Moving Fast and Importing Things

The Software Supply Chain of Agentic AI

César Soto Valero

The 5th KTH Workshop on the Software Supply Chain, 24 April 2026.

~~vibe coding~~

coding

code

Importing Things

code dependencies,
runtime tool ecosystems

Importing Things

code dependencies,
runtime tool ecosystems,
persistence layers,
model registries,
cloud services,
multiple states,
storage,
deployment,
integration surfaces,
…

“The application code could be small, but
the trust boundary is enormous”

Accountability

A COMPUTER

CAN NEVER BE HELD ACCOUNTABLE

THEREFORE A COMPUTER MUST NEVER

MAKE A MANAGEMENT DECISION

IBM - 1979

claude-code-fork Public

forked from DonutShinobu/claude-code-fork

Pin Watch 0 Fork 0 Star 0

main 2 Branches 0 Tags Go to file T Add file Code

This branch is up to date with DonutShinobu/claude-code-fork:main. Contribute Sync fork

instructkr Publish Claude Code source snapshot from a single baseline commit a99de1b · 3 weeks ago 1 Commit

src	Publish Claude Code source snapshot from a single baseli...	3 weeks ago
README.md	Publish Claude Code source snapshot from a single baseli...	3 weeks ago

README

Claude Code — Leaked Source (2026-03-31)

> **On March 31, 2026, the full source code of Anthropic's Claude Code CLI was leaked** via a `.map` file exposed in their npm registry.

How It Leaked

About

Claude Code is an agentic coding tool that lives in your terminal, understands your codebase, and helps you code faster by executing routine tasks, explaining complex code, and handling git workflows - all through natural language commands. All original source code is the property of Anthropic.

x.com/Fried_rice/status/2038894...

