discovery timeline, component architecture, observed capabilities, verified indicators of compromise (IOCs), high-level misuse scenarios, risk assessment, and prioritised mitigations and detection guidance; it intentionally omits operational exploitation instructions and focuses solely on defensive, forensic, and governance implications for security teams.

Discovery and Evidence Base

The emergence of *Villager* was first documented by Straiker's AI Research (STAR) team, who published their findings on September 11, 2025, in a blog post authored by Dan Regalado and Amanda Rousseau.

Their analysis positions Villager as a successor-style framework to Cobalt Strike, albeit architected from the ground up as an AI-native penetration-testing platform. The credibility of these findings rests not only on Straiker's write-up but also on a body of corroborating artifacts drawn from public repositories and malware-sharing platforms.

Specifically, the research references:

- **PyPI package entries** for both `villager` and its supporting module `kali-driver`, maintained under the alias **stupidfish001**.
- **Infrastructure indicators**, such as a GitLab registry under the domain `gitlab.cyberspike.top`.
- **Historical binaries and installers** for earlier "Cyberspike Studio" tooling uploaded to **VirusTotal**, showing lineage with known RAT families and plugin-based exploit modules.

Confidence in these observations stems from Straiker's direct evidence, which includes screenshots of package metadata, test files, code snippets, and architectural diagrams. Taken together, these sources provide a well-grounded basis for technical assessment without requiring speculative interpretation.

High-level Architecture

Villager implements a layered orchestration pattern that pairs an agentic, model-driven decision layer with a lightweight coordination plane and ephemeral tooling runners.

At the top of the stack sits a **GenAI task planner** that ingests a high-level objective (for example: "assess example.com for high-risk vulnerabilities"), reformulates that objective into a sequence of discrete subtasks, and issues structured subtasks to downstream drivers.

Straiker's analysis shows this is implemented as a retrieval-augmented prompt bank and a hosted model endpoint — a classical **RAG (retrieval-augmented generation)** pattern where non-parametric knowledge

(indexes/prompts) supplements the generative model's outputs to improve accuracy and provenance.

Beneath the planner is an **MCP-style coordination plane** (message/coordination broker) that acts as the central messaging bus for task negotiation, state management, and telemetry aggregation. This plane provides a programmatic API surface and queueing semantics so the planner can submit tasks, observe progress, request re-planning, and enforce verification gates.

In Villager the coordination plane exposes a FastAPI-backed tasking interface (documented frameworks such as **FastAPI** are a common and efficient choice for such HTTP task APIs), which gives a predictable, typed schema for task submission and retrieval and enables synchronous and asynchronous orchestration patterns.

The execution tier is realized as **containerized tool drivers** — primarily ephemeral Kali-based containers — orchestrated programmatically from the coordination plane. Each driver provides a bounded runtime that can be instantiated, instrumented (e.g., injected with task parameters and an agent context), and destroyed on a short lifecycle to limit forensic surface area.

Practically this is implemented with Docker APIs or the Docker Python SDK (the same primitives documented in the Docker SDK let an application create containers, attach volumes/networks, start processes, and enforce removal policies), enabling a controller process to call `create()`/`run()` with controlled mounts, capability drops, and network namespaces.

Villager's reported Kali driver behavior (randomized SSH ports, 24-hour self-destruct, image pull from a private registry) maps directly to these container lifecycle controls.

Tooling inside containers is heterogeneous and task-specific: full Kali toolchains for network and host scanning, browser automation agents for web-interaction flows, and lightweight evaluators that can execute small code snippets (`pyeval()`-style primitives) for payload shaping or environment introspection.

Browser automation is typically implemented with frameworks such as Playwright or Chromium automation endpoints that expose programmatic navigation, DOM snapshots, and form interactions — this lets the planner validate complex auth flows or exercise client-side logic before delegating to native exploit modules.

