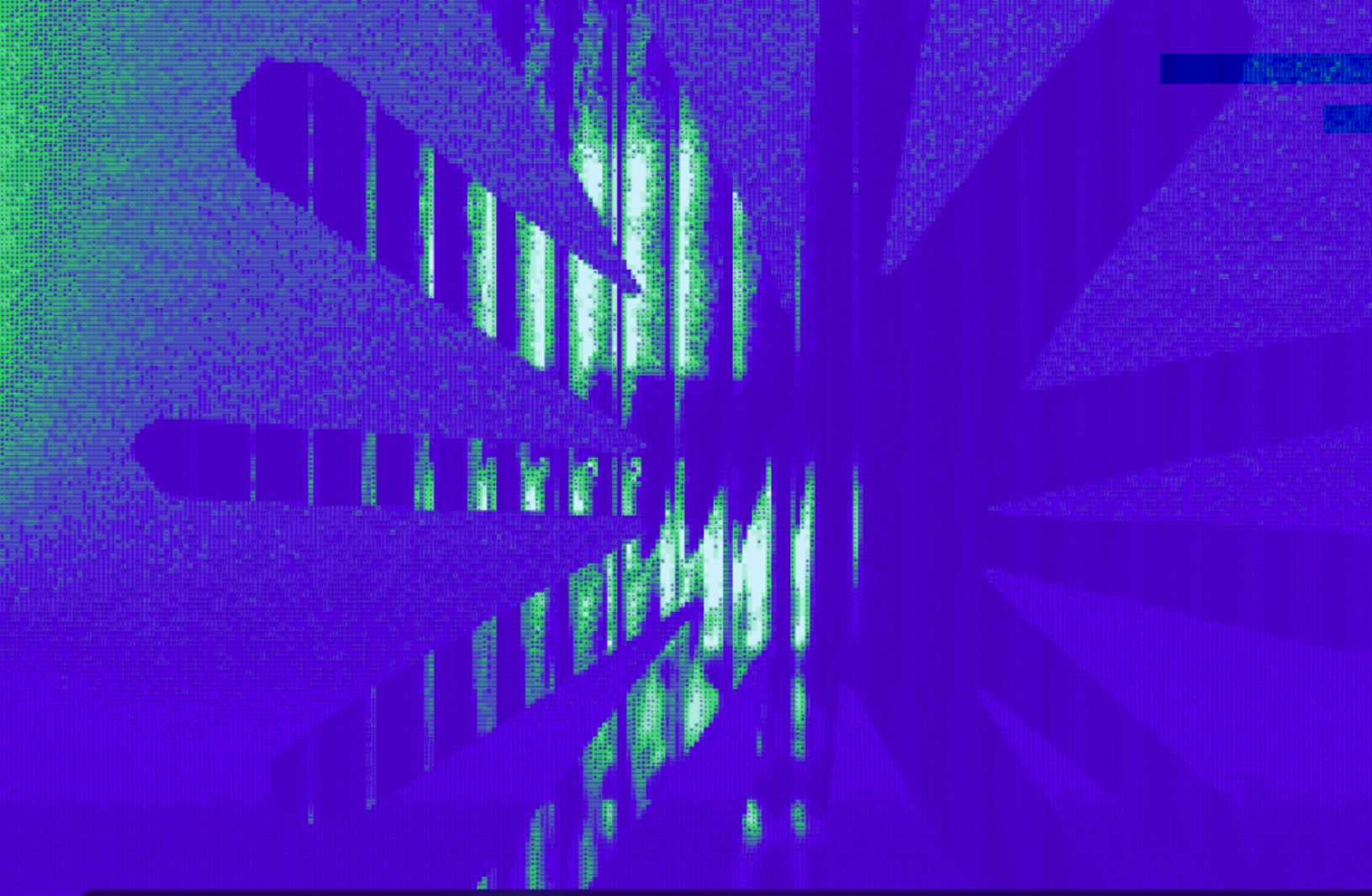


# SentinelOne's Singularity Platform autonomously found and stopped Anthropic's Claude from executing a zero day supply chain attack

SentinelOne BLOG | 7 MINUTE READ

# How SentinelOne’s AI EDR autonomously discovered and stopped Anthropic’s Claude from executing a zero day supply chain attack, globally

*Host-based Behavioral Autonomous AI Detection is by far the most effective way to generically see, and stop both Human and/or machine-speed AI Agent based rogue or malicious activities.*

On March 24, 2026, SentinelOne's autonomous detection caught what manual workflows never could have: a trojaned version of LiteLLM, one of the most widely used proxy layers for LLM API calls, executing malicious Python across multiple customer environments. The package had been compromised hours earlier. No analyst wrote a query. No SOC team triaged an alert. The Singularity Platform identified and blocked the payload before it could run, across every affected environment, on the same day the attack was launched.