Readme

Activity

0 stars

0 watching

0 forks

Releases

No releases published

Create a new release

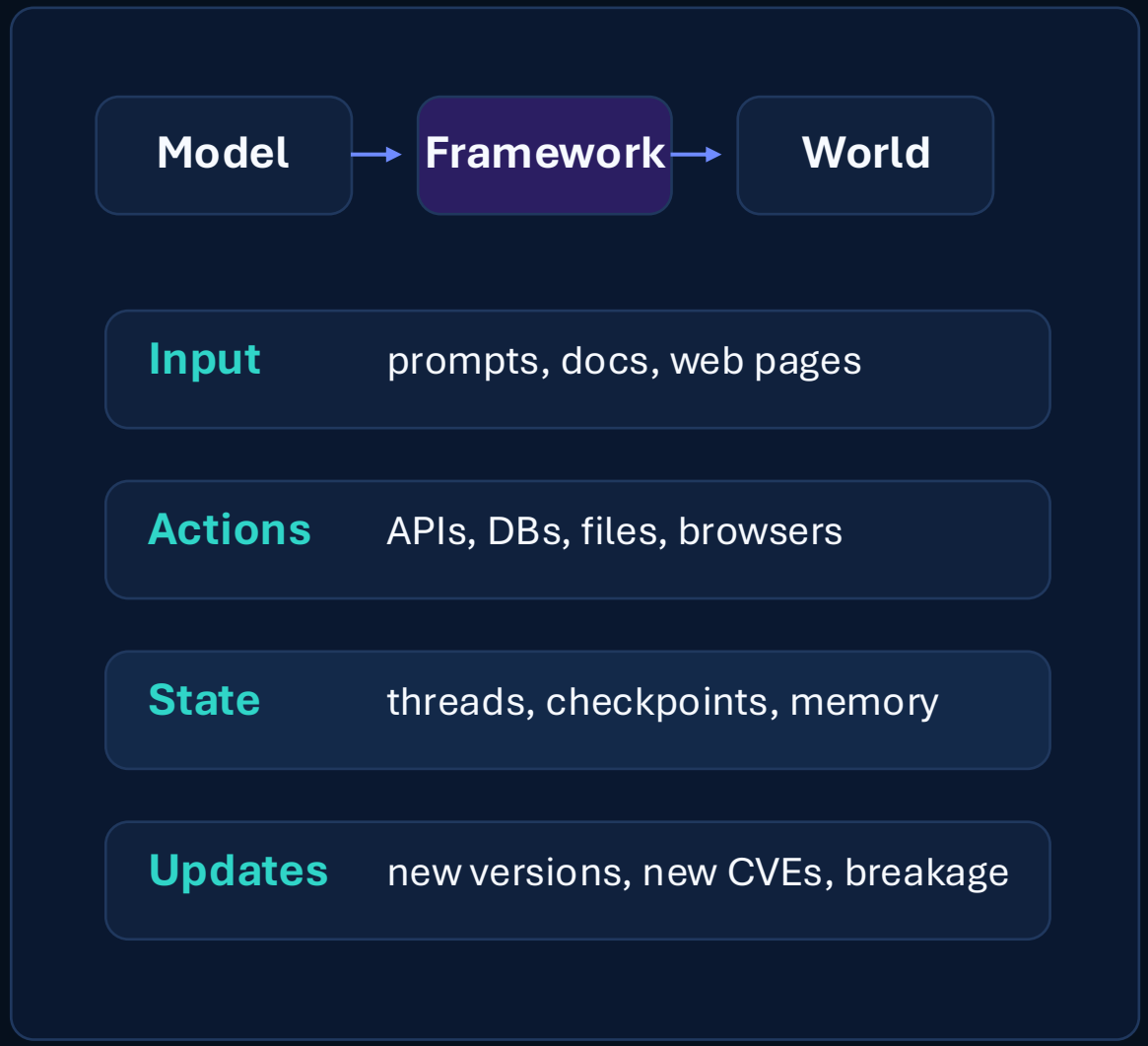
Boris Cherny
@bcherny

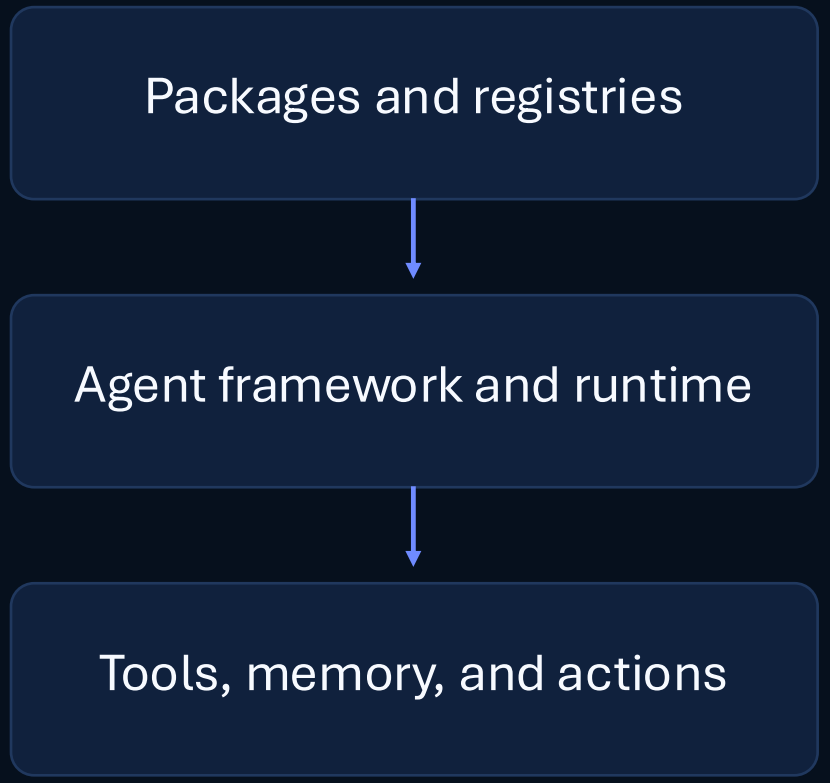
👋 it was human error. Our deploy process has a few manual steps, and we didn't do one of the steps correctly. We have landed a few improvements and are digging in to add more sanity checks.

Like with any other incident, the counter-intuitive answer is to solve the problem by finding ways to go faster, rather than introducing more process. In this case more automation & claude checking the results.

7:04 AM · Apr 1, 2026 · **528.4K** Views

Agentic Frameworks


Model
Framework
World
Input
prompts, docs, web pages
Actions
APIs, DBs, files, browsers
State
threads, checkpoints, memory
Updates
new versions, new CVEs, breakage


