SkillSpector: A Pre-Publication Security Control for Agent Skills

Nir Paz*†
NVIDIA
npaz@nvidia.com

Keshav Pradeep*
NVIDIA
keshavp@nvidia.com

Narendran Raghavan
NVIDIA
nraghavan@nvidia.com

Ashley Nikirk
NVIDIA
anikirk@nvidia.com

Yashraj Basavaraj Patil
NVIDIA
yashrajbasav@nvidia.com

Mohit Gupta
NVIDIA
mohgupta@nvidia.com

Abstract

Agent skills package procedural knowledge for large language model (LLM) agents in a form that can be discovered and loaded at inference time. They let agents acquire domain-specific workflows without model retraining, but they also create a new software supply-chain surface: a skill may combine natural-language instructions, activation metadata, permission declarations, dependencies, and executable helper code. Existing scanners can inspect code or dependencies, but they rarely reason across model-visible metadata, declared permissions, natural-language instructions, and bundled implementation.

We present SkillSpector, a pre-publication security scanner for agent skills. SkillSpector normalizes a skill bundle into a shared state and runs static pattern detectors, abstract syntax tree (AST) and taint analyzers, manifest consistency checks, metadata poisoning checks, supply-chain checks, optional LLM-based semantic analyzers, and report generation. Its taxonomy is grounded in Open Worldwide Application Security Project (OWASP) guidance for LLM and agentic artificial intelligence (AI) applications [14, 15], MITRE Adversarial Threat Landscape for Artificial-Intelligence Systems (ATLAS) adversarial categories [10], and Open Source Vulnerabilities (OSV.dev) dependency vulnerability data [13]. The implementation exposes terminal, JavaScript Object Notation (JSON), Markdown, and Static Analysis Results Interchange Format (SARIF) output [12]; a 0–100 risk score; and 64 rule patterns across 16 categories. We position deterministic analyzers as the enforcement core and LLM-backed analyzers as advisory signals for human review. Functional validation exercises distinct skill risk surfaces, an unlabeled 178-skill field exercise checks operability on real skill layouts, and a 1,058-trace internal CI/CD analysis characterizes finding distribution and reviewer workload on a benign-skewed skill catalog. We do not claim precision, recall, or ecosystem-scale accuracy without a labeled corpus. The goal is not to prove skill safety, but to make skill risk visible, reviewable, and actionable before publication or installation. The main contribution is a practical control point that treats instructions, metadata, permissions, dependencies, and code as one review target.

CCS Concepts: • **Security and privacy → Software and application security**; • **Computing methodologies →** *Artificial intelligence.*

Keywords: agent skills, agentic AI security, LLM agents, prompt injection, tool poisoning, metadata poisoning, software supply chain, static analysis

1 Introduction

Agent skills are emerging as a practical abstraction for extending LLM agents. The agentskills.io Agent Skills specification defines a skill as a directory containing, at minimum, a `SKILL.md` file, with optional scripts, references, assets, configuration, dependency files, and tool metadata [1]. Skills let teams encode repeatable workflows, teach agents local conventions, and give smaller models access to procedural expertise that would otherwise be unavailable at inference time. Recent work argues that skills are becoming a distinct layer in agent architecture, with progressive disclosure, portability, and Model Context Protocol (MCP) integration as common design themes [19]. SkillsBench further shows why this layer matters: curated skills improved pass rates by 16.2 percentage points across a benchmark of 86 tasks, while self-generated skills did not provide comparable benefit [6].

The same properties that make skills useful also make them security-sensitive. A skill is not only code. It is agent context. Its name, description, triggers, parameter descriptions, warnings, and examples may be read by the model and used to select tools or plan actions. Its scripts may execute with the privileges of the agent process. Its dependency files may pull untrusted packages into the agent environment. Its permission declarations may be the only visible contract between the skill author, the invoking agent, and the user. MCP's own specification warns that tools can represent arbitrary code execution, that tool behavior descriptions should be treated as untrusted unless obtained from a trusted server, and that user consent and control are required for data access and tool invocation [11].

*These authors contributed equally to this work.
†Corresponding author.

Recent research makes this risk concrete. "Agent Skills in the Wild" analyzed 31,132 skills from two marketplaces and found that 26.1% contained at least one vulnerability; skills bundling executable scripts were 2.12 times more likely to be vulnerable than instruction-only skills [9]. A later empirical study behaviorally verified 98,380 skills, confirmed 157 malicious skills with 632 vulnerabilities, and identified data theft and agent hijacking as dominant archetypes [8]. Other work demonstrates skill-specific prompt injection, automated stealthy skill injection, and lifecycle risks spanning creation, distribution, deployment, and execution [3, 7, 16].

