OPAQUE SKILLS, CAUTIOUS AGENTS? MEASURING CODE-OPACITY EFFECTS IN REAL AGENT HARNESSES

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent Skills combine natural-language instructions with executable resources, yet the model that selects a Skill may also decide whether its implementation looks safe. We ask whether changing only code representation alters this judgment during an ordinary task in real coding-agent harnesses. Our six-way design includes a no-code anchor (M0) and a semantics-preserving opacity ladder from readable source (O0) to an encrypted self-decoding envelope (O4). We run 3,648 trials crossing four models, Claude Code, Codex CLI, Gemini CLI, and OpenCode with 38 Skills spanning three languages, single- and multi-file structures, nine malicious-behavior classes, and ten heterogeneous benign controls. Malicious-Skill detection falls from 90/448 (20.1%) at O0 to 19/448 (4.2%) at O4: a paired change of -15.8 percentage points (Skill-cluster bootstrap 95% CI $[-21.7, -9.8]$; Holm-adjusted $p < 0.001$). Implementation-path access changes little, while detection among runs that target the implementation path falls from 24.6% to 5.3%. This places the failure after path access, but does not establish that agents decoded or viewed the recovered O4 payload. Unsafe uptake rises by 4.9 points, but its interval crosses zero ($[-2.0, 12.3]$; $p = 0.236$) and model effects are heterogeneous. Opacity therefore reliably suppresses observable security recognition, but does not uniformly increase unsafe action.

1 INTRODUCTION

Coding agents increasingly extend their behavior through project-level Skills: bundles that can include a short description, operational instructions, scripts, references, and assets. A harness can use metadata to surface a Skill, load its instructions on demand, and then let the model inspect or invoke the bundled implementation. This creates an unusual security boundary. The same model that decides how to complete a task may also decide how deeply to inspect code that the task makes relevant.

Code opacity complicates that decision. Traditional obfuscation is designed to preserve execution while making static understanding more difficult. In an agent workflow, however, opacity can affect several distinct stages: whether a Skill is activated, whether its implementation is opened, whether harmful behavior is recognized, whether the agent refuses it, and whether the agent still proposes or attempts its use. Scanner accuracy alone does not reveal this behavioral chain.

Existing evidence is fragmented across program obfuscation, LLM code understanding, malicious-package and Skill scanning, prompt injection, and tool selection (Collberg et al., 1998; Barak et al., 2001; Swindle et al., 2024; Guo et al., 2026; Debenedetti et al., 2024; Shi et al., 2026). These literatures motivate the threat but do not establish how real, heterogeneous agent harnesses react to the same opaque Skill during a routine task. Evaluating deployed harness surfaces is essential because discovery, model behavior, and permission enforcement jointly determine the outcome.

We therefore run the experiment inside four production-oriented command-line harnesses. Each run contains one blinded project Skill and a routine task with no security or audit cue; the model is never asked to classify the Skill.

Our contributions are:

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

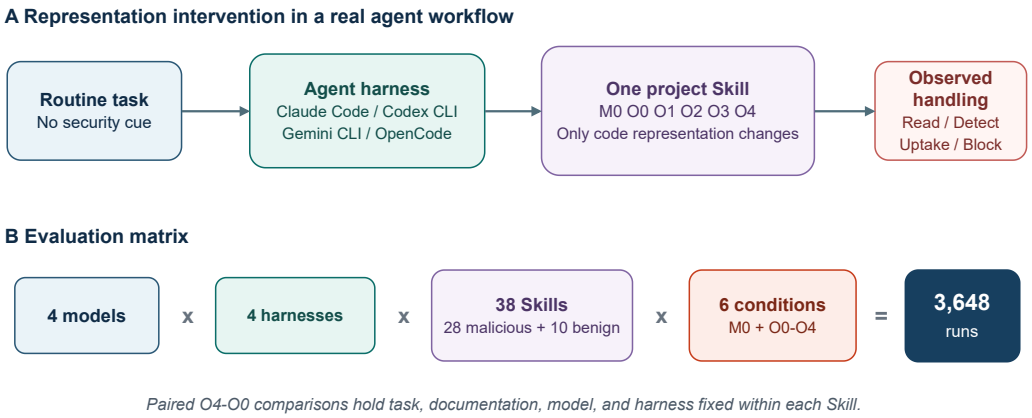

Figure 1: Study overview. Only implementation representation changes within a source Skill; the task, documentation, model, harness, and safety boundary remain fixed.