Packages and registries
Agent framework and runtime
Tools, memory, and actions

Upstream composition risks

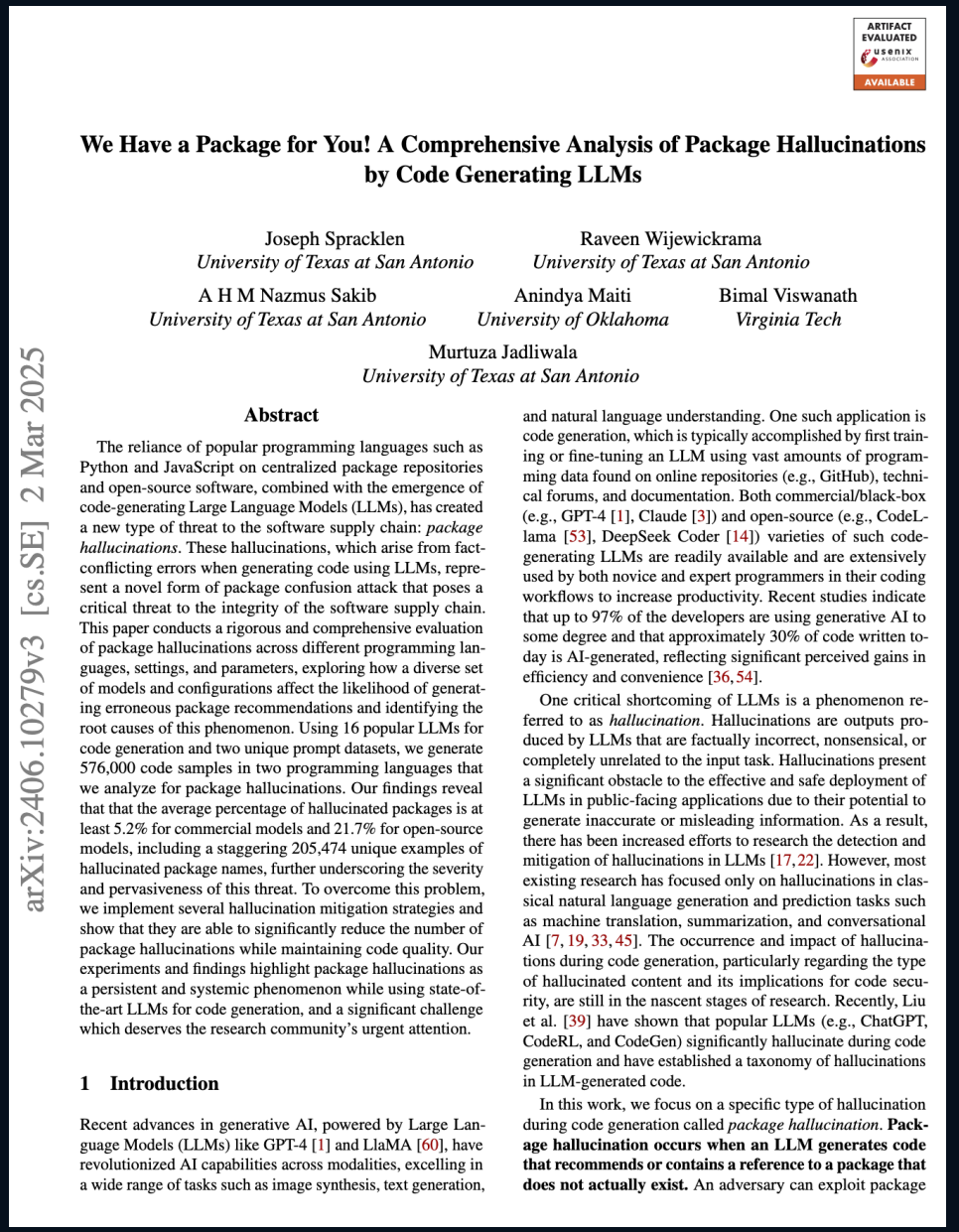
Packages and registries

↓

Agent framework and runtime

↓

Tools, memory, and actions



ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs

Joseph Spracklen
University of Texas at San Antonio

Raveen Wijewickrama
University of Texas at San Antonio

A H M Nazmus Sakib
University of Texas at San Antonio

Anindya Maiti
University of Oklahoma

Bimal Viswanath
Virginia Tech

Murtuza Jadliwala
University of Texas at San Antonio

Abstract

The reliance of popular programming languages such as Python and JavaScript on centralized package repositories and open-source software, combined with the emergence of code-generating Large Language Models (LLMs), has created a new type of threat to the software supply chain: *package hallucinations*. These hallucinations, which arise from fact-conflicting errors when generating code using LLMs, represent a novel form of package confusion attack that poses a critical threat to the integrity of the software supply chain. This paper conducts a rigorous and comprehensive evaluation of package hallucinations across different programming languages, settings, and parameters, exploring how a diverse set of models and configurations affect the likelihood of generating erroneous package recommendations and identifying the root causes of this phenomenon. Using 16 popular LLMs for code generation and two unique prompt datasets, we generate 576,000 code samples in two programming languages that we analyze for package hallucinations. Our findings reveal that that the average percentage of hallucinated packages is at least 5.2% for commercial models and 21.7% for open-source models, including a staggering 205,474 unique examples of hallucinated package names, further underscoring the severity and pervasiveness of this threat. To overcome this problem, we implement several hallucination mitigation strategies and show that they are able to significantly reduce the number of package hallucinations while maintaining code quality. Our experiments and findings highlight package hallucinations as a persistent and systemic phenomenon while using state-of-the-art LLMs for code generation, and a significant challenge which deserves the research community's urgent attention.

1 Introduction

Recent advances in generative AI, powered by Large Language Models (LLMs) like GPT-4 [1] and LlaMA [60], have revolutionized AI capabilities across modalities, excelling in a wide range of tasks such as image synthesis, text generation, and natural language understanding. One such application is code generation, which is typically accomplished by first training or fine-tuning an LLM using vast amounts of programming data found on online repositories (e.g., GitHub), technical forums, and documentation. Both commercial/black-box (e.g., GPT-4 [1], Claude [3]) and open-source (e.g., CodeLlama [53], DeepSeek Coder [14]) varieties of such code-generating LLMs are readily available and are extensively used by both novice and expert programmers in their coding workflows to increase productivity. Recent studies indicate that up to 97% of the developers are using generative AI to some degree and that approximately 30% of code written today is AI-generated, reflecting significant perceived gains in efficiency and convenience [36, 54].

One critical shortcoming of LLMs is a phenomenon referred to as *hallucination*. Hallucinations are outputs produced by LLMs that are factually incorrect, nonsensical, or completely unrelated to the input task. Hallucinations present a significant obstacle to the effective and safe deployment of LLMs in public-facing applications due to their potential to generate inaccurate or misleading information. As a result, there has been increased efforts to research the detection and mitigation of hallucinations in LLMs [17, 22]. However, most existing research has focused only on hallucinations in classical natural language generation and prediction tasks such as machine translation, summarization, and conversational AI [7, 19, 33, 45]. The occurrence and impact of hallucinations during code generation, particularly regarding the type of hallucinated content and its implications for code security, are still in the nascent stages of research. Recently, Liu et al. [39] have shown that popular LLMs (e.g., ChatGPT, CodeRL, and CodeGen) significantly hallucinate during code generation and have established a taxonomy of hallucinations in LLM-generated code.

In this work, we focus on a specific type of hallucination during code generation called *package hallucination*. **Package hallucination occurs when an LLM generates code that recommends or contains a reference to a package that does not actually exist.** An adversary can exploit package

Packages and registries

Agent framework and runtime

Tools, memory, and actions

Tool-integration runtime risks

AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents

Edoardo Debenedetti[1*] **Jie Zhang**[1] **Mislav Balunovic**[1,2]
Luca Beurer-Kellner[1,2] **Marc Fischer**[1,2] **Florian Tramèr**[1]

[1]ETH Zurich [2]Invariant Labs

Abstract

AI agents aim to solve complex tasks by combining text-based reasoning with external tool calls. Unfortunately, AI agents are vulnerable to prompt injection attacks where data returned by external tools hijacks the agent to execute malicious tasks. To measure the adversarial robustness of AI agents, we introduce AgentDojo, an evaluation framework for agents that execute tools over untrusted data. To capture the evolving nature of attacks and defenses, AgentDojo is not a static test suite, but rather an extensible environment for designing and evaluating new agent tasks, defenses, and adaptive attacks. We populate the environment with 97 realistic tasks (e.g., managing an email client, navigating an e-banking website, or making travel bookings), 629 security test cases, and various attack and defense paradigms from the literature. We find that AgentDojo poses a challenge for both attacks and defenses: state-of-the-art LLMs fail at many tasks (even in the absence of attacks), and existing prompt injection attacks break some security properties but not all. We hope that AgentDojo can foster research on new design principles for AI agents that solve common tasks in a reliable and robust manner.

https://agentdojo.spylab.ai

1 Introduction

Large language models (LLMs) have the ability to understand tasks described in natural language and generate plans to solve them [20, 27, 49, 60]. A promising design paradigm for AI *agents* [65] is to combine an LLM with tools that interact with a broader environment [14, 35, 40, 47, 51, 53, 55, 69]. AI agents could be used for various roles, such as digital assistants with access to emails and calendars, or smart "operating systems" with access to coding environments and scripts [24, 25].

