skarn

RESEARCH / MEASUREMENT

How precise is attack-chain detection? We measured it.

0.0%

false-positive rate over a labeled corpus of 59 benign, chain-shaped agent sessions.

Zero false attack-chains, with 19 real-incident attacks all detected and every multi-stage chain correlated to a single critical finding.

Executive summary

Skarn's attack-chain detector raises 0 false attack-chains and 0 false findings of any kind over a labeled corpus of 59 benign, chain-shaped AI coding-agent sessions: a measured 0.0% false-positive rate. On the same corpus, 19 adversarial sessions modeled on real, published agent-exfiltration incidents are all detected, and every multi-stage attack is correlated into a single, correctly prioritized critical attack-chain rather than a pile of disconnected alerts.

Most benign sessions are grounded in an official vendor document that records the pattern as normal practice, every attack is grounded in a real published incident or advisory, and every credential in the corpus is synthetic and non-functional. The false-positive rate is the headline because the behavioral, multi-step view - a credential read that feeds an exfiltration, an injected page that drives a file read - is exactly where a detector can be loud and wrong, and a detector you do not trust is a detector you turn off.

The figures at a glance: 0 false attack-chains over 59 benign chain-shaped sessions (0.0%); 19 real-incident attacks all detected, recall held against regression; 5 multi-stage chains each correlated and tiered critical; 12 documented benign false-positive families; 9 attack families modeled on real incidents; 42 cited public sources behind 66 of the 78 behavioral fixtures; 189 synthetic secrets exercised, none ever shown unredacted in any output.

This report gives the full method, the conservatism that produces the number, the complete corpus breakdown, and a short checklist for measuring your own exposure with the same tool.

What we measured, and why the false-positive number is the one that matters

Most secret scanners tell you when a key looks like a key. Skarn also tells you when a session looks like an attack - a credential read that feeds an exfiltration, an injected web page that drives a file read, an encode step staged before a network call. That behavioral, multi-step view is where Skarn is different, and it is exactly where a detector can be loud and wrong. So we did what a behavioral detector should always be made to do: we built a labeled corpus of the benign developer sessions that look like attacks, and we measured the false-positive rate.

A precision claim is only worth as much as the corpus behind it. The credible bar in secret detection was set by published benchmarks with a named dataset, a stated method, and exact figures. Behavioral attack-chain detection on agent sessions has no public benchmark, because almost nobody does it. So we defined one, and we publish the method in full so the number is reproducible and falsifiable, not a marketing figure.

The false-positive rate is the number that matters for a tool you run against your own sessions every day. A detector that flags your installer one-liner, your OAuth flow, and your `curl` with a bearer token is a detector you turn off. Skarn's value is that it stays silent on all of those and fires on the real thing.

Where this sits against the other local scanner

This is also the axis on which the tooling diverges, and it is worth being precise about which axis it is. GitGuardian's Developer Endpoint Protection, announced 2026-06-16, scans the same surface Skarn does - AI tool directories and log files - locally on the endpoint, and states that only structured metadata leaves it. So the difference is not local versus cloud. It is the unit of analysis.

That product's documented scope is a credential inventory, honeytokens, and forwarding to a SIEM; as of that announcement it documents no attack-chain, behavioral, or MITRE ATLAS correlation over agent sessions. An inventory answers "is there a key here". The number in this report answers a different question - did this session behave like an attack, and how often is that answer wrong - and that is the question a correlating detector has to earn the right to ask. Earning it is what the rest of this report is about.

How the corpus is built: real scenarios, synthetic secrets

Two rules govern every fixture.

Real scenarios. Every benign pattern is grounded in an official vendor doc that documents it as normal practice - GitHub REST authentication, the uv, rustup, Ollama, Homebrew, and Deno install pages, OAuth 2.0 client-credentials guides, AWS CLI configuration, Docker Compose, and so on. Every attack pattern is grounded in a real, published incident or advisory: among them EchoLeak (CVE-2025-32711, the zero-click Microsoft 365 Copilot exfiltration), Claude Code DNS exfiltration (CVE-2025-55284), the Devin and Windsurf prompt-injection-to-secret-leak writeups, Invariant Labs' MCP tool-poisoning research, Snyk's malicious-agent-skills study, and MITRE ATT&CK's cloud-instance-metadata technique. We fetched and verified each source before using it; 42 distinct cited sources back 66 of the 78 behavioral fixtures.

The other 12 have a different provenance, and we would rather name it than blur it: they are regression guards for false positives this benchmark caught in Skarn itself - a duration named `TOKEN_TTL` read as a credential, a lone zero-width space read as an injection, a README whose prose mentions a dotfile read as a config-file write. Each was a real defect, each was fixed, and each is now a fixture that fails the build if it ever comes back. Their grounding is the defect, not an outside citation.