To ensure machine-readable interoperability between planner outputs and tool responses, Villager integrates schema enforcement via **Pydantic AI** (or similar schema-validation layers).

Schema enforcement guarantees that each tool returns predictable, typed outputs (for example: a JSON schema for "service fingerprint result" or "exploit attempt result"), which the planner uses to validate success criteria, update the task graph, and decide whether to progress, retry, or escalate to human review.

This pattern reduces brittle, string-matching logic and allows for automated verification gates and reproducible audit trails.

In a nutshell, Villager's architecture couples (a) a RAG-style generative planner for intent decomposition, (b) an HTTP/messaging coordination plane for stateful orchestration, (c) programmatic container drivers for

ephemeral, instrumented execution, and (d) schema validation for deterministic inter-component communication.

Each of these design choices—well understood in modern cloud and agent architectures—when combined, creates a highly automated, scalable playbook execution environment that can parallelize reconnaissance and exploitation tasks while minimizing the manual effort required to operate them.

Observed Capabilities & Behavioral Patterns (non-operational)

> Note: descriptions are intentionally non-actionable — focused on defensive understanding.

- **Automated task decomposition & orchestration:** Villager accepts a high-level objective (e.g., “find and exploit vulnerabilities in example.com”), decomposes into smaller tasks (enumeration → fingerprinting → vulnerability checks → exploitation), and orchestrates distributed tools to execute them in parallel or sequence, tracking dependencies and progress.
- **Model-driven decisioning:** A retrieval-augmented generation (RAG) pattern selects and refines prompts from a large curated prompt DB to guide tool selection and next steps.
- **Ephemeral containerized execution:** Tasks are executed in on-demand Kali containers that can be created, used, and destroyed automatically; containers reportedly enforce short lifetimes and ephemeral logging to complicate forensic recovery.
- **Polymorphic session/environment controls:** Randomized SSH ports, ephemeral logs, and container self-destruct policies observed in the architecture are intended to reduce forensic visibility and persistency on the hosting infrastructure.
- **Multi-tool integration:** Integrates browser automation, Kali toolchains (scanners, vulnerability tools), and direct code execution primitives to craft payloads, probe APIs, and run exploitation routines.
- **Task verification & gating:** A verification mechanism (task verification step) is used to validate outputs before progressing to subsequent tasks.
- **Scale & distribution vectors:** Published as PyPI packages (`villager`, `kali-driver`) making supply-chain distribution and casual installation possible; Straiker reports ~10k downloads in ~two months.

Indicators of Compromise (IOCs) & package metadata (extracted)

> Collated from the report and supporting artifacts — use for situational awareness, blocking, and threat intel ingestion.

Domains / Endpoints

- `cyberspike.top` (and subdomains: `gitlab.cyberspike.top`, `www.cyberspike.top`)

- `gpus.dev.cyberspike.top:8000/v1/chat/completions` (dev model endpoint referenced in code snippets)

IP (defanged in source)

- `111[.]173[.]106[.]69` (reported historic resolution and observed in installer behavior)

PyPI / package metadata

- PyPI package: `villager` — maintainer **stupidfish001**; pre-releases from **2025-07-23**; release candidate **0.2.1rc1** dated **2025-09-15**; license: **MIT**; requires Python ≥3.11.
- PyPI package: `kali-driver` — maintainer **stupidfish001**; first release **2025-05-07**.

Hashes / Binaries

- Reported VirusTotal hash for **Cyberspike Studio v1.1.7**: `40127d53ee0554fa997240fc37860a79` (reported in Straiker figures, dated Dec 10, 2023).

Ports / Services (observed in analysis)

- **25989** — MCP coordination plane
- **1611** — Kali driver control
- **8080** — Browser automation
- **37695** — FastAPI tasking API

Behavioral artifacts

- RAG prompt DB size **~4,201** prompts (metadata indicated in package screenshots).
- Model name: `al-1s-20250421` used by dev endpoint; tokenizer references to GPT-style tokenizers reported.