However, a key security challenge is that LLMs operate directly on *text*, lacking a formal way to distinguish instructions from data [44, 74]. *Prompt injection attacks* exploit this vulnerability by inserting new malicious instructions in third-party data processed by the agent's tools [17, 44, 62]. A successful attack can allow an external attacker to take actions (and call tools) on behalf of the user. Potential consequences include exfiltrating user data, executing arbitrary code, and more [18, 23, 33, 42].

To measure the ability of AI agents to safely solve tasks in adversarial settings when prompt injections are in place, we introduce *AgentDojo*, a dynamic benchmarking framework which we populate–as a first version–with 97 realistic tasks and 629 security test cases. As illustrated in Figure 1, AgentDojo

*Correspondence to edoardo.debenedetti@inf.ethz.ch

38th Conference on Neural Information Processing Systems (NeurIPS 2024) Track on Datasets and Benchmarks.

Packages and registries

↓

Agent framework and runtime

↓

Tools, memory, and actions

Cross-cutting AI supply chain analysis

Securing the AI Supply Chain: What Can We Learn From Developer-Reported Security Issues and Solutions of AI Projects?

The Anh Nguyen
School of Computer Science and Information Technology, Adelaide University
Adelaide, Australia
theanh.nguyen@adelaide.edu.au

Triet Huynh Minh Le
School of Computer Science and Information Technology, Adelaide University
Adelaide, Australia
triet.h.le@adelaide.edu.au

M. Ali Babar
School of Computer Science and Information Technology, Adelaide University &
Elevexai Systems
Adelaide, Australia
ali.babar@adelaide.edu.au

Abstract

The rapid growth of Artificial Intelligence (AI) models and applications has led to an increasingly complex security landscape. Developers of AI projects must contend not only with traditional software supply chain issues but also with novel, AI-specific security threats. However, little is known about what security issues are commonly encountered and how they are resolved in practice. This gap hinders the development of effective security measures for each component of the AI supply chain. We bridge this gap by conducting an empirical investigation of developer-reported issues and solutions, based on discussions from Hugging Face and GitHub. To identify security-related discussions, we develop a pipeline that combines keyword matching with an optimal fine-tuned `distilBERT` classifier, which achieved the best performance in our extensive comparison of various deep learning and large language models. This pipeline produces a dataset of 312,868 security discussions, providing insights into the security reporting practices of AI applications and projects. We conduct a thematic analysis of 753 posts sampled from our dataset and uncover a fine-grained taxonomy of 32 security issues and 24 solutions across four themes: (1) System and Software, (2) External Tools and Ecosystem, (3) Model, and (4) Data. We reveal that many security issues arise from the complex dependencies and black-box nature of AI components. Notably, challenges related to Models and Data often lack concrete solutions. Our insights can offer evidence-based guidance for developers and researchers to address real-world security threats across the AI supply chain.

CCS Concepts

• **Security and privacy → Software and application security**.

Keywords

Artificial Intelligence, Software Security, Security Vulnerability

ACM Reference Format:
The Anh Nguyen, Triet Huynh Minh Le, and M. Ali Babar. 2026. Securing the AI Supply Chain: What Can We Learn From Developer-Reported Security Issues and Solutions of AI Projects? . In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3744916.3787796

ACM ISBN 979-8-4007-2025-3/2026/04
https://doi.org/10.1145/3744916.3787796

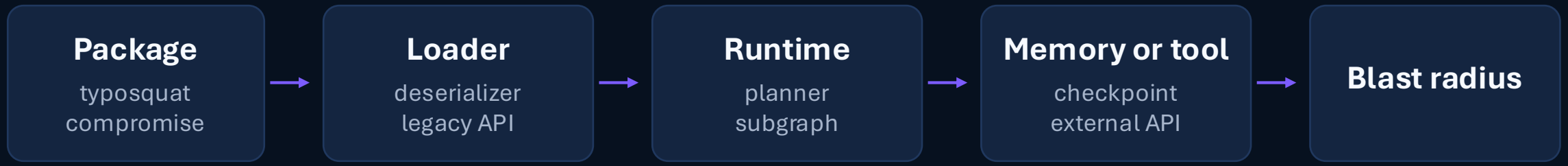
1 Introduction

With the emergence of Artificial Intelligence (AI)-enabled software, developers of these projects face unique security challenges due to the growing complexity of the AI supply chain. They contend with not only the risks inherited from traditional software, e.g., vulnerable dependencies and insecure coding practices [76], but also novel AI-specific threats such as adversarial manipulations [74]. Consequently, multiple high-level risk frameworks, e.g., the OWASP Top 10 for Large Language Models (LLMs) Applications [28], have been developed. These frameworks provide valuable guidance on the theoretical dimensions of AI supply chain risks. However, a significant gap remains between these theoretical frameworks and concrete security problems across different AI supply chain components that developers actually face. Specifically, it is unclear what security issues are commonly encountered, what challenges they pose, and how developers can effectively mitigate or resolve them.

