How to audit an AI agent skill

The 7-check framework we use on every skill that ships in SkillVault. Free to redistribute with attribution.

TL;DR

This is the full methodology we use to audit AI agent skills (Claude Code, Cursor, Codex CLI, Gemini Code Assist) before we ship them in the SkillVault bundle. It is the same 7-check framework that surfaced 146 rejects when we ran a 187-skill pass in May 2026.

Read this if:

- You are a solo developer who installs skills from random GitHub repos and wants to do that more safely.
- You are a security engineer on a small team being asked "are these skills OK to use."
- You are building your own skills and want to ship something that would pass an audit.

The methodology is free. The 41 skills we audited and pre-cleared are in SkillVault for $99 lifetime if you do not have time to do this work yourself. Either path is valid.

Table of contents

Why this framework exists

In February 2026 Snyk published the ToxicSkills audit. They scanned 3,984 skills from ClawHub and skills.sh and found:

- 13.4% contained critical-level issues
- 36% carried prompt-injection payloads
- 91% of confirmed malware combined natural-language jailbreaks with executable shell payloads

A month later, Anthropic accidentally shipped a debugging sourcemap for Claude Code v2.1.88 to npm exposing 512,000 lines of TypeScript. The leak confirmed that `bashSecurity.ts` has 23 numbered security checks, each one reportedly added in response to a real incident. A documented `CLAUDE.md` prompt injection technique was shown to generate a 50+ subcommand pipeline that bypasses standard deny rules.

If you install a skill without auditing it, you are accepting a non-trivial probability that the skill can read your environment variables, exfiltrate `~/.ssh/` keys, or execute a payload that bypasses your tool deny list. This framework is how you mitigate that.

What you need before you start

Set up once, use for every audit.

- A terminal you can pipe `cat -v` and `xxd` into.
- A Python or Node environment for dependency analysis.
- A scratch directory at `~/skill-audits/` to keep your audit notes per skill.
- A text editor that can show invisible unicode (most editors hide it by default; use VS Code with "Render Whitespace: all" or a hex view).
- A copy of the OWASP Agentic Skills Top 10 bookmarked.

If you are auditing for a team, also set up:

- A shared spreadsheet for tracking which skills passed and which were rejected, with the rejection reason.
- A standard rejection notice template you can send to the skill maintainer (responsible disclosure).

Check 1: Source and maintainer

The first 5 minutes per skill.

Do:

- Find the upstream Git repository. If there is no public source, fail immediately. Do not install closed-source skills from unknown maintainers.
- Check the maintainer's profile. Do they have commits before 2024? Do they ship to multiple projects? Does the GitHub account look real (followers, issues, contributor history)?
- Check the repo's commit history. Is the skill being maintained? Last commit in 2026 is green. Last commit in 2022 is yellow. Last commit "Initial commit" only is red.
- Check the issue tracker. Are issues being responded to? Are there open security-flavored issues that have been ignored for months?

Red flags that fail the check:

- No public source
- Maintainer account created in the last 90 days with no prior history
- Repo is a fork with no clear documentation of what was changed from upstream
- Last commit message is "fix" or "update" with no context, and there are no other commits

Documentation: Record the upstream URL, the maintainer handle, the commit hash you reviewed, and the date.

Check 2: Description and metadata

The next 5 minutes.

Do:

- Open the `SKILL.md` (or equivalent) in a hex viewer or run `cat -v skill.md` to surface invisible characters.
- Read the description as untrusted input. The skill's description is something Claude Code reads when deciding whether to invoke the skill. Prompt injections embedded here run *before* you confirm.
- Check for unicode tag characters. Range `U+E0020` through `U+E007F`. These are invisible in most editors but are valid text that an LLM will read as instructions.
- Check for base64 blobs, hex strings, or URL-encoded text that has no business being in a skill description.
- Confirm the description matches what the skill actually does. A skill named "format JSON" should describe formatting JSON, not "advanced productivity workflows."

Red flags that fail the check:

- Any invisible unicode in the description or metadata
- Base64 or hex blobs without a clear purpose

- Instructions to "ignore previous instructions," "as an exception, also do X," or "first, before anything else..."
- URLs to domains that are unrelated to the skill's stated purpose

Documentation: Paste the raw description into your audit notes. Note any anomalies.

Check 3: Tool surface

5-10 minutes.

Claude Code, Cursor, and Codex skills declare which tools they need. The principle is least-privilege: a skill should request only the tools it actually uses.

Do:

- List every tool the skill requests. `Bash`, `Read`, `Edit`, `WebFetch`, `Glob`, `Grep`, etc.
- For each tool, ask: does the skill's stated purpose need this?
- Watch especially for `Bash(*)` (unrestricted bash), `WebFetch` to arbitrary URLs, and `Edit` with no path restrictions.

Red flags that fail the check:

- A "format JSON" skill that requests `Bash(*)` and `WebFetch`
- Any skill that requests both `Read` of arbitrary paths AND `WebFetch` to arbitrary URLs (exfiltration vector)
- A skill that requests `Bash(*)` when narrower patterns like `Bash(git:*)` would suffice
- Tool requests not explained anywhere in the documentation

Documentation: List every tool, every flag, every restriction. Note any over-broad requests.

Check 4: Dependencies

5-15 minutes depending on how many.

Most skills are not single files. They pull in npm packages, Python packages, or shell tools.

Do:

- List every dependency in `package.json`, `requirements.txt`, `Cargo.toml`, or whatever the skill uses.
- For each dependency, check:
- Is it pinned to a specific version, or to `*` / `latest` /a range?
- Has it been published for at least 12 months?
- Does the maintainer have a real history?