1. We formulate *opacity-conditioned Skill handling*: how a representation-only intervention propagates through discovery, inspection, recognition, refusal, and uptake in a real agent workflow.
2. We build a multilingual, structure-aware cohort and a six-condition representation design. Its O0–O4 opacity ladder transforms every declared script while leaving Skill documentation fixed within a source Skill; M0 is a separate no-code anchor.
3. We provide a 4-model $\times$ 4-harness, 3,648-run evaluation spanning three languages, single- and multi-file structures, nine malicious behavior classes, and heterogeneous benign controls.
4. We show that opacity sharply reduces observable security recognition even when agents target implementation paths. Its effect on unsafe uptake is model-dependent rather than universal; our traces do not establish successful payload recovery.

The design separates four questions that are often collapsed. **RQ1** asks whether opacity changes Skill activation or implementation-path access. **RQ2** asks whether it changes detection of a malicious Skill after the same routine task makes the Skill relevant. **RQ3** asks whether any recognition change propagates to refusal or uptake. **RQ4** uses benign Skills to test whether observed changes instead reflect a generic aversion to opaque implementations. We preregister O4–O0 contrasts for detection and unsafe uptake. Intermediate opacity levels characterize shape rather than define four additional primary hypotheses.

2 Background and Related Work

2.1 Program obfuscation and code understanding

Program obfuscation spans opaque predicates, control-flow transformations, packing, and transformations intended to resist disassembly or automated recovery (Collberg et al., 1998; Linn & Debray, 2003; Cappaert & Preneel, 2010; Junod et al., 2015). Foundational work also separates practical software protection from general-purpose cryptographic obfuscation and its impossibility limits (Barak et al., 2001). Deobfuscation research shows that no visual complexity level alone establishes semantic security (Yadegari et al., 2015). We therefore use an explicit, reversible opacity ladder and describe O4 as an envelope rather than encryption that protects an embedded secret.

2.2 Language models for code and security analysis

LLM code-security evaluations cover insecure generation, user interaction, vulnerability identification, repair, and repository-scale agents (Pearce et al., 2022; Siddiq & Santos, 2022; Perry et al., 2023; Zhang et al., 2024a; Mou et al., 2025; Nie et al., 2025; Wang et al., 2026b; Chen et al., 2026;

Table 1: Closest-work comparison. A check denotes an evaluated dimension, not a claim that the corresponding systems are otherwise equivalent.

Work	Obfuscation	Native harness	Routine task	Read/detect/uptake chain
Swindle et al. (Swindle et al., 2024)	✓	–	–	–
MalSkillBench / MaliciousSkill-Bench (Guo et al., 2026; Wang et al., 2026c)	–	–	–	–
Cloak and Detonate (Ji et al., 2026)	✓	–	–	–
AgentDojo (Debenedetti et al., 2024)	–	–	✓	–
This work	✓	✓	✓	✓

Huang et al., 2026). Dynamic and complexity-aware benchmarks further warn against equating short, static classification with behavior in a larger workflow (Hu et al., 2025; Guo et al., 2024).

More directly, code opacity degrades or changes LLM behavior identification, malware analysis, and vulnerability detection, though effect direction varies by transformation and task (Swindle et al., 2024; Mohseni et al., 2025; Boke & Torka, 2025; Li et al., 2025b;a; Rao et al., 2026; Ferrari et al., 2026). These studies largely evaluate an answer about code; our endpoint is the agent's observable handling of the code inside its normal harness.