This changes the scanner problem. A traditional static application security test may flag `subprocess.run`, vulnerable dependencies, or suspicious network calls, but it usually ignores YAML frontmatter, skill triggers, natural-language instructions, and description-behavior mismatch. A prompt-injection detector may flag obvious instruction overrides but ignore whether a skill silently reads environment variables and posts them to an external endpoint. A dependency scanner may flag CVEs but cannot tell whether a skill requests wildcard permissions or hides instructions in Unicode homoglyphs.

SkillSpector addresses this gap by treating a skill as a compound artifact. The scanner asks a concrete pre-installation question: what risks are visible before this skill is installed, published, or invoked? It answers by combining deterministic analysis over skill text and code with optional LLM-based semantic checks for human review.

This paper makes three contributions:

- We describe a threat model for agent skills that unifies instruction-layer, metadata-layer, permission-layer, code-layer, and dependency-layer risks.
- We present SkillSpector, a systems integration of static-analysis, dependency-analysis, metadata-analysis, and advisory semantic-analysis primitives around the agent-skill abstraction, using a shared graph state and unified finding model.
- We provide a canonical deployment profile, exception-management guidance, and multi-signal correlation patterns that separate enforcement-suitable deterministic findings from advisory semantic findings without making unsupported accuracy claims.

2 Threat Model

We consider skills obtained from public registries, third-party teams, generated outputs, or internal repositories. The defender wants to inspect a skill before installation, before merge into a skill repository, or before approval for enterprise use. We assume the scanner can read the skill bundle but does not execute untrusted code. We also assume the invoking agent may later load natural-language skill content into context and may run bundled helper scripts or MCP tools with user-granted privileges.

SkillSpector targets six risk surfaces.

Natural-language instruction risk. Skill instructions can include direct prompt injection, hidden comments, gradual deception, role-played authority, harmful content, system prompt extraction, or data-leak instructions.

Metadata and trigger risk. Skill names, descriptions, triggers, parameter names, and parameter descriptions influence when a skill activates and how the model interprets tool affordances. These fields can contain hidden HTML comments, zero-width characters, base64 payloads, Unicode homoglyphs, direction overrides, generic triggers, or parameter-level instruction overrides.

Permission risk. Skills may declare no permissions while performing sensitive operations, declare wildcard permissions, request capabilities they do not use, or use capabilities that are not declared. The security issue is not only overprivilege; it is the mismatch between declared intent and actual behavior.

Executable code risk. Helper scripts may read credentials, enumerate files, execute shell commands, invoke dynamic code, persist state, modify their own files, or chain tools in unsafe ways.

Dependency and supply-chain risk. A skill may depend on vulnerable packages, fetch and execute remote scripts, use unpinned versions, include abandoned packages, or reference likely typosquats.

Semantic mismatch risk. A skill's declared purpose may be benign while its implementation does something materially different. Conversely, a raw syntactic pattern may be benign in a context where it is expected. Optional LLM analysis can surface review leads here, but SkillSpector does not treat LLM-only findings as enforcement evidence.

We do not model active exploitation after installation, sandbox escape at runtime, or full malware detonation. SkillSpector is a pre-installation scanner and auditor, not an active red-team harness. We also do not claim robustness against adaptive evasion of the current intraprocedural taint engine: helper-function, imported-module, and cross-file exfiltration flows are detection boundaries for the deterministic implementation, even when adjacent static signals may still route the skill to review.

3 Taxonomy Grounding

SkillSpector's rule taxonomy is grounded in existing AI-security guidance rather than in an isolated scanner-specific vocabulary. The OWASP Top 10 for LLM Applications 2025 final document identifies prompt injection, sensitive information disclosure, supply-chain risk, improper output handling, excessive agency, system prompt leakage, vector and embedding weaknesses, misinformation, and

unbounded consumption as core LLM application risks [14]. Several of these categories map directly onto skill artifacts. OWASP notes that prompt injections do not need to be human-visible if the model can parse them; this motivates hidden-instruction checks over comments, zero-width characters, encoded blobs, and metadata fields. OWASP's emphasis on excessive agency as excessive functionality, permissions, or autonomy motivates SkillSpector's excessive-agency patterns, manifest least-privilege checks, and human-review recommendations. OWASP's supply-chain guidance motivates dependency scanning, remote script detection, and OSV-backed CVE checks.