Synthetic secrets. Not one real credential appears anywhere in the corpus. Every token is synthetic, format-valid, and non-live - it matches a detector's shape, and its checksum where one applies, but unlocks nothing. This is enforced by an automated invariant: a gate extracts every secret-shaped token from the corpus - 189 of them - and proves none ever appears, unredacted, in any output Skarn produces. The chain fixtures push those tokens through the riskiest surface of all, the taint edges that hold raw values in memory while correlating a chain, and the gate still passes.

Extended methodology

What the number measures, precisely

A chain-shaped session is one that spans, or appears to span, more than one stage of an attack: a recon-shaped step (reading config or credentials, dumping the environment) plus an exfil-shaped step (an outbound network command, a pull-request body, a DNS lookup), or a poison step (fetched content carrying an instruction) feeding either. Skarn correlates the security events a session produces into attack-chains by propagating taint across the steps.

A correlated chain surfaces as a distinct attack-chain finding. A benign session counts as a false positive if it produces any finding at all - and a false chain cannot form without first producing a false finding. So "0 false positives over 59 benign chain-shaped sessions" is exact: no false attack-chain and no false single-stage finding occurs on any of the 59. This is a precision measurement, not a base-rate prevalence claim: when a benign developer does chain-shaped things, Skarn stays silent; when an attacker runs a real exfiltration chain, Skarn fires and correlates it.

Corpus composition

The behavioral track holds 78 labeled fixtures: 59 benign chain-shaped decoys across 12 real-world false-positive families, and 19 adversarial positives across 9 attack families, 5 of which form a correlated multi-phase attack-chain. It sits inside a larger labeled benchmark of 261 fixtures that also covers the secret, supply-chain, lockfile, and insecure-code tracks. 66 of the behavioral fixtures carry, in their label metadata, the real public URL the scenario is modeled on; the remaining 12 are regression guards for defects this benchmark found in Skarn.

Grounding and verification

Fixtures are grounded adversarially, not from memory. A structured research pass mined public security writeups, vendor postmortems, CVE and advisory pages, MITRE ATLAS and ATT&CK, and the OWASP LLM Top 10 2025 for each attack family and each benign false-positive family. A verification phase then fetched each cited URL and confirmed the page actually substantiates the claim, defaulting to "unverified" on any doubt; unverified examples were dropped rather than used. Benign-pattern citations not covered by the research - official vendor install, authentication, and configuration docs - were fetched and confirmed by hand.

What makes the detector precise

The 0.0% rate has held through every expansion of the corpus, without a single change to the correlation engine: the documented conservatism already holds the false-positive rate at zero. That conservatism is the precision the number certifies.

Taint edges form on full-value, case-insensitive matches only, never substrings, so two benign credential-shaped strings cannot spuriously link. Taint values are length-gated; short or common strings never create an edge. Edges are timestamp-ordered, so a later event cannot taint an earlier one. The exfiltration and DNS detectors are credential-gated: a network primitive alone is benign, and only reading local secret state into the outbound step is exfiltration - a bare `ping $(hostname)` or an authenticated `curl -H "Authorization: Bearer $(printenv TOKEN)"` stays silent. And unrelated taint clusters split into separate chains rather than inflating one merged blob.

Honest limitations

Stated plainly so the number survives scrutiny rather than inviting it.

It is precision, not prevalence. The number is a false-positive measurement over a curated corpus, not a field study. It says Skarn does not false-fire on these 59 real benign sessions, not that the false-positive rate in the wild is exactly zero. A corpus sampled from real traffic would strengthen it further; this one is built adversarially from the documented false-positive families, which is the right first bar.

Recall is held, not headlined. Recall is held against regression on 19 adversarial sessions, not measured as a percentage of the universe of agent attacks. That is intentional: a published "we miss X" figure is an attacker's map, so the false-positive rate is the number we lead with.

One hard case is deliberately out of scope. Documentation that merely discusses injection in prose is handled, and a decoy in the corpus holds it silent. But a security writeup that quotes a live injection string verbatim as an example would still trip prompt-poisoning: the detector keys on the instruction text, and it does not yet tell a quoted example apart from a real one. We document that boundary rather than engineer around it with a change that could regress recall.

The benign families we test against

These are the sessions that fool a naive behavioral detector. Skarn stays silent on all 59, across 12 families.