Discussions on open-source platforms such as Hugging Face (HF) and GitHub (GH) can provide insights into real-world security issues in different components of AI-enabled projects. These platforms are central hubs where developers not only share models, code, and datasets but also engage in critical discussions that drive innovation and problem-solving [22, 42]. Security-wise, discussions on these platforms include the security of application, source code, and infrastructure on GH [9, 36, 73], as well as the security of AI models, e.g., model serialization vulnerabilities on HF [71]. An example of such security-related discussions is the Ultralytics supply chain attack report [86], where multiple issues were raised from downstream projects [56, 67, 77] when developers found that the model package was compromised to install crypto miners. This attack demonstrates how a single vulnerability in the AI supply chain can have widespread consequences. The community discussions raised during the attack have proven to be critical data sources in understanding the practical security landscape for developing AI-enabled projects. However, to the best of our knowledge, there is no prior empirical effort to identify and analyze these large-scale discussions to distill real-world security issues that affect the AI supply chain and potential solutions to address these issues.

Our study bridges these gaps by empirically investigating security issues and solutions raised by developers on open-source platforms. We propose a new pipeline combining keyword matching and an optimal fine-tuned `distilBERT` model, selected from a wide

Every new agent capability creates a new trust edge



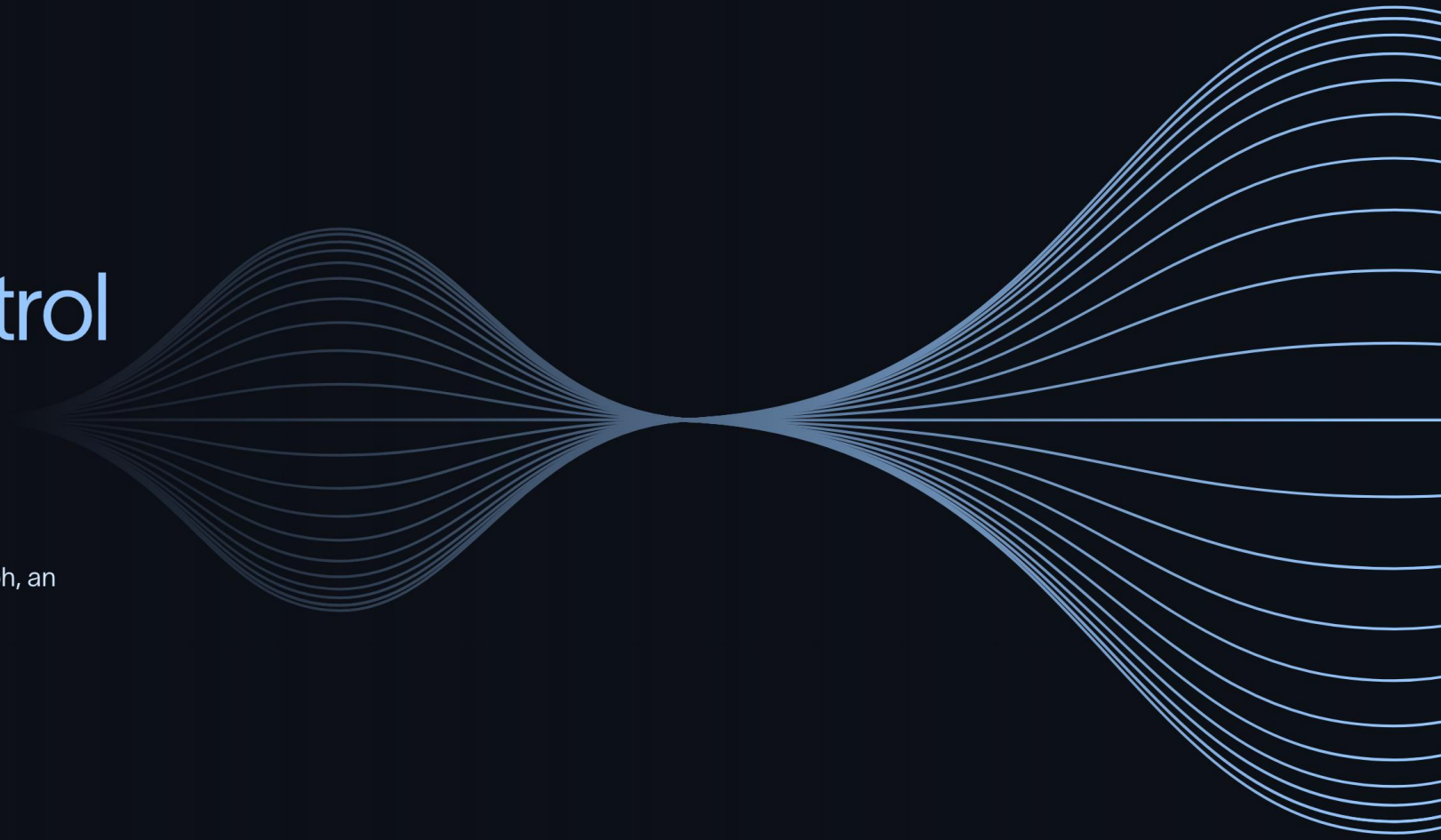
- Tools : code can act, not just answer
- Memory : compromise can persist across runs
- Connectors : pull untrusted content into the reasoning loop
- Packages : make “import” a live security decision

OSS risks + Agentic Properties = Larger Blast Radius

The LangGraph Framework

langgraph

Balance agent control with agency



Design agents that reliably handle complex tasks with LangGraph, an agent runtime and low-level orchestration framework.

Start building
Read the docs

Trusted by companies shaping the future of agents

The LangGraph framework

LangGraph is the orchestration layer between models, tools, memory, and deployment.

Application layer	workflows, prompts, business logic
Framework layer	LangGraph, LangChain, orchestration
Model and data	LLMs, embeddings, vector stores, retrieval
Infrastructure	cloud, queues, runtimes, secrets, registries

What LangGraph itself promises

- Durable execution
- Human in the loop
- Memory and persistence
- Tool calling
- Deployment and SDKs

The safer the framework tries to be for long-lived agents, the more state and infrastructure it must own.