The agentic layer sharpens these mappings. OWASP's 2026 Agentic Applications guidance focuses on systems that plan, act, and make decisions across tools and workflows [15]. SkillSpector therefore treats a skill as a pre-publication control point for a catalog or registry: provenance is necessary, but safety also depends on whether the artifact's instructions, triggers, permissions, tool descriptions, and bundled code align with its stated purpose. MITRE ATLAS provides a complementary adversarial taxonomy for AI systems [10], and SkillSpector findings carry framework tags where the implementation has a clear mapping, such as ASI02 for tool misuse and AML.T0080 for AI agent context poisoning.

The same trend appears in MCP-focused work. MCPTox and MCP-ITP show that malicious instructions embedded in tool metadata can manipulate agent behavior, including stealthier variants where the poisoned tool need not be invoked directly [5, 18]. Recent MCP threat modeling similarly finds tool poisoning to be a prevalent and high-impact client-side vulnerability and recommends static metadata analysis, decision-path tracking, behavioral anomaly detection, and user transparency as complementary defenses [2]. These findings support SkillSpector's choice to scan tool descriptions, parameter descriptions, Unicode deception, hidden instructions, and description-behavior mismatch before a skill is published or installed.

4 System Design

SkillSpector is implemented as a command-line scanner backed by a LangGraph workflow. The CLI maps command arguments into graph state, invokes the workflow, writes the selected output format, and can return a nonzero exit code for policy-configured high-risk scans. The graph has five main phases:

Resolve input. The scanner accepts local directories, single files, zip archives, Git repositories, and URLs. Non-directory inputs are normalized into a temporary local skill directory.

Build context. The scanner walks the skill directory, skips common irrelevant directories such as `.git` and virtual environments, reads text files into a file cache, extracts `SKILL.md` frontmatter into a manifest, records component metadata, and marks whether executable scripts are present.

Run analyzers. Analyzer nodes consume the same state and append findings. Most analyzers can run independently after context construction. The current graph includes static pattern analyzers, YARA signature scanning [17], Python AST analysis, lightweight taint tracking, MCP least-privilege checks, MCP tool-poisoning checks, and optional semantic analyzers.

Consolidate and validate findings. The meta-analyzer consumes analyzer findings before report generation, correlates them with file and manifest context, suppresses likely false positives, and adds remediation context. This phase preserves the separation between raw analyzer output and the reportable finding set, while retaining deterministic and LLM-backed signals with provenance so enforcement can distinguish evidence classes.

Report. The report node computes a bounded 0–100 risk score, emits a recommendation, and renders terminal, JSON, Markdown, or SARIF 2.1.0 output. SARIF support makes findings consumable by existing code-scanning and CI systems [12].

The shared finding model contains a rule id, message, severity, confidence, file location, category, matched text or snippet, explanation, remediation, and optional framework tags. This common shape lets heterogeneous analyzers participate in the same report without separate post-processing paths.

Deterministic confidence is assigned by rule specificity rather than learned calibration. Credential or environment-variable data flowing to a network sink, hidden metadata instructions, bidirectional controls, and known vulnerable dependencies are treated as high-confidence because benign explanations are narrow. Broader indicators such as subprocess use, file writes, network calls, or dynamic imports are routed to review unless they correlate with other signals.

5 Analyzer Families and Enforcement Posture

SkillSpector separates findings that are suitable for automated gating from findings that should only guide human review. This boundary is part of the design: skill content is adversarial input, and semantic interpretation by an LLM should not become an uncalibrated security oracle.

The LLM analyzer base uses token budgeting, line-numbered prompts, chunking with overlap, rate limiting, structured Pydantic output, and conversion back into standard findings. The prompt template treats skill content as untrusted input, asks the model not to follow instructions inside the skill excerpt, requires line-grounded findings, and returns structured fields for rule id, severity, confidence, explanation, remediation, and location. These controls

Table 1. Risk coverage by external risk family. SkillSpector rule identifiers are defined in Appendix A.

Source risk family	SkillSpector coverage
OWASP LLM01 Prompt Injection	P1–P5 prompt-injection patterns; advisory semantic prompt-injection discovery; TP1 and TP3 metadata injection checks
OWASP LLM02 Sensitive Information Disclosure	E1–E4 data-exfiltration checks; taint flows from environment or files to network sinks; system-prompt leakage checks
OWASP LLM03 Supply Chain	SC1–SC6 dependency, remote script, vulnerability, abandoned package, and typosquatting checks; OSV.dev lookup
OWASP LLM05 Improper Output Handling	OH1–OH3 output validation, cross-context output, and unbounded output checks
OWASP LLM06 Excessive Agency	EA1–EA4 excessive functionality/autonomy/resource checks; LP1–LP4 least-privilege checks; TM1–TM3 tool misuse checks
OWASP LLM07 System Prompt Leakage	P6–P8 direct, indirect, and tool-mediated system prompt leakage checks
OWASP LLM10 Unbounded Consumption	EA4 and OH3 resource-bounding checks; risk-score multiplier for executable skill artifacts
Agentic and MITRE ATLAS-style risks	Tool poisoning, memory poisoning, rogue-agent behavior, and trigger abuse checks