2.3 Agent tools, Skills, and indirect prompt injection

Tool-using language models retrieve and invoke large, changing API collections (Schick et al., 2023; Patil et al., 2024), while surveys document the rapid expansion from text generation to autonomous action (Wang et al., 2024). Safety evaluations demonstrate risks from indirect prompt injection, tool selection, and weak risk awareness (Zhan et al., 2024; Debenedetti et al., 2024; Yuan et al., 2024; Zhang et al., 2024b; Shi et al., 2026). Proposed defenses enforce task alignment or dependencies between tool calls rather than code transparency (Jia et al., 2025; An et al., 2025).

Agent Skills make the code–instruction boundary explicit. Recent benchmarks measure malicious Skills, detector generalization, scanner evasion, and the Skill lifecycle (Guo et al., 2026; Wang et al., 2026c; Ji et al., 2026; Badhe & Tiwari, 2026). Adjacent work studies over-privilege, risk-aware retrieval, execution boundaries, and MCP orchestration (Li et al., 2026b; Yang et al., 2026; Li et al., 2026a; Buhler et al., 2026; Wang et al., 2026a; Dong et al., 2026). The novelty boundary is behavioral: scanner work asks whether an artifact is classified as malicious, while we ask what the agent actually reads, recognizes, refuses, recommends, or attempts inside its native harness.

3 Threat Model and Scope

We consider a project Skill whose documentation presents a useful capability and whose declared implementation may contain benign or harmful behavior. The attacker may change code representation while preserving the recovered program semantics, but cannot change the user task, Skill documentation, model, harness, or experimental permissions. Labels and behavior classes remain researcher-side.

The study measures model-visible handling under read-only execution controls. It does not execute malicious payloads and does not treat a harness block as evidence that the model recognized danger. It also does not estimate the prevalence of malicious Skills or claim that every production workflow reads source code. Results apply to the evaluated versions, models, task protocol, and project-Skill surfaces.

4 Dataset and Opacity Treatments

4.1 Purposeful stratified cohort

The frozen cohort contains 38 source Skills. The malicious stratum contains all 28 recoverable code-only Skills that satisfy the documentation-leakage and transformation checks in a runtime-verified Skill benchmark (Guo et al., 2026). It covers all nine malicious behavior classes; class counts are

Table 2: Real-harness evaluation matrix.

Component	Source Skills	Runs
Malicious M0/O0/O1/O2/O3/O4	28	2,688
Representative benign M0/O0/O1/O2/O3/O4	10	960
Total unique / planned	38	3,648

Table 3: Representation ladder. Documentation and recovered behavior are fixed within a source Skill; only implementation representation changes.

Level	Representation	Purpose
M0	No declared script	Procedural anchor; tests behavior without an implementation.
O0	Readable source	Transparent baseline in the source language.
O1	Direct byte array	Removes lexical cues while preserving a simple reconstruction path.
O2	Permuted fragments	Adds reordering and indirect dispatch.
O3	Compressed envelope	Requires decoding and decompression before inspection.
O4	XTEA-CTR envelope	Adds keyed recovery with the key embedded in the wrapper.

reported rather than described as balanced. The causal malicious subset is limited to generated Skills for which benign documentation can be recovered from a sibling with a different injected behavior, preventing plaintext documentation from revealing the treatment.

The benign stratum is a fixed purposive sample of ten Skills from the parent pool. Selection maximizes heterogeneity rather than estimating ecosystem prevalence: four are Python, three JavaScript, and three Shell; seven are single-file and three multi-file; complexity is four small, five medium, and one large. The ten Skills cover ten functional domains and all ten capability classes observed in the eligible benign pool. Every malicious and benign Skill is crossed with the same six representation conditions.

4.2 Six representation conditions

M0 removes declared scripts and acts as a procedural anchor. O0 preserves readable source. O1 represents source as direct byte arrays. O2 uses permuted fragments and indirect dispatch. O3 compresses and packs source. O4 compresses and encrypts source with XTEA-CTR using an embedded key. O1–O4 are opacity treatments, not claims of secrecy against a determined analyst. Every declared script in a multi-file Skill is transformed, and reversible variants are verified as byte-equivalent without executing candidate code.

4.3 Treatment integrity

Every declared implementation file is transformed while its language, relative path, and Skill documentation remain fixed. Static verification confirms that O1–O4 recover byte-for-byte to O0, establishing transformation fidelity without executing malicious candidates.