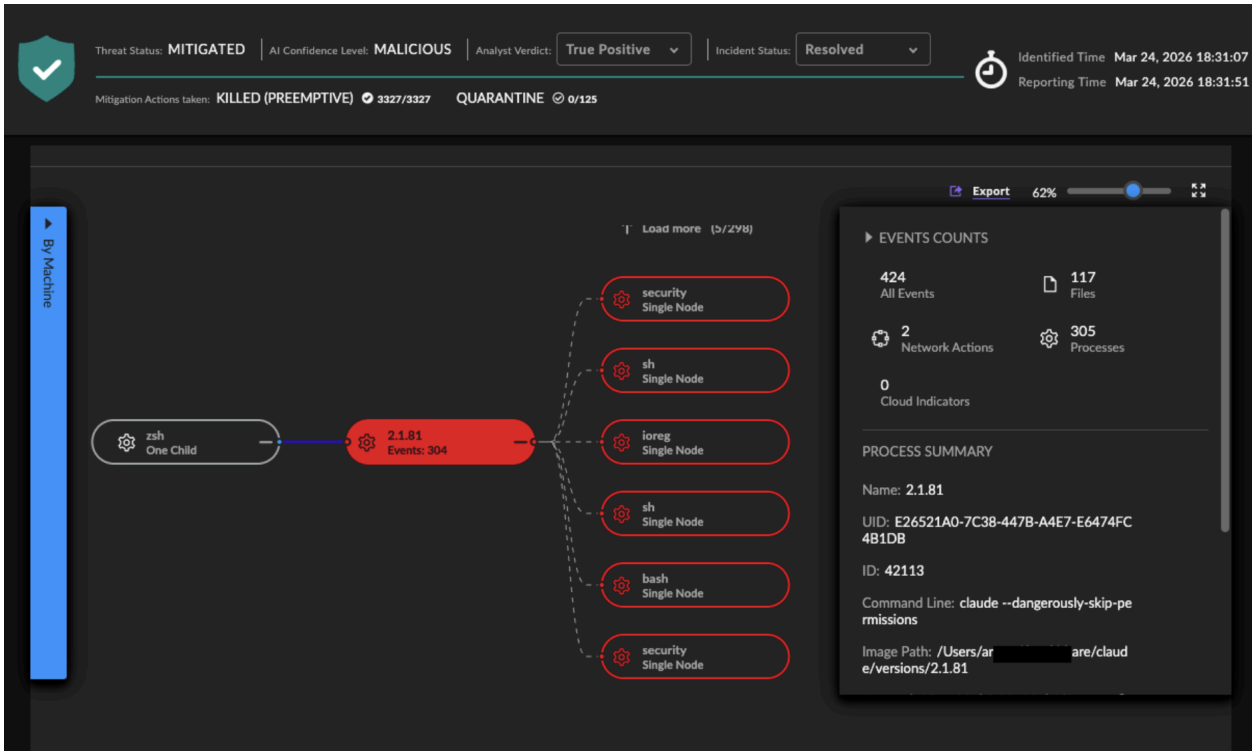
The LiteLLM supply chain compromise is not an anomaly. It is the new pattern: multi-stage, multi-surface, designed to evade manual workflows at every turn. A compromised security tool led to a compromised AI package, which led to data theft, persistence, Kubernetes lateral movement, and encrypted exfiltration, all within a window measured in hours.

**SentinelOne detected and blocked this attack autonomously, on the same day it was launched, across multiple customer environments. No manual triage. No signature update. No analyst in the loop for the initial containment. This is what autonomous, AI-native defense looks like when it meets a real-world threat at machine speed.**

The gap between the velocity of this attack and the capacity of human-driven investigation is the gap where organizations get compromised. Closing that gap is not a feature request. It is an architectural decision.

This is what happens when AI infrastructure gets targeted by a multi-stage supply chain campaign, and what it looks like when autonomous, AI-native defense is already in position.