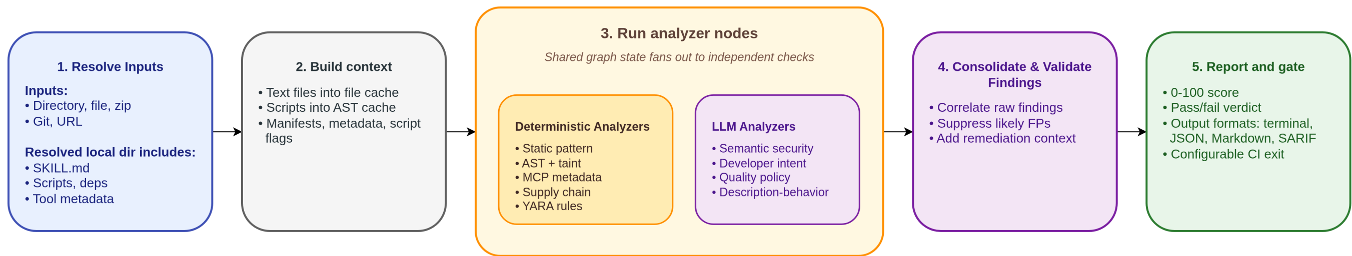

Figure 1. SkillSpector system design: SkillSpector is implemented as a LangGraph workflow over one shared artifact context: it normalizes each skill, builds that context, runs deterministic and LLM analyzers over it, consolidates and validates findings, and emits report models for terminal review, CI integration, and other output formats.

improve operability, but they are not a proof of adversarial robustness. If LLM analysis is disabled, unavailable, or manipulated by skill content, the deterministic pipeline still produces the enforceable report.

6 Functional Validation

We validated SkillSpector on representative fixtures included with the implementation. The goal was to test whether distinct analyzer families exercise distinct risk surfaces while preserving a clean result for a benign docs-only skill. This is functional validation, not an accuracy evaluation: no precision, recall, or false-positive rate should be inferred from these fixtures. All scans below used deterministic mode with LLM analysis disabled.

These cases demonstrate why skill review needs multiple signals. The metadata-poisoning fixture is suspicious because of model-visible metadata, not executable code. The underdeclared-agent fixture requires comparing manifest omission against code behavior, then correlating environment access, shell execution, network output, and unvalidated output handling. The vulnerable-dependencies fixture is primarily a supply-chain problem. SkillSpector reports these cases through one finding model and one risk score, but these examples do not establish detection accuracy.

We also ran SkillSpector over an internal sample of 178 real skills. The sample is not public and has not been manually labeled, so we use it only as a field exercise rather than an evaluation. The field exercise answers a narrower operational question: whether the scanner can process non-fixture skill layouts and produce reports that a reviewer can triage. It should not be read as evidence of precision, recall, or false-positive rate, because unlabeled findings cannot distinguish malicious behavior from benign build automation or expected integration logic. It did, however, expose the expected sensitivity of optional LLM-backed advisory findings to model and configuration choices. For that reason, the deployment profile below excludes LLM-only findings from blocking decisions.

Table 2. Analyzer families: methods, posture, and known limits.

Analyzer family	Method and scope	Default posture	Known limits
Static text and metadata patterns	Match prompt injection, exfiltration, trigger abuse, system-prompt leakage, harmful content, and suspicious metadata in `SKILL.md`, manifests, tool descriptions, and scripts	Enforce for high-confidence literal or encoded indicators; review otherwise	Can miss novel paraphrases, multi-file intent, and heavy obfuscation
Python AST and lightweight taint	Parse Python, flag `exec`, `eval`, `subprocess`, `os.system`, dynamic imports, and simple flows from environment/files/input to network or execution sinks	Enforce for direct exfiltration or dangerous execution chains; review for expected build automation	Intraprocedural and conservative; helper-function, imported-module, and cross-file flows are not fully tracked
MCP least privilege	Compare manifest permissions with detected capabilities such as shell, network, file read/write, environment access, and MCP use	Enforce for wildcard permissions, missing declarations, and underdeclared sensitive capabilities	Cannot prove runtime authorization or intent
MCP tool poisoning	Detect hidden instructions, zero-width characters, base64/data URI payloads, mixed scripts, bidirectional controls, homoglyphs, and parameter-description overrides	Enforce for deterministic hidden-instruction and Unicode deception evidence	Purpose-behavior mismatch remains partly semantic
Supply chain	Inspect dependency files and install instructions; query OSV.dev for known vulnerabilities when available; fall back to offline checks	Enforce for known vulnerable dependencies, remote script execution, and likely typosquats	Offline fallback is incomplete; vulnerability databases lag reality
LLM semantic and meta-analysis	Surface novel prompt injection, gradual deception, policy concerns, and description-behavior mismatch; optionally enrich candidate findings	Advisory only	Model-dependent, prompt-injection exposed, and not calibrated for precision or recall

Table 3. Functional validation fixtures.