Sources: LangGraph PyPI and documentation (durable execution, memory, HITL, deployment).

Why agent frameworks amplify supply-chain risk

Four design features make agent stacks riskier than plain libraries.

ACTION

Tools and actions

A bad dependency can become a bad action.

STATE

Memory and checkpoints

Persistence increases dwell time.

CONTENT

Connectors and untrusted content

External content can carry instructions, not just data.

AUTONOMY

Autonomy and long-lived loops

One compromise can cascade before detection.

A single LangGraph app already spans a large OSS surface

The application layer looks small in code. The trust boundary is not small.

Representative top-level dependencies

- langgraph
- langchain-openai
- langchain-anthropic
- langchain
- langchain-fireworks
- python-dotenv
- langchain-elasticsearch
- langchain-pinecone

LangGraph itself is split across multiple libraries (checkpoint, prebuilt, SDK, SQLite and Postgres backends, CLI), so the framework already arrives as a mini-supply-chain.

700,387
packages listed in the PyPI ecosystem (Libraries.io snapshot)

8
template deps

7+
repo libraries

The shocking part is not just depth. It is churn, heterogeneity, and the fact that some of these packages call the outside world for you.

Sources: LangChain retrieval-agent template, LangGraph repo dependency map, Libraries.io PyPI package count.

LangGraph case study: the attack surface is not hypothetical

Recent advisories show how framework features map directly to classic software weakness classes.

Recent LangGraph and LangChain issues

Dec 2025	**CVE-2025-67644**	SQL injection in SQLite checkpoint metadata filtering
Nov 2025	**GHSA-wwqv-p2pp-99h5**	RCE in JsonPlusSerializer json mode
Mar 2026	**CVE-2026-28277**	Unsafe msgpack deserialization during checkpoint loading
Mar 2026	**CVE-2026-34070**	Path traversal in legacy prompt loading APIs

Why these bugs matter specifically for agents

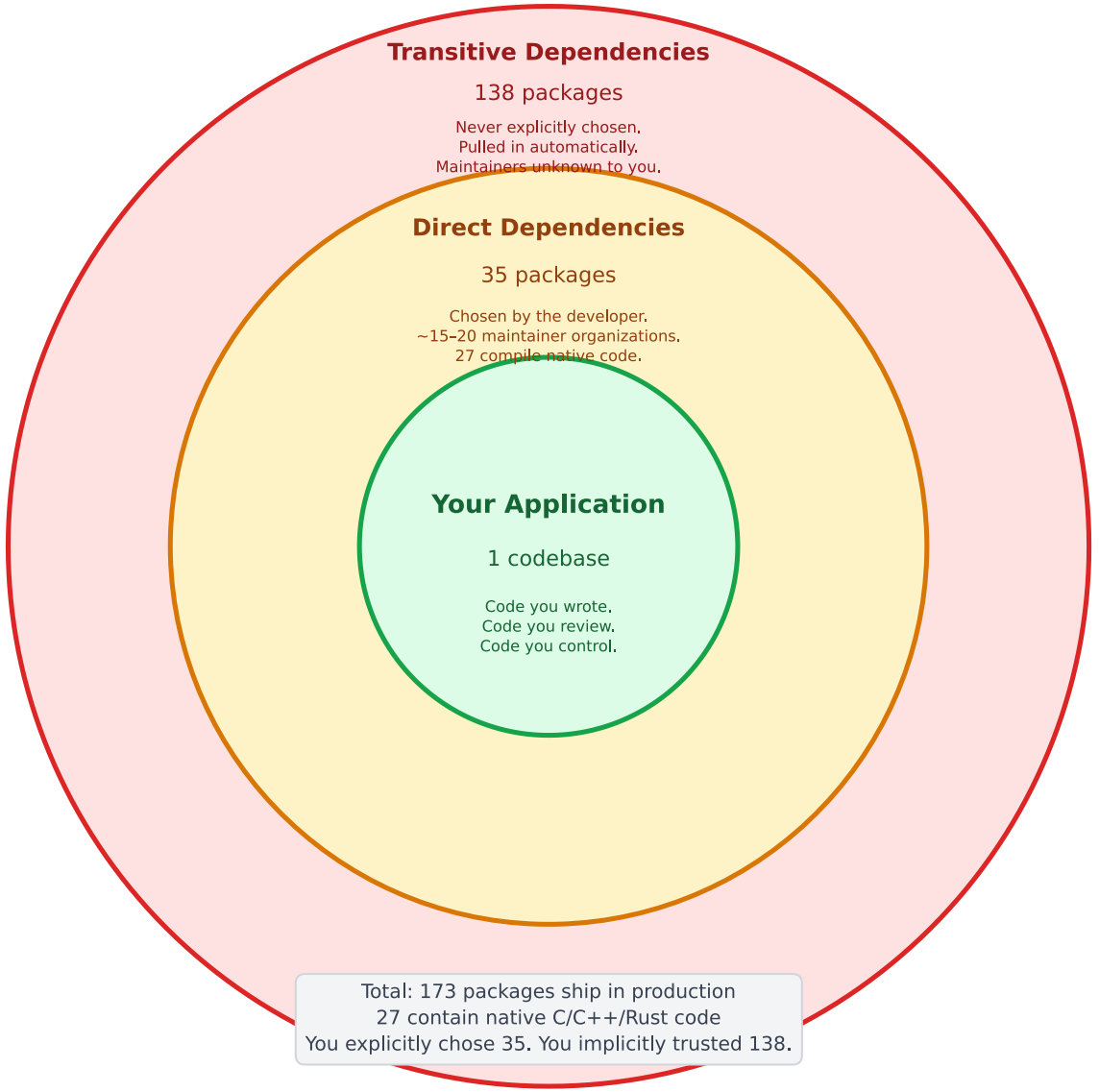
- Checkpoint bugs are agent bugs because checkpoints are part of the control plane.
- Deserialization bugs matter more when agents store tool outputs, memory, and human feedback for replay.
- Path traversal matters because agent apps often load prompts, examples, and local policies dynamically.
- Statefulness and flexibility are features. They are also risk multipliers.