DNS and network diagnostics (9) - a bare `nslookup`, `dig`, or `ping`, an internal-IP connectivity check, listing `~/.ssh` or reading the public half of a key pair, or a runbook that discusses exfiltration in prose.

Multi-tool recon-then-action (9) - reading a config or non-credential file then building, installing, or calling a safe host, with nothing tainted flowing between the steps.

Installer one-liners (6) - `curl ... | sh` straight from the official install page (uv, rustup, nvm, Ollama, Homebrew, Deno).

Authenticated API requests (5) - a token in a `curl -H "Authorization: Bearer ..."` is how you are supposed to call an API (GitHub, OpenAI, Stripe, Atlassian).

The write-script-then-run loop (5) - the most normal thing an AI coding agent does.

OAuth 2.0 token grants (4) - client-credentials, refresh, and device-code flows that carry a client secret in the request body.

PR and commit bodies (4) - a changelog piped into a pull request, or a message that names an environment variable.

A dump-shaped command plus a separate outbound request (4) - the hardest false-merge case, where two innocent steps in one session share no taint.

Payload-shaped web pages (4) - a normal page with an inline base64 data-URI image or font and an analytics script, or documentation whose prose quotes a jailbreak phrase, but no injected instruction.

Named, not acted on (4) - source code whose variable names contain `token` or `secret`, a firewall rule that names the cloud-metadata IP without fetching it, a README whose prose mentions `.bashrc` without writing to it. The detector keys on the act, not the noun.

Env-dump diagnostics (3) - `printenv` or `env | grep` while debugging, with no secret in the output, and config values such as `TOKEN_TTL` that are durations rather than credentials.

Incidental invisible characters (2) - a lone zero-width space in a read file or a fetched page title. A bidi override staged in source is a Trojan-Source attack and still fires; one stray invisible is not.

The corpus is built to be hard, not easy. It includes the boundary cases - reading `~/.aws/config` (benign) versus `~/.aws/credentials` (a finding), `.env.example` versus `.env`, a public `.pub` key versus its private half, an environment dump sitting in the same session as a separate outbound request, an OAuth grant whose body carries `client_secret=$VAR` - precisely the places a sloppy correlator would merge two innocent steps into a false alarm. Each one trips a different benign-exemption gate and stays silent.

The attacks we still catch

The same corpus proves the detector did not go quiet to win on precision. Across 9 attack families it catches credential-read-then-exfiltrate chains and raw environment dumps (modeled on Snyk's ToxicSkills and MCP research and the Devin env-dump writeup), indirect prompt injections that drive a file read and a network call (Windsurf, EchoLeak), DNS-tunneled key exfiltration (Claude Code, CVE-2025-55284), exfiltration to known drop sites, pushes to attacker-controlled git remotes and pull-request-body credential theft (Microsoft's agentic CI/CD case, Invariant Labs' GitHub MCP), cloud-metadata credential theft (MITRE ATT&CK T1552.005), Trojan-Source bidi overrides staged in a source file, and persistence written into a shell startup file.

Each is modeled on a real reported incident or attack primitive, all 19 are detected, and the 5 multi-stage attacks - two-phase recon-to-exfiltrate, three-phase poison-to-recon-to-exfiltrate, and three-phase recon-to-obfuscate-to-exfiltrate - each correlate into a single attack-chain at the right severity, all tiered critical.

Reproducibility

The number is not a slide. It is the output of one command - so here is the command, the corpus it runs over, and the scorecard it prints.

The benchmark and the labeled corpus ship with the tool. One command scores the whole corpus and gates the build: it fails the moment one benign session trips a detector, one adversarial session stops firing, a chain drops to the wrong severity, or a number published in this report stops matching the corpus. A second command proves the corpus holds no live credential. `skarn taxonomies` prints the same figures from the binary itself.

```
zig build -Dtest-mode benchmark
zig build -Dtest-mode redaction-check
```

What the first command scores, on the behavioral track: 78 labeled fixtures.

- 59 decoys. Benign, chain-shaped sessions across the 12 false-positive families listed above. Each must produce no finding of any kind; one finding is a false positive and fails the build. A false chain cannot form without first producing a false finding, which is why this number gates the chain claim.
- 19 positives. Adversarial sessions across 9 attack families, each modeled on a published incident or attack primitive. Each must fire its expected rule, and where the fixture asserts a severity it must match. 5 of them are multi-stage and must correlate into a single attack-chain finding, tiered critical.
- 0 known gaps and 0 known bugs on this track - the other two label kinds the corpus uses. Both registers are gate-neutral by design and both are currently empty here.