Fixture	Scenario	Findings observed	Score	Recommendation
`safe-greeting`	Benign docs-only skill	none	0	`SAFE`
`metadata-poisoning`	Hidden HTML instruction, Cyrillic homoglyph, parameter override	TP1, TP2, P1, P2	100	`DO_NOT_INSTALL`
`underdeclared-agent`	No permissions but code reads env, invokes shell, posts to network	AST4, TT3, LP3, E1, E2, OH1	100	`DO_NOT_INSTALL`
`env-exfiltration`	Environment variable collection sent to external endpoint	E1, E2	58	`DO_NOT_INSTALL`
`vulnerable-dependencies`	Remote install script, vulnerable packages, abandoned package, typosquat	SC2, SC4, SC5, SC6, TM2	100	`DO_NOT_INSTALL`
`wildcard-permissions`	Wildcard permission plus chmod and delete operations	AST4, AST5, LP2, TR2, TM1	100	`DO_NOT_INSTALL`

7 Analysis of Live CI/CD Traces

7.1 Setup

We deployed SkillSpector as part of a larger CI/CD-integrated skills scanning pipeline at a large corporation, where developers iterate on skills until they pass a multi-stage gate. Over the course of this deployment we collected 1,058 LangSmith [4] traces of SkillSpector scans against a live internal skills catalog that is continuously updated as developers contribute new skills and revise existing ones. Each trace captures the full LangGraph state at every analyzer node — raw and consolidated findings, the model and token count for each LLM analyzer call, and the final risk score and recommendation — which lets us decompose any reported result back to the analyzer that produced it.

The catalog is non-public and benign-skewed: skills that reach this pipeline are authored under standard contribution norms and are not adversarially constructed. We therefore

use the corpus to characterize what SkillSpector reports on real internal skill layouts, not to estimate ecosystem prevalence or precision/recall. All scans run with the canonical deterministic profile (Section 8) plus advisory LLM analyzers enabled.

Because the deployment includes downstream stages beyond SkillSpector, the same skill may appear multiple times in the trace set: a developer iterates on a SkillSpector finding, re-scans, and the new scan may still fail a stage that has nothing to do with SkillSpector. We treat this as a confound for any iteration-fix-rate analysis but it does not affect the per-scan finding distributions reported below.

7.2 Finding-Rate and Category Distribution

Findings concentrate heavily in the semantic and permission layers (Figure 2, left). 34.1% of scans surface at least one semantic finding and 14.5% at least one permission finding, while instruction- and metadata-layer findings are rare (0.2% and 1.0% respectively). The intensity panel of the same plot inverts this picture: when an instruction-layer finding does appear, the affected scan averages 5.0 findings, indicating that one such issue is rarely isolated. Permission and metadata findings cluster less heavily (1.03 and 1.00 per affected scan), consistent with discrete misconfigurations rather than systemic prompt-injection authoring.

At the rule level (Figure 2, right), two rules carry most of the volume. SQP-2 fires on 32.8% of scans: it is an advisory semantic quality-policy rule that flags missing user-facing warnings for operations that may affect data, privacy, or system integrity. LP3 fires on 13.7%: it is a deterministic least-privilege rule that fires when a skill’s code uses detectable capabilities (shell, network, file, or environment access) while the manifest declares no permissions. The remaining top-five rules each fire on 8% or fewer scans. The shape — a few rules dominant, a long tail of rare-firing rules — is consistent with a benign-skewed corpus where most skills pass the deterministic gate cleanly while routinely surfacing advisory review leads. It also explains the asymmetry between the two top rules and the enforcement-vs-advisory split from Section 5: the dominant rule (SQP-2) is advisory and contributes only review queue pressure, while the enforcement-grade rule (LP3) fires on a much smaller fraction of scans.