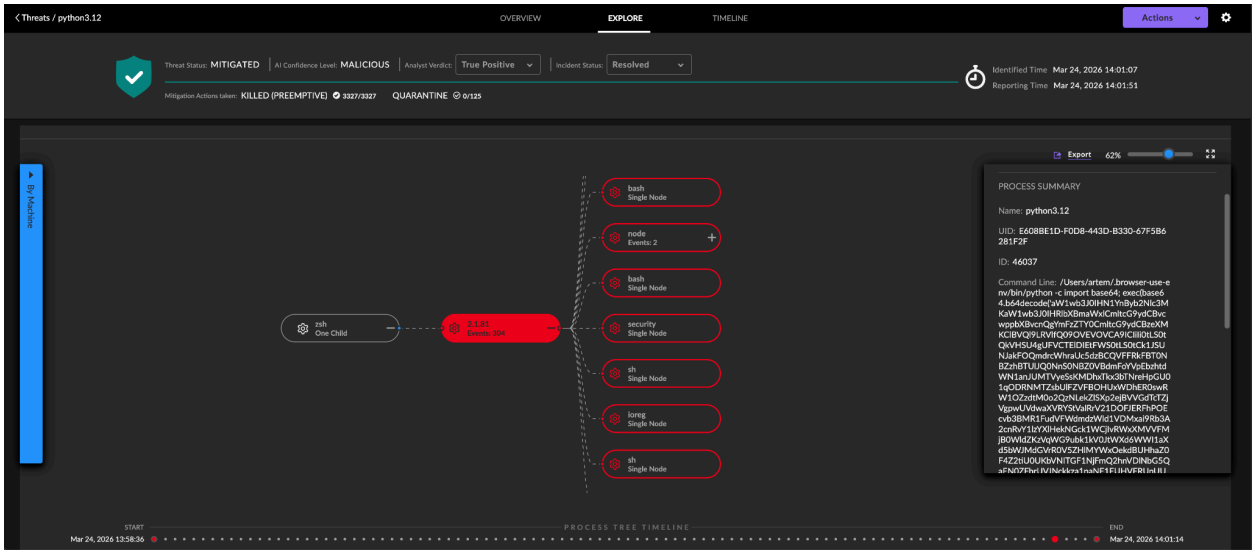
Here is what we detected, how the attack was structured, and why this is the class of threat that the Singularity Platform was built to stop.




**Screenshot 1: Autonomous Detection at Machine Speed**

SentinelOne's macOS agent identified and preemptively killed a malicious process chain originating from Anthropic's Claude Code running with unrestricted permissions (`claude --dangerously-skip-permissions`). No human developer ran `pip install`, an autonomous AI coding assistant updated LiteLLM to the compromised version as part of its normal workflow.

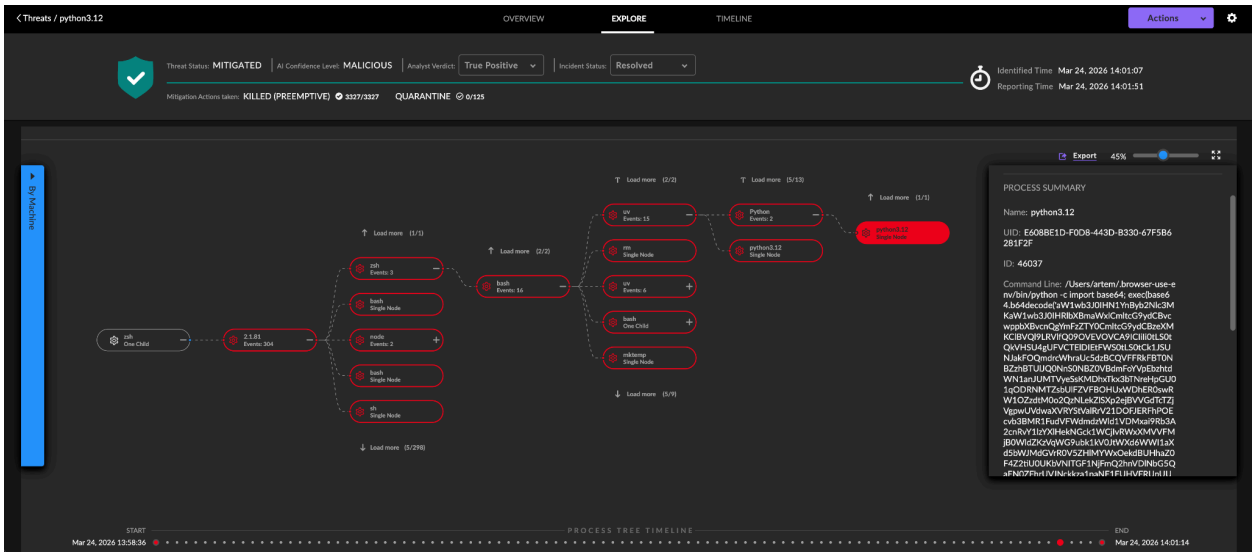
The AI engine classified the behavior as **MALICIOUS** and took immediate action: **KILLED (PREEMPTIVE)** across 424 related events in under 44 seconds. The agent didn't need to know the package was compromised, it watched what the process *did* and stopped it based on behavior, regardless of what initiated the install.