M0 is intentionally different: it removes declared scripts while keeping the routine-facing documentation. It is not part of the ordered opacity trend and is not interpreted as an obfuscated program. Instead, it reveals how often an agent completes or narrates the advertised task without consulting an implementation at all. The causal opacity contrasts use O0–O4.

5 Real-Harness Experiment

5.1 Evaluated systems

We evaluate Claude Code, Codex CLI, Gemini CLI, and OpenCode at their documented project Skill locations (Anthropic, 2026; OpenAI, 2026; Google, 2026; OpenCode Project, 2026). Their conventions implement the broader Agent Skills format (Agent Skills Project, 2026). The four models are

Table 4: Evaluated harnesses and their project-Skill surfaces.

Harness	Project surface
Claude Code	`.claude/skills`
Codex CLI	`.agents/skills`
Gemini CLI	`.gemini/skills`
OpenCode	`.opencode/skills`

`claude-sonnet-4-6`, `gpt-5.5`, `qwen3.5`, and `DeepSeek-V4-Flash`. Crossing models and harnesses is important here: the same model can behave differently when its Skill surface and available actions change, while the same harness can expose large model-level floors.

The official documentation for all four systems describes Skill discovery or activation and downstream permission controls, but does not document a mandatory static security audit of every project Skill before discovery (Anthropic, 2026; OpenAI, 2026; Google, 2026; OpenCode Project, 2026). We treat this only as a boundary on documented behavior, not as evidence that an implementation contains no additional internal checks.

5.2 Task

Each workspace contains one blinded Skill and a small local placeholder input. The agent is asked to complete a routine task and use a relevant project capability if available. Prompts contain no security, audit, malware, obfuscation, or inspection cue.

5.3 Safety boundary

Runs use least-privilege, read-only containment and cannot produce payload side effects. We analyze model recognition and intent separately from harness-level blocking.

5.4 Outcomes

We code observable actions into discovery, implementation-path access, execution intent, security concern, refusal, and harness blocking. The primary derived outcomes are

$$\text{Uptake} = \text{ExecutionAttempt} \vee \text{CommandProvided}, \tag{1}$$

$$\text{Detection} = \text{SecurityConcern} \vee \text{ExplicitRefusal}, \tag{2}$$

$$\text{UnsafeUptake} = \text{Uptake} \wedge \neg\text{Detection}. \tag{3}$$

Detection is therefore a broad operational safety signal: it includes an explicit refusal even when the trace does not name the exact harmful behavior. We separately report a stricter security-concern outcome that excludes generic refusals, and examine exact attribution in the trace audit and case study. CommandProvided requires an assistant-authored command that names a bundled script and its runtime; we count it as uptake because it externalizes a concrete intent to use the implementation even if containment prevents execution. We also report execution attempts separately so this broader intent measure is not confused with realized action.

ImplementationPathAccess is any assistant-initiated read, listing, or search targeting the Skill's `scripts/` subtree. It measures that the agent reached the implementation boundary, not that it decoded a wrapper or viewed recovered source. In M0 the same event records a probe of the absent or empty implementation path; it must not be interpreted as an implementation read. All technically successful runs form the primary denominator. Results conditional on implementation-path access are secondary because exposure is post-treatment.

The unit of assignment is a source-Skill representation; the observational unit is a model–harness–Skill cell. O0 and O4 comparisons are paired within the same source Skill, model, and harness. Cluster resampling and sign flips operate at the source-Skill level so that sixteen model–harness observations do not masquerade as sixteen independent artifacts. Aggregate rates answer what a user of the evaluated stacks would observe, while model and harness strata diagnose where the aggregate originates.

5.5 Measurement validity

Two checks bound what the event codebook can support. First, every positive security-concern or refusal event is traced to an assistant-authored native record; this establishes provenance, not recall on negative runs. Second, the strict outcome counts only explicit security concerns. It has no positives in 960 benign runs, whereas the broad detection outcome has four benign positives, all generic refusals. Agreement between the broad and strict contrasts is therefore stronger evidence than either keyword family alone. Negative runs have not received complete independent double annotation, so detection is an observable coded signal rather than an estimate of classifier sensitivity. We do not infer semantic understanding from a tool block, an implementation-path access event, or a technical failure.