8 Canonical Deployment Profile

SkillSpector is conservative by design. It should make skill risk visible before publication or installation; it should not imply that every finding is a hard rejection. We recommend a canonical enterprise profile as the reproducible baseline: deterministic analyzers are enabled for enforcement; LLM semantic and meta-analyzers are disabled for blocking decisions and may run only as reviewer triage; OSV.dev checks are enabled when network access is available, with documented offline fallback behavior; and CI gates consume SARIF or JSON outputs using the policy below.

Manual exceptions should be explicit review artifacts, not informal overrides. Each exception should be tied to a skill version or content hash, finding identifiers, reviewer identity, business justification, allowed scope, and expiration date. Exceptions should be re-reviewed whenever permissions, scripts, dependencies, trigger metadata, or tool descriptions change.

In an enterprise deployment, the exception register should have an explicit owner. We recommend that application security own the register, that the skill or product owner provides the business justification, and that security architecture or vendor-risk review approve exceptions for third-party or marketplace skills. A hard-fail finding with a compelling business case should not be silently overridden in CI; it should create an auditable escalation record with the content hash, affected rule ids, compensating controls, expiration date, and re-review trigger. For third-party skills, the same record should connect to software bill of materials and vendor-risk workflows so that dependency, provenance, and skill-permission changes can invalidate prior approvals. In regulated environments, the escalation record should also link to change-management artifacts so the approval chain survives personnel transitions and audit review.

The risk score is an engineering policy mechanism, not an empirical research contribution. It is intentionally simple, and the shared finding model supports higher-order review patterns. We recommend treating the multi-signal correlations in Table 5 as priority review queues even before a learned scoring model exists.

Internal experiments on the 178-skill sample showed substantial configuration sensitivity in LLM-backed advisory findings. We therefore do not report those experiments as accuracy evidence and do not recommend LLM-only findings for automated blocking. The reproducible gate is the canonical deterministic profile above.

9 Limitations and Future Work

The current validation is fixture-based, and the 178-skill field exercise is unlabeled and non-public. Neither measures recall or false-positive rate. Our next step is a labeled corpus evaluation over more than 20,000 public skills, supplemented with synthetic malicious variants and curated benign enterprise skills. This evaluation will report per-category precision and recall, measure false-positive pressure on benign build and integration skills, and specifically measure how many known malicious data-theft skills are captured by direct taint, missed by the intraprocedural taint boundary, or still routed to review through adjacent metadata, permission, or dependency signals. It will also include adversarial stress tests for mixed-script homoglyph chains, bidirectional-control

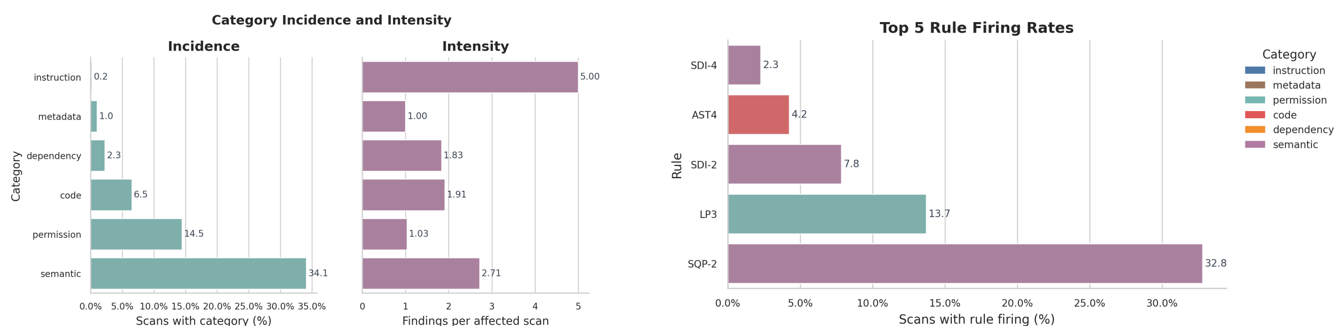

Figure 2. Live CI/CD trace analysis over 1,058 SkillSpector scans. Left: category-level incidence and finding intensity. Right: top-five rule firing rates. Results characterize reviewer workload on a benign-skewed internal catalog and are not precision, recall, or ecosystem-prevalence estimates.

Table 4. Default action policy by finding type.