Agent frameworks do not invent new bugs so much as repackage old bugs behind very productive abstractions.

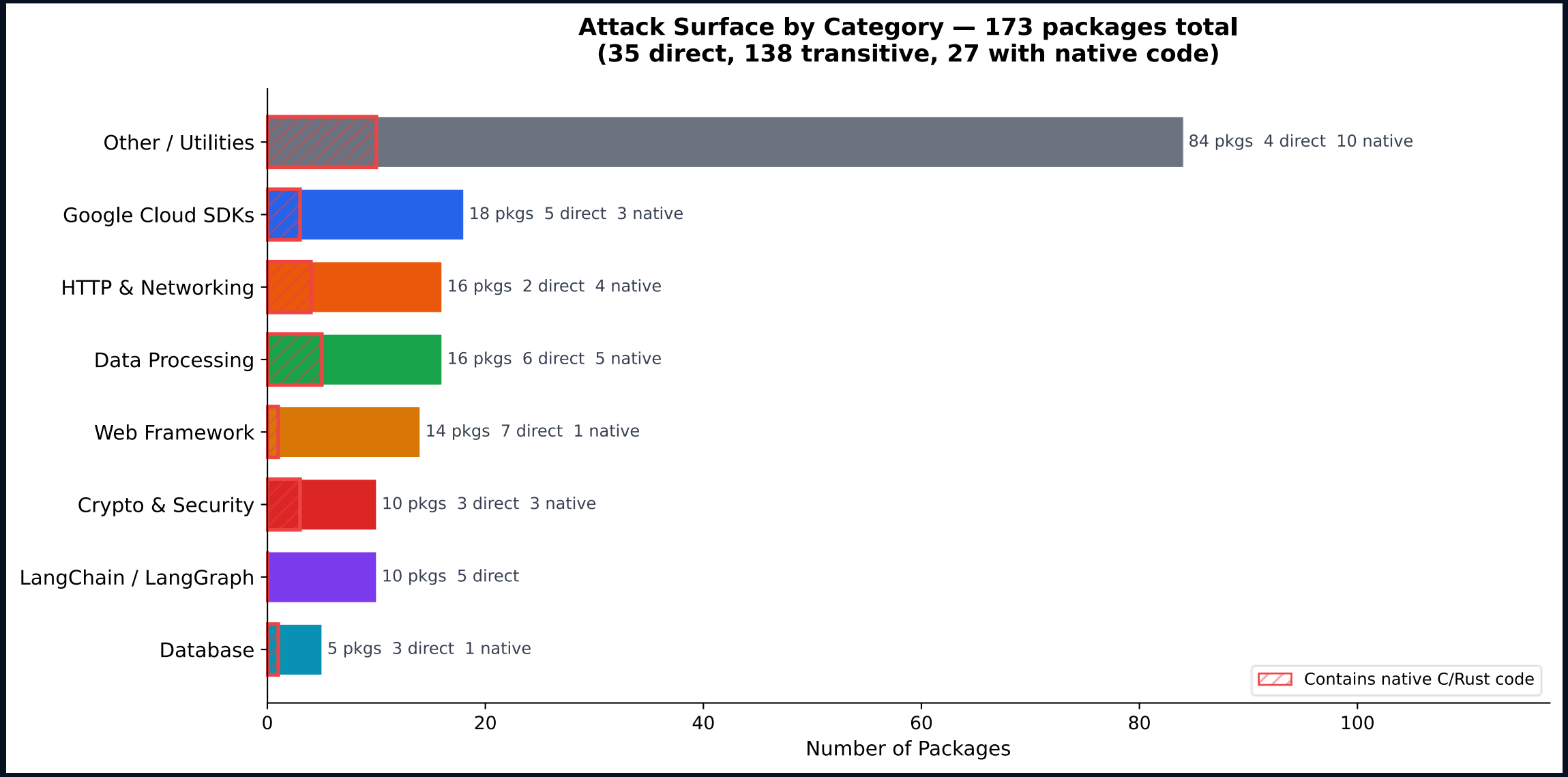
Sources: GitHub, GitLab, and NVD advisories for 2025 to 2026 LangGraph and LangChain issues.

Exploring one of my projects...

Trust Boundaries in Your Agentic AI Supply Chain



Total: 173 packages ship in production
27 contain native C/C++/Rust code
You explicitly chose 35. You implicitly trusted 138.