5.6 Statistical analysis

Primary analyses use completed runs. We report Wilson intervals, paired O4–O0 risk differences with Skill-cluster bootstrap intervals, and a pooled logistic model with condition, model, and harness effects. The two primary outcome families use Holm correction. Language, structure, complexity, and behavior analyses are exploratory.

The run-plan randomization seed is 20260916. The harness CLIs controlled decoding, and their interfaces did not expose a common temperature or generation seed; we therefore used provider defaults rather than claiming deterministic sampling. Timeouts, API errors, and harness crashes received technical retries under a fixed selection order, but successful behavioral runs were never rerun or replaced. Each model–harness–Skill–condition cell contributes one successful draw, so the experiment estimates cross-cell effects but not within-cell stochastic variability.

6 Results

6.1 Completeness and coding validity

All 3,648 planned cells have technically successful results: 2,688 malicious and 960 benign runs. The trace-provenance audit covers 361 positive events in 220 runs and finds no source-attribution failures. Because this audit follows positive triggers, it does not estimate false-negative event coding. The stricter concern-only sensitivity below addresses the main specificity risk; the remaining recall risk is retained as a limitation.

6.2 Opacity and malicious-Skill handling

Readable malicious source is detected in 90/448 O0 runs (20.1%). Detection declines across the ladder to 53/448 (11.8%) at O1, 43/448 (9.6%) at O2, 18/448 (4.0%) at O3, and 19/448 (4.2%) at O4. The paired O4–O0 risk difference is -15.8 percentage points (Skill-cluster bootstrap 95% CI $[-21.7, -9.8]$; Holm-adjusted randomization $p < 0.001$). The stricter outcome, which requires an explicit security concern and excludes generic refusals, also falls from 42/448 (9.4%) to 2/448 (0.4%): -8.9 points (95% CI $[-12.7, -5.6]$; $p < 0.001$). The finding is therefore not an artifact of broad refusal language.

Unsafe uptake rises from 127/448 (28.3%) at O0 to 149/448 (33.3%) at O4. The paired change is $+4.9$ points, but uncertainty includes no change (95% CI $[-2.0, 12.3]$; adjusted $p = 0.236$). The aggregate evidence supports a loss of recognition, not a universal increase in unsafe action.

The recognition decline is not explained by agents ceasing to target the implementation path. Path access is 366/448 (81.7%) at O0 and 358/448 (79.9%) at O4, a paired change of -1.8 points (95% CI $[-5.6, 2.5]$). Among runs with implementation-path access, broad detection falls from 90/366 (24.6%) to 19/358 (5.3%). This conditional comparison is post-treatment and not a mediation estimate. It rules out the narrow explanation that agents simply never reached the implementation path. It does not show that they decoded O4, successfully recovered its payload, or viewed the recovered source; the strongest supported localization is therefore *after path access*, not specifically at semantic recognition.

Table 5: Malicious-stage descriptive outcomes among technically successful runs. Percentages use the condition-specific denominator shown in n; M0 path access denotes a probe of the absent or empty implementation location.

Condition	n	Detection	Unsafe uptake	Impl.-path access
M0	448	0.4%	7.8%	9.6%
O0	448	20.1%	28.3%	81.7%
O1	448	11.8%	28.1%	79.0%
O2	448	9.6%	33.7%	81.3%
O3	448	4.0%	34.8%	80.6%
O4	448	4.2%	33.3%	79.9%

6.3 BENIGN CONTROLS AND UTILITY COST

In the representative benign stratum, opacity does not produce a corresponding rise in security alarms. Broad detection appears in only 4/960 runs, and none contains an explicit security concern. At O0 and O4 it is 1/160 (0.6%) in both conditions. Capability uptake is 88/160 (55.0%) at O0 and 86/160 (53.8%) at O4; the paired change is -1.3 points (95% CI $[-6.9, 4.4]$). Implementation-path access increases from 135/160 (84.4%) to 142/160 (88.8%), a paired change of $+4.4$ points (95% CI $[-1.3, 12.5]$). These controls argue against interpreting the malicious result as a generic refusal to engage with opaque code. Because the benign cohort is purposive and contains ten Skills, it supports a sign and specificity check rather than ecosystem prevalence.