Finding type	Default action	Rationale
Intraprocedurally detectable environment or file data flowing to network sinks; explicit credential harvesting; system prompt leakage	Hard fail	Direct confidentiality risk
Wildcard permissions, missing permission declarations, or underdeclared sensitive capabilities	Hard fail for catalogs and CI gates; manual exception for trusted internal tools	Breaks least-privilege contract
Remote install scripts, known vulnerable dependencies, likely typosquats, or unpinned high-risk dependencies	Hard fail or dependency-security review	Conventional supply-chain risk
Hidden instructions, zero-width payloads, bidirectional controls, suspicious base64/data URI metadata, or parameter-description overrides	Hard fail	Model-visible deception has little benign justification
Subprocess, file-write, network, or dynamic-import use without other suspicious context	Manual review	Often legitimate for build and integration skills
LLM-only semantic mismatch, vague triggers, policy concerns, or meta-analyzer enrichments	Advisory review only	Useful review lead, not enforcement-grade evidence

Table 5. Priority multi-signal correlation patterns.

Correlation pattern	Signals	Default handling
Underdeclared exfiltration	Missing or underdeclared permissions plus environment/file reads plus network sink	Hard fail unless a reviewer validates benign telemetry
Deceptive activation	Hidden metadata instruction plus broad trigger or parameter override	Hard fail for publication
High-agency helper	Wildcard permissions plus subprocess/file write plus generic activation trigger	Manual review; hard fail for shared catalogs without documented exception
Supply-chain launcher	Remote install script plus unpinned or vulnerable dependency plus executable helper	Hard fail or dependency-security review

stacking, helper-function and imported-module exfiltration, trigger abuse, and description-behavior mismatch.

A second planned direction is dynamic analysis. The current implementation does not execute skills, sandbox runtime behavior, or perform malware detonation. This avoids running malicious artifacts but misses behavior hidden behind runtime branching, remote payloads, generated code, or delayed execution. We plan to add a sandboxed dynamic-analysis mode for high-risk review workflows, with controlled network egress, filesystem observation, environment-variable canaries, process tracing, and comparison between declared permissions and observed runtime behavior. Dynamic results would complement, not replace, the static and semantic pipeline: deterministic static findings remain

the low-latency CI gate, while sandbox execution provides deeper evidence for escalated review.

Version-drift and rug-pull detection are not implemented as finding-producing analyzers. This is an important gap because skills may change behavior across versions while retaining a trusted name or trigger. Planned work includes manifest and code diffing, capability-delta reporting, and hash/provenance integration.

Finally, the risk score is intentionally simple. It adds severity weights, applies an executable-script multiplier, caps at 100, and maps to broad recommendations. Future work should learn or tune score weights from labeled review outcomes, separate exploitability from impact, and add policy profiles for developer preflight, marketplace ingestion, and enterprise approval.

10 Conclusion

Agent skills are becoming a practical distribution format for procedural knowledge, but they are also a new supply-chain and prompt-surface artifact. They need review tools that understand the whole bundle: instructions, metadata, permissions, dependencies, and executable code. SkillSpector provides such a scanner through a multi-stage graph of deterministic analyzers, advisory semantic analyzers, unified findings, SARIF-compatible reporting, and a pre-installation risk recommendation. The central claim is intentionally practical: skill safety cannot be reviewed by code scanning, prompt scanning, or dependency scanning alone. It requires a multi-signal control built for the skill abstraction itself.

A Rule Identifier Legend

This appendix defines identifiers referenced directly or through ranges in this paper; it does not enumerate every SkillSpector rule implemented by the scanner.

P Prompt-injection and harmful-content patterns: P1 instruction override; P2 hidden instructions; P3 external transmission instruction; P4 subtle steering; P5 harmful content; P6 direct system-prompt leakage; P7 indirect system-prompt extraction; P8 tool-mediated system-prompt exfiltration.

E Data-exfiltration patterns: E1 external transmission; E2 environment-variable harvesting; E3 file-system enumeration; E4 conversation-context leakage.

SC Supply-chain patterns: SC1 unpinned dependencies; SC2 remote code execution or install script; SC3 obfuscated code; SC4 known vulnerable dependency; SC5 abandoned dependency; SC6 typosquatting.

EA Excessive-agency patterns: EA1 unrestricted tool access; EA2 autonomous high-impact decision making; EA3 scope creep beyond stated purpose; EA4 unbounded resource consumption.

OH Output-handling patterns: OH1 unvalidated output injection; OH2 cross-context output flow; OH3 unbounded output.

LP Least-privilege checks: LP1 underdeclared capability; LP2 wildcard permission; LP3 missing permission declaration; LP4 overdeclared permission.

TM Tool-misuse patterns: TM1 tool parameter abuse; TM2 chaining abuse; TM3 unsafe defaults.

TP Tool-poisoning metadata checks: TP1 hidden metadata instructions; TP2 Unicode deception; TP3 parameter-description injection.

SQP Semantic quality-policy advisory rules: SQP-2 missing user-facing warning for operations that may affect data, privacy, or system integrity.