**Screenshot 2: Catching the Payload in the Act**

The macOS agent caught the trojaned LiteLLM package mid-execution. The process summary tells the story: `python3.12` launching with a command line containing `import base64; exec(base64.b64decode(...`, the exact bootstrap mechanism described in the attack's first stage, decoding and executing the obfuscated payload in a child process.

The agent didn't need a signature for this specific package. It recognized the behavioral pattern, a Python interpreter executing base64-decoded code in a spawned subprocess, classified it as **MALICIOUS**, and killed it preemptively before the stealer, persistence, or lateral movement stages could deploy.



**Screenshot 3: The Full Process Tree: Containing the Blast Radius**

Zooming out on the same detection reveals the full scope of what the autonomous AI agent was doing when the payload fired. The process tree expands from Claude Code (2.1.81) into a sprawling chain: `zsh`, `bash`, `node`, `uv`, `ssh`, `rm`, `python3.12`, `mktemp`, with hundreds of child events still loadable (304 events captured). This is what unrestricted AI agent activity looks like at the endpoint level: a single command spawning an entire dependency management workflow that pulled, installed, and attempted to execute the trojaned package.

The SentinelOne macOS agent traced every branch of this tree, correlated the events back to the root cause, and killed the malicious execution; all while preserving the full forensic record for investigation.

## The Compromise Was Indirect. That's What Makes It Dangerous.

The attacker, operating under the alias TeamPCP, never attacked LiteLLM directly. They first compromised Trivy, a widely trusted open-source security scanner, on March 19. From there, they obtained the LiteLLM maintainer's PyPI credentials and used them to publish two malicious versions: 1.82.7 and 1.82.8.

A security tool, built to find vulnerabilities, became the vector that enabled the compromise of an AI infrastructure package used by thousands of organizations. The same actor went on to compromise Checkmarx KICS and AST on March 23, and Telnyx on March 27. This was not a smash-and-grab. It was a coordinated campaign that exploited the transitive trust woven through open-source supply chains.

For security leaders asking, "Could this have reached us?" the more pressing question is: "How fast could we have answered that?"

## A New Attack Surface: AI Agents With Unrestricted Permissions

In one customer environment, SentinelOne detected the infection arriving through an unexpected vector: an AI coding assistant running with unrestricted system permissions autonomously updated LiteLLM to the trojaned version without human review. The update pulled the infected package, and the payload attempted to execute. Our agent blocked it.

This is a new class of attack surface that most organizations have not yet scoped. AI coding agents operating with full system permissions can become unwitting vectors for supply chain compromises. The speed and automation that make these tools valuable are the same properties that make them dangerous when the packages they pull have been weaponized. Organizations that have not yet established governance policies for AI assistant permissions are carrying risks they cannot see.

SentinelOne's behavioral detection operates below the application layer. It does not matter whether a malicious package is installed by a human, a CI pipeline, or an AI agent. The platform monitors process behavior via the Endpoint Security Framework, which is why this detection fired regardless of how the infected package arrived.

## Two Infection Vectors, One Designed to Run Without You

Version 1.82.7 embedded its payload in `proxy_server.py`, which executes every time the `litellm.proxy` module is imported. For anyone using LiteLLM as a proxy layer for LLM API calls, this fires constantly during normal operations.

Version 1.82.8 escalated. The attacker placed the payload in a `.pth` file, `litellm_init.pth`. Files with the `.pth` extension are processed by the Python interpreter at startup, regardless of which modules are imported. Any Python script running on a system with this version installed would trigger the malicious code, even if that script had nothing to do with LiteLLM.