  If you ingest IOCs into security tools, treat the reported IP/domain names as investigative leads — validate via your infrastructure's threat intelligence before taking network actions.

Attack Scenarios

Purpose: help defenders understand plausible misuse patterns. These are conceptual and non-operational.

1. **Automated external reconnaissance → prioritized scanning**
   - Villager issues discovery tasks across a target surface, spawns Kali containers to run platform-specific scanners, ranks discovered assets by exploitability using model scoring, and returns prioritized targets for manual follow-on by an operator.
2. **API/Browser-oriented exploitation chain**

    - Browser automation probes for authentication flows and CSRF/logic flaws; model suggests appropriate post-auth probes and payload shaping; Kali driver runs relevant exploitation/checkers. Task verification gates prevent blind escalation without confirming success.

3. **Supply-chain / developer environment abuse**
    - Installation of `villager`/`kali-driver` in CI runners or developer machines could provide attackers (or misused insiders) with an automated, containerized test harness that can be repurposed to scan internal networks or pivot from dev hosts.

4. **Rapid campaign automation**
    - Using templates/prompts, a low-skill operator could execute multi-target campaigns at scale: auto-discover → auto-exploit → maintain ephemeral access via containerized primitives.

Risk Assessment

- **Likelihood (near term): High** — code is published to PyPI, downloads in the thousands increase adoption and experimentation by benign red teams and opportunistic malicious actors.
- **Impact: High for organizations without controls** — automated exploitation at scale shortens attacker dwell time and increases blast radius (especially where CI/dev hosts and internal registries are poorly segmented).
- **Overall risk posture:** Elevated — the synthesis of model-driven decisioning, ephemeral execution, and easy distribution materially expands the threat surface relative to traditional, manual tooling.

Defensive Recommendations

Below are defensive actions your organization should treat as high priority.

A. Immediate (Days)

- **Block / monitor Indicators:** Ingest domains/IPs and PyPI package names into threat intel / blocklists; block or monitor egress to `*.cyberspike.top` and known dev endpoints. (Validate before broad blocking to avoid false positives.)
- **Harden package policies:** Enforce strict controls on installation of third-party packages in CI/CD and developer environments (allowlist approved packages; deny direct internet installs).
- **Limit runtime permissions:** Ensure build/CI agents and dev workstations run with least privilege and cannot directly initiate long-lived network containers or arbitrary outbound SSH connections.
- **Network egress controls:** Enforce egress filtering and proxy usage for developer hosts; log and inspect outbound traffic to non-standard destinations and ports.
- **Inventory & detect:** Scan repositories for references to `villager`, `kali-driver`, or `stupidfish001` and remediate unapproved usage.

B. Near term (Weeks)

- **MCP protocol inspection:** Where possible, deploy broker/gateway inspection or protocol allowlisting around inter-service message buses; monitor unusual high-rate task submission patterns.
- **Detect ephemeral container abuse:** Add monitoring for frequent container spin-up events originating from user or build accounts, unusual container images pulled from external registries, and short-lived containers with outbound network activity.
- **Model endpoint telemetry:** If your org uses internal LLM endpoints, audit and monitor model usage for high-risk prompts and chains (e.g., those invoking code execution or system orchestration).
- **IR playbooks for agentic/AI incidents:** Develop and tabletop AI-specific incident response steps: isolating hosting nodes, preserving ephemeral container artifacts, capturing model input/outputs, and obtaining legal/forensic authorization.

C. Medium term (Months)

- **Supply-chain policies:** Adopt stronger supply chain controls for package registries (e.g., signing, SBOM tracking, reproducible builds).
- **AI governance:** Define governance and approval models for any internal agentic tooling, including allowed prompts, model access tokens lifecycle, and human-in-the-loop controls.
- **Continuous red team & purple team exercises:** Simulate agentic adversary patterns (in a controlled environment) to validate detection and containment of ephemeral, model-driven attacks.
- **Threat intel collaboration:** Share sanitized findings and IOCs with industry ISACs and trusted intel partners to collectively track variants.