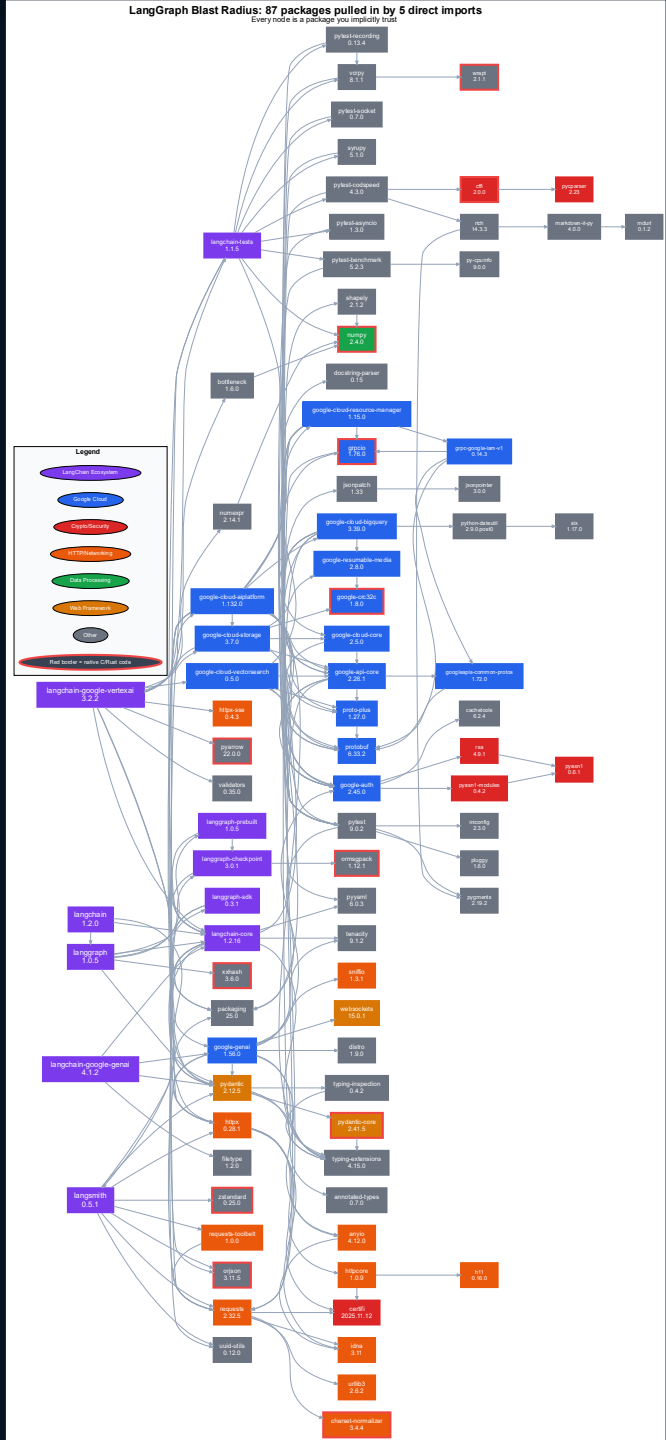

Attack Surface by Category — 173 packages total
(35 direct, 138 transitive, 27 with native code)
Other / Utilities
84 pkgs 4 direct 10 native
Google Cloud SDKs
18 pkgs 5 direct 3 native
HTTP & Networking
16 pkgs 2 direct 4 native
Data Processing
16 pkgs 6 direct 5 native
Web Framework
14 pkgs 7 direct 1 native
Crypto & Security
10 pkgs 3 direct 3 native
LangChain / LangGraph
10 pkgs 5 direct
Database
5 pkgs 3 direct 1 native
Contains native C/Rust code
0
20
40
60
80
100
Number of Packages

LangGraph dependency ~~tree~~ forest

Starting from the 5 imports:

1. langgraph
2. langchain
3. langchain-google-genai
4. langchain-google-vertexai
5. langsmith

87 packages total, max depth = 5




THIS IS FINE

https://github.com/github/awesome-copilot/blob/main/skills/agent-supply-chain/SKILL.md

awesome-copilot / skills / agent-supply-chain / SKILL.md

imran-siddique and Copilot feat: add 3 agent security skills (MCP audit, OWASP compliance, suppl... e95bd8c · 2 weeks ago History

Preview Code Blame 339 lines (277 loc) · 10.5 KB · Raw

name	agent-supply-chain
description	Verify supply chain integrity for AI agent plugins, tools, and dependencies. Use this skill when: - Generating SHA-256 integrity manifests for agent plugins or tool packages - Verifying that installed plugins match their published manifests - Detecting tampered, modified, or untracked files in agent tool directories - Auditing dependency pinning and version policies for agent components - Building provenance chains for agent plugin promotion (dev → staging → production) - Any request like "verify plugin integrity", "generate manifest", "check supply chain", or "sign this plugin"

Agent Supply Chain Integrity

Generate and verify integrity manifests for AI agent plugins and tools. Detect tampering, enforce version pinning, and establish supply chain provenance.

Overview

Agent plugins and MCP servers have the same supply chain risks as npm packages or container images — except the ecosystem has no equivalent of npm provenance, Sigstore, or SLSA. This skill fills that gap.

```
Plugin Directory → Hash All Files (SHA-256) → Generate INTEGRITY.json
                                                    ↓
Later: Plugin Directory → Re-Hash Files → Compare Against INTEGRITY.json
                                                    ↓
                                          Match? VERIFIED : TAMPERED
```

Recommendations

1. Inventory the real graph	Generate SBOMs for the agent app and for runtime-loaded tools, extensions, and models.
2. Pin and verify	Use lockfiles, hashes, trusted registries, signed releases, and provenance when available.
3. Monitor agent state	Harden checkpoint stores, serializers, and memory backends as if they were code-adjacent infrastructure.
4. Isolate the agent	Use least privilege, egress controls, sandboxing, secret scoping, and human approval for risky actions.
5. Ground the automation	Never let an agent select packages or versions from stale model memory alone. Verify against live registries and live security intelligence.

Selected references

Year	Paper	Why it matters	Memorable finding
2024	**InjecAgent**	Tool-integrated agent security	1,054 test cases. ReAct-prompted GPT-4 was vulnerable 24 percent of the time.
2024	**BIPIA**	Indirect prompt injection	LLMs do not reliably separate instructions from external content.
2024	**We Have a Package for You!**	Package hallucinations	Code LLMs can invent package names, creating a slopsquatting entry point.
2025	**AI-Induced Supply-Chain Compromise**	Systematic review	Frames hallucination and slopsquatting as an AI-era supply-chain threat class.
2024	**Instruction Hierarchy**	Mitigation direction	Improves robustness, but reinforces the need for privileged instruction boundaries.
2026	**Securing the AI Supply Chain**	Systematic review	Taxonomy of agentic AI risks inc. systems, tools and ecosystem, model, and data.

cesarsotovalero.net