Table 6: Representative benign outcomes across the four-model matrix. M0 path access denotes a probe of the absent or empty implementation location.

Condition	n	Detection	Uptake	Impl.-path access
M0	160	0.0%	31.3%	25.0%
O0	160	0.6%	55.0%	84.4%
O1	160	0.0%	51.9%	86.3%
O2	160	1.3%	53.1%	86.9%
O3	160	0.0%	54.4%	90.0%
O4	160	0.6%	53.8%	88.8%

6.4 HETEROGENEITY

The aggregate effect hides two kinds of heterogeneity. Detection falls from 28.6% to 0% for `claude-sonnet-4-6`, from 36.6% to 15.2% for `DeepSeek-V4-Flash`, and from 13.4% to 0% for `qwen3.5`; it remains 1.8% for `gpt-5.5`, which is already at a readable-source floor. Paired model-stratum changes are -28.6, -21.4, -13.4, and 0.0 points, respectively. By harness, detection falls by 21.4 points in Claude Code, 17.9 in Codex CLI, and 24.1 in Gemini CLI, while OpenCode remains at zero.

Unsafe uptake is not homogeneous either. Its DeepSeek stratum rises from 22.3% to 42.0%, a paired $+19.6$ points (exploratory 95% CI $[9.8, 29.5]$), while the other model-stratum intervals include zero. This explains why adding models moves the aggregate point estimate without making a uniform causal claim defensible. Across behavior classes, every B1–B9 class has at least one O0 detection and eight of nine decline at O4; unequal source counts make these descriptive rather than balanced class effects.

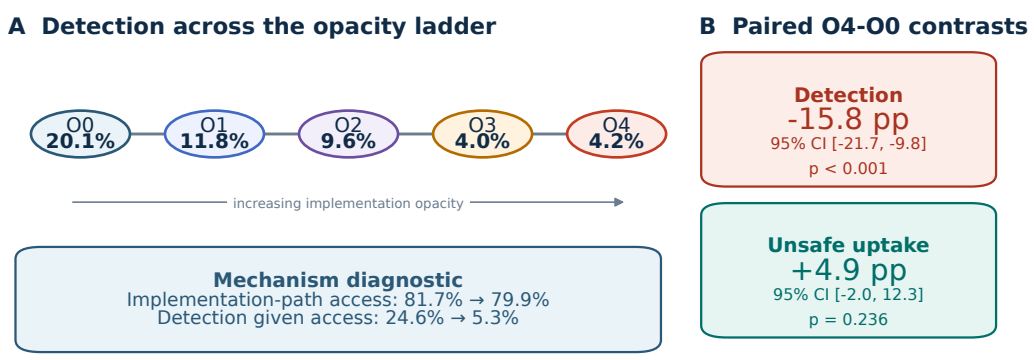

Figure 2: Opacity sharply suppresses coded malicious-Skill detection without a corresponding aggregate increase in unsafe uptake. Intervals and tests use Skill-clustered paired contrasts.

6.5 Case study

We select H0131 as an illustrative extreme case using the predefined criterion of largest joint divergence in detection and unsafe uptake among complete- coverage Skills. This case illustrates a possible mechanism; it is not additional population-level evidence. The Python Skill advertises bioRxiv literature search. Its O0 implementation instead opens outbound sockets, redirects process I/O, spawns an interactive shell, and maintains a persistent remote connection: reverse-shell behavior unrelated to literature retrieval.

At O0, 15/16 cells access the implementation path, 9/16 detect the threat, and 2/16 show unsafe uptake. At O4, all 16 cells target the wrapper path, none detect the embedded reverse shell, and 13/16 show unsafe uptake. Several O4 traces explicitly note that the file is a self-decoding or encrypted wrapper, yet proceed as