If version 1.82.7 was a targeted shot, version 1.82.8 was a blast radius expansion. The attacker removed the requirement that the victim actually use the compromised library.

## What the Payload Did Once Inside

The attack was structured as a multi-stage delivery system, each stage decoding, decrypting, and executing the next. The first stage was a minimal bootstrap, a single line of base64-decoded Python launched in a detached subprocess with stdout and stderr suppressed. Lightweight enough to slip past signature-based tools. Quiet enough to avoid raising flags.

The second stage was a comprehensive data stealer. It harvested system and user information, cryptocurrency wallets, cloud credentials, application secrets, and system configurations. For practitioners wondering what the blast radius looks like if a developer workstation is compromised, this is the answer: the attacker collects everything needed to move from a laptop to production infrastructure.

The third stage established persistence through a systemd user service at `~/.config/systemd/user/sysmon.service`, executing a script at `~/.config/sysmon/sysmon.py`. The naming convention, "sysmon," was deliberately chosen to mimic legitimate system monitoring tools. It is designed to survive casual inspection and blend into environments where dozens of services run as expected background noise. This is precisely the kind of evasion that signature-based detection misses and behavioral AI catches: the process looks normal until you observe what it actually does.

The persistence mechanism included a 5-minute initial delay before any network activity, a technique specifically designed to outlast automated sandbox analysis. After that, the script contacted its C2 server every 50 minutes, fetching dynamic payload URLs. This sparse

communication pattern makes behavioral detection through network monitoring significantly harder, and gives the attacker the ability to push new tooling without ever re-compromising the target.

## It Moved Laterally Through Kubernetes

The attack did not stop at the workstation. It created privileged pods across Kubernetes cluster nodes in the `kube-system` namespace, using standard container images like `alpine:latest`, with `hostPID`, `hostNetwork`, and a privileged security context. By mounting the host filesystem directly, these pods gained root-level access to underlying nodes.

Each pod deployed persistent backdoors as systemd services on the host system. The pods operated in legitimate namespaces, used standard images, and ran with privileges that many production workloads legitimately require. For SOC practitioners asking whether their admission control and runtime detection would have caught this: the attack was designed specifically so they might not. Detecting this requires runtime visibility into container behavior after deployment, not just policy enforcement at the admission gate. This is the difference between cloud security that checks configuration and cloud security that watches execution.

## Exfiltration Was Encrypted and Camouflaged

Stolen data was encrypted using a hybrid RSA and AES-256-CBC scheme. A random 32-byte AES session key encrypted the data, then the session key itself was protected with RSA encryption using a hardcoded public key. This meant the malware could encrypt and exfiltrate without first communicating with the C2 server. The encrypted payload was packaged as `tpcp.tar.gz` and transmitted via a single HTTP POST to `models.litellm.cloud`, a domain chosen to blend with legitimate LiteLLM API traffic and slip past network monitoring that whitelists expected destinations.

## What This Attack Proves

The LiteLLM supply chain compromise is not an anomaly. It is the new pattern: multi-stage, multi-surface, designed to evade manual workflows at every turn. A compromised security tool led to a compromised AI package, which led to data theft, persistence, Kubernetes lateral movement, and encrypted exfiltration, all within a window measured in hours.

SentinelOne detected and blocked this attack autonomously, on the same day it was launched, across multiple customer environments. No manual triage. No signature update. No analyst in the loop for the initial containment. This is what autonomous, AI-native defense looks like when it meets a real-world threat at machine speed.

The gap between the velocity of this attack and the capacity of human-driven investigation is the gap where organizations get compromised. Closing that gap is not a feature request. It is an architectural decision.

## Why This Detection Worked: Architecture, Not Luck

The LiteLLM detection wasn't a one-off. It's what happens when autonomous, behavioral AI is built into the foundation, not bolted on after the fact. The Singularity Platform's visibility across endpoint, cloud, identity, and AI workloads is why the agent saw this regardless of whether the install came from a human, a CI pipeline, or an AI coding assistant.

For teams that need the human expertise layer on top, Wayfinder MDR extends that autonomous detection with 24/7 investigation and response, closing the gap between detection and resolution.

This is the Autonomous Security Intelligence (ASI) framework in practice: AI that acts at machine speed, backed by human expertise when it matters, across every surface the attack See how the Singularity Platform protects AI infrastructure and request a demo today.