- What is the download count? (Low download counts are not automatic fails but warrant extra scrutiny.)
- Is the name one character off from a popular package? (Typo-squatting check.)
- Look at the dependency's own dependencies. Supply-chain attacks often hide one level deep.

Red flags that fail the check:

- Any unpinned dependency
- A dependency published in the last 90 days with low downloads
- A dependency name that is a typo of a popular package
- A dependency that pulls in a known-vulnerable transitive package

Documentation: Record each dependency's name, pinned version, publication date, and download count. Note any that warrant follow-up.

Check 5: Example invocations

5-10 minutes.

Skills typically ship with example invocations. These are the most common attack vector after metadata injection.

Do:

- Read every example command in the skill's docs.
- For each, walk through what would actually run on your machine.
- Pay special attention to bash pipelines. The Anthropic source leak confirmed that 50+ subcommand pipelines can bypass deny rules; long pipelines are a red flag.
- Look for `curl | bash`, `wget | sh`, `eval`, `source <(...)`, base64 decoding into execution, etc.

Red flags that fail the check:

- Any pipeline longer than 3 stages without clear necessity
- Any `curl` / `wget` / `nc` to a domain you do not recognize
- Inline base64 or hex strings that are subsequently decoded and executed
- Use of `eval`, `bash -c "$(...)"`, `source <(...)`, or similar dynamic execution patterns
- Commands you would not type into a terminal yourself

Documentation: Quote any example that triggered a flag. Note your reasoning.

Check 6: License

2-5 minutes.

This check catches a category of problem that most "audited" bundles ignore: legal redistribution.

Do:

- Confirm a `LICENSE` file exists.
- Read it. Is it MIT, Apache 2.0, BSD, ISC, or another standard permissive license?
- Critically: are any bundled assets *source-available* rather than open source? Anthropic's official document skills (pdf, docx, xlsx, pptx) are source-available, not redistributable. If a paid bundle ships them, the bundle is infringing.

Red flags that fail the check:

- No LICENSE file
- A custom license you have not read
- Source-available components being redistributed
- License language that conflicts between the README and the LICENSE file

Documentation: Record the license, the source of the license text, and any per-file license differences.

Check 7: Prompt-injection scan

5-10 minutes.

The final integrity pass. Run automated checks for prompt-injection patterns.

Do:

- Pipe the skill's metadata, README, and examples through a prompt-injection scanner. Open-source tools include `promptinject`, `garak`, and `llm-guard`.
- Manually scan for the OWASP Agentic Skills Top 10 patterns (next section).
- Look for "ignore previous instructions" variants, indirect injection patterns ("if the user asks about X, also Y"), and confused-deputy patterns where the skill uses the LLM's trust to perform unauthorized actions.

Red flags that fail the check:

- Any flagged injection pattern from the automated scan
- Manual finding of any OWASP top-10 pattern

- Even one instance of "as an exception" or "ignore the system prompt" in metadata or examples

Documentation: Note the scanner used, the version, and any findings.

The pass/fail decision

A skill must pass all 7 checks to ship in our bundle. One failure = reject.

If you are auditing for personal use, the threshold can be lower (some teams accept yellow flags on dependencies if everything else is green). For team use, especially team use that touches customer data, 7-of-7 is the bar.

When you reject a skill, send a polite note to the maintainer explaining what would need to change for it to pass. Many maintainers will fix issues if they know about them. This is responsible disclosure in practice.

Mapping to OWASP Agentic Skills Top 10

The OWASP Agentic Skills Top 10 is the emerging standard framework. Our 7 checks map cleanly:

OWASP item	Our check
AS01: Prompt Injection	Check 2, Check 7
AS02: Excessive Agency	Check 3 (tool surface)
AS03: Insecure Output Handling	Check 5 (examples)
AS04: Supply Chain	Check 4 (dependencies)
AS05: Sensitive Information Disclosure	Check 3, Check 5
AS06: Unbounded Resource Consumption	Check 3 (tool flags)
AS07: System Prompt Leakage	Check 2
AS08: Improper Tool Use	Check 5
AS09: Misinformation	Check 1 (maintainer)
AS10: Inadequate Documentation	Check 1, Check 6

If you cite a finding to a maintainer, cite the OWASP number alongside our check number. Maintainers respond better to a standards-backed finding than to a one-off complaint.

Time budget per skill

Realistic time budget if you are doing this honestly:

- Check 1 (source): 5 minutes
- Check 2 (metadata): 5 minutes
- Check 3 (tool surface): 5-10 minutes
- Check 4 (dependencies): 5-15 minutes
- Check 5 (examples): 5-10 minutes
- Check 6 (license): 2-5 minutes
- Check 7 (injection scan): 5-10 minutes

Total per skill: 30-60 minutes.

For 40 skills, that is 20-40 hours of audit work. For a single solo developer evaluating their own install set, that is a weekend project. For a team auditing a bundle for production use, that is a real engineering investment.

The math for buying SkillVault is exactly this: $99 buys back 20-40 hours of skilled engineering time. If your time is worth more than $2.50/hour, the bundle is the right buy.

Get the audit methodology

We ship the full methodology PDF (this article plus the per-check worksheets, the scanner setup scripts, and the OWASP mapping in spreadsheet form) free to anyone who wants to do this work themselves. The form is on the SkillVault page. No credit card. Email only.

If you want the methodology *and* the 41 skills we audited with it, the lifetime bundle is $99.

Free methodology PDF → Get it on the SkillVault page

Skip the audit work → SkillVault $99 lifetime

Either path is honest. Pick the one that fits your time.

The 41 skills we audited with this framework are bundled at https://venture-hub-snowy.vercel.app/skillvault for $99 lifetime. Bug bounty: $200 cash for any verified vulnerability in a shipped skill, public disclosure within 48 hours.