Every credential in the corpus is synthetic and non-live, and that is not an assurance but a second gate: `zig build -Dtest-mode redaction-check` extracts all 189 secret-shaped tokens from the corpus and asserts that none of them appears unredacted in any output Skarn produces - every format, the baseline file, the audit log, the export, the local scan API, and the guard verdicts. The chain fixtures are the hardest case for that invariant, because correlating a chain means holding raw credential values in memory across steps.

The scorecard the gate prints:

```
FALSE-POSITIVE RATE (benign fixtures that fired any rule)
  all decoys:        0/108  (0.0%)
  behavioral decoys: 0/59  (0.0%)   <-- attack-pattern / chain precision (L1)
  secret decoys:     0/40  (0.0%)   <-- leaked-credential precision
  secret recall:     111/111  (100.0%)   current-format coverage (held against
regression)

GATE: PASS
```

The raw corpus is not published as a download. The fixture bodies, the generator, and the scoring rule map together spell out the exact shape each detector keys on, the precise boundary of every benign exemption, and the full register of what does not fire - which is the same "we miss X" map this report declines to publish for the reason given above. The manifest, the command, and the expected output are the falsifiable part: they state exactly what is measured and what result would contradict it.

How to assess your own exposure

The corpus shows the shape of the problem. Measuring yours takes a few minutes and never leaves your machine.

1. Install Skarn. It is a single static binary for macOS, Windows, and Linux, on both Intel and ARM.
2. Run `skarn check`. It scans your AI coding-assistant sessions locally - Claude Code, Gemini CLI, Codex CLI, Cursor, and GitHub Copilot - with no network calls by default. Nothing is uploaded.

3. Read the report. A correlated attack-chain is surfaced as one prioritized finding with its stages, not a scatter of disconnected alerts, and any secrets in it are masked to a fixed form that does not disclose their length.
4. Act on the findings, and trust the silence. When Skarn raises a chain, the precision measured here is what makes it worth investigating rather than dismissing. When Skarn stays quiet on your installer one-liners, OAuth flows, and authenticated API calls, that silence is measured, not assumed. Skarn surfaces the chain; responding to it - rotating a credential, revoking access - is yours to do.

Sources

The attack families are grounded in these public incidents and advisories, each re-verified on 2026-07-12; the benign families cite official vendor install, authentication, and configuration docs. The corpus counts in this report were re-measured against the live gate the same day.

- EchoLeak, the zero-click Microsoft 365 Copilot data-exfiltration vulnerability (CVE-2025-32711): https://simonwillison.net/2025/Jun/11/echoleak/
- Claude Code data exfiltration via DNS requests (CVE-2025-55284): https://embracethered.com/blog/posts/2025/claude-code-exfiltration-via-dns-requests/
- Devin can leak your secrets (env-dump exfiltration): https://embracethered.com/blog/posts/2025/devin-can-leak-your-secrets/
- Windsurf data-exfiltration vulnerabilities (prompt injection to leak): https://embracethered.com/blog/posts/2025/windsurf-data-exfiltration-vulnerabilities/
- OpenHands and the lethal trifecta (read, stage, exfiltrate): https://embracethered.com/blog/posts/2025/openhands-the-lethal-trifecta-strikes-again/
- Snyk ToxicSkills, malicious AI agent skills on ClawHub: https://snyk.io/blog/toxicskills-malicious-ai-agent-skills-clawhub/
- Snyk Labs, prompt injection over MCP (file read then network POST): https://labs.snyk.io/resources/prompt-injection-mcp/
- Invariant Labs, GitHub MCP vulnerability (PR-body credential theft): https://invariantlabs.ai/blog/mcp-github-vulnerability
- Microsoft, securing CI/CD in an agentic world (push to attacker remote): https://www.microsoft.com/en-us/security/blog/2026/06/05/securing-ci-cd-in-agentic-world-claude-code-github-action-case/
- MITRE ATT&CK T1552.005, unsecured credentials in the cloud instance metadata API: https://attack.mitre.org/techniques/T1552/005/
- GitGuardian, extending our mission with Developer Endpoint Protection (2026-06-16), the source for the endpoint-scanner scope described above: https://blog.gitguardian.com/extending-our-mission-with-developer-endpoint-protection/

Benign-family grounding (official vendor documentation, not linked individually): GitHub, OpenAI, Stripe, and Atlassian authentication docs; the uv, rustup, nvm, Ollama, Homebrew, and Deno install pages; the oauth.com, Auth0, and Microsoft Entra OAuth guides; AWS CLI configuration and environment-variable

docs; the dotenv, Docker Compose, pip, and GNU Make docs; the MDN data-URI documentation; and the pytest, npm, pandas, and Node usage docs.