Detection guidance & hunting queries

> These are high-level detection ideas — do not include exploit commands or operational playbook steps.

- **Hunt for package installs:** Search endpoints, build logs, and container registries for `villager`, `kali-driver`, or references to `stupidfish001`.
- **Monitor for anomalous container lifecycle events:** Frequent create/destroy cycles from unexpected users; containers pulling Kali images from unknown registries.
- **Egress anomaly detection:** Outbound traffic to unusual domains/ports (e.g., dev/model endpoints), especially accompanied by short session durations or randomized SSH port handshakes.
- **Task API access patterns:** Repeated POSTs to tasking endpoints or unusual task graphs from a single user/agent.
- **Model-driven artifacts:** Large numbers of structured tool outputs that conform to predictable Pydantic schemas — useful as a signature to flag automated tool output vs. manual investigator output.

Policy & Governance Recommendations

- **Approve agentic tooling explicitly:** Any agentic/AI orchestration tool must undergo a security approval process before deployment (risk assessment, code review, model vetting).
- **Least privilege model for CI/dev runners:** Remove ability to pull arbitrary container images or to open outbound SSH without explicit policy exceptions.
- **Token & key lifecycle management:** Rotate and scope model/API keys; detect and alert on use outside approved contexts.
- **SBOM & provenance for pip dependencies:** Require SBOMs for third-party packages in production pipelines; require signed packages for critical build steps.

Timeline

- **Nov 27, 2023:** `cyberspike.top` domain registration (reported).
- **Dec 10, 2023:** VirusTotal entries for Cyberspike Studio installer (hash: `40127d53...`) — RAT-like artifacts reported.
- **May 7, 2025:** First release reported for `kali-driver` package (PyPI).
- **Jul 23, 2025:** `villager` pre-releases appear on PyPI (reported).
- **Sep 11, 2025:** Straiker publishes the Villager report.
- **Sep 15, 2025:** `villager` release candidate `0.2.1rc1` observed on PyPI (reported). *(Dates drawn from the artifacts referenced in Straiker's write-up.)*

IOCs (concise list for IOC ingestion)

- Domains: `cyberspike.top`, `gitlab.cyberspike.top`, `gpus.dev.cyberspike.top`
- IP (reported): `111.173.106.69`
- PyPI: `villager`, `kali-driver` (maintainer `stupidfish001`)
- VT hash (reported): `40127d53ee0554fa997240fc37860a79`
- Ports: `25989`, `1611`, `8080`, `37695`

Villager represents a notable architectural evolution: combining orchestrated, containerized offensive tooling with model-driven task planning and a message/coordination plane. The combination reduces attacker skill requirements, increases automation, and amplifies scale. Organizations must treat the publication of such tooling as a call to harden package policies, improve telemetry for ephemeral execution, and evolve incident response and governance to explicitly cover agentic/AI-driven attacks.

Related Articles

FileFix

FileFix phishing embeds PowerShell in clipboard, uses steganographic JPGs to del...

The latest iteration of the *FileFix* attack technique has emerged as a fully weaponized campaign, blending social engineering with stegan...

17-Sep-2025

4 min read

iCloud

Attackers exploit iCloud Calendar invites via Apple servers to deliver phishing ...

Attackers are creating **iCloud Calendar events** whose **Notes/DESCRIPTION** field contains a classic **refund/billing lure** (e.g., fake...

- 13-Sep-2025
- 7 min read

RCE

Actively exploited CVE-2025-21043 lets attackers run code via Samsung’s Quram im...

Samsung fixed a **critical remote-code-execution** bug in the Quramsoft image codec (`libimagecodec.quram.so`) used on Galaxy devices. The...

- 12-Sep-2025
- 9 min read