SDI Semantic developer-intent advisory rules: SDI-2 context-inappropriate capability; SDI-4 intent-code divergence between comments/docstrings and code behavior.

AST Python abstract-syntax-tree checks: AST4 subprocess call; AST5 `os.system` or OS exec-family call.

TT Taint-tracking checks: TT3 credential or environment variable flow to a network sink.

TR Trigger-abuse checks: TR2 trigger shadows a built-in command or another skill trigger.

OWASP LLM codes LLM01 prompt injection; LLM02 sensitive information disclosure; LLM03 supply chain; LLM05 improper output handling; LLM06 excessive agency; LLM07 system prompt leakage; LLM10 unbounded consumption.

External framework tags ASI02 OWASP Agentic Applications tool misuse; AML.T0080 MITRE ATLAS AI agent context poisoning.

References

[1] Agent Skills. n.d.. Specification. https://agentskills.io/specification

[2] C. Huang, X. Huang, N. P. Tran, and A. M. Fard. 2026. Model Context Protocol Threat Modeling and Analyzing Vulnerabilities to Prompt Injection with Tool Poisoning. arXiv:2603.22489 https://arxiv.org/abs/2603.22489

[3] X. Jia, J. Liao, S. Qin, J. Gu, W. Ren, X. Cao, Y. Liu, and P. Torr. 2026. SkillJect: Automating Stealthy Skill-Based Prompt Injection for Coding Agents with Trace-Driven Closed-Loop Refinement. arXiv:2602.14211 https://arxiv.org/abs/2602.14211

[4] LangChain. n.d.. LangSmith Observability. https://docs.langchain.com/langsmith/observability

[5] R. Li, Z. Wang, Y. Yao, and X.-Y. Li. 2026. MCP-ITP: An Automated Framework for Implicit Tool Poisoning in MCP. arXiv:2601.07395 https://arxiv.org/abs/2601.07395

[6] X. Li, W. Chen, Y. Liu, S. Zheng, X. Chen, Y. He, et al. 2026. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. arXiv:2602.12670 https://arxiv.org/abs/2602.12670

[7] Z. Li, J. Wu, X. Ling, X. Cui, and T. Luo. 2026. Towards Secure Agent Skills: Architecture, Threat Taxonomy, and Security Analysis. arXiv:2604.02837 https://arxiv.org/abs/2604.02837

[8] Y. Liu, Z. Chen, Y. Zhang, G. Deng, Y. Li, J. Ning, Y. Zhang, and L. Y. Zhang. 2026. Malicious Agent Skills in the Wild: A Large-Scale Security Empirical Study. arXiv:2602.06547 https://arxiv.org/abs/2602.06547

[9] Y. Liu, W. Wang, R. Feng, Y. Zhang, G. Xu, G. Deng, Y. Li, and L. Zhang. 2026. Agent Skills in the Wild: An Empirical Study of Security Vulnerabilities at Scale. arXiv:2601.10338 https://arxiv.org/abs/2601.10338

[10] MITRE. n.d.. MITRE ATLAS. https://atlas.mitre.org/

[11] Model Context Protocol. 2025. Specification, version 2025-06-18. https://modelcontextprotocol.io/specification/2025-06-18

[12] OASIS. 2020. Static Analysis Results Interchange Format (SARIF) Version 2.1.0. https://docs.oasis-open.org/sarif/sarif/v2.1.0/os/sarif-v2.1.0-os.html OASIS Standard.

[13] OSV.dev. n.d.. Introduction to OSV. https://google.github.io/osv.dev/

[14] OWASP GenAI Security Project. 2025. OWASP Top 10 for LLM Applications 2025. https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/

[15] OWASP GenAI Security Project. 2026. OWASP Top 10 for Agentic Applications for 2026. https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/

[16] D. Schmotz, S. Abdelnabi, and M. Andriushchenko. 2025. Agent Skills Enable a New Class of Realistic and Trivially Simple Prompt Injections. arXiv:2510.26328 https://arxiv.org/abs/2510.26328

[17] VirusTotal. n.d.. YARA Documentation. https://yara.readthedocs.io/en/stable/

[18] Z. Wang, Y. Gao, Y. Wang, S. Liu, H. Sun, H. Cheng, G. Shi, H. Du, and X. Li. 2025. MCPTox: A Benchmark for Tool Poisoning Attack on Real-World MCP Servers. arXiv:2508.14925 https://arxiv.org/abs/2508.14925

[19] R. Xu and Y. Yan. 2026. Agent Skills for Large Language Models: Architecture, Acquisition, Security, and the Path Forward. arXiv:2602.12430 https://arxiv.org/abs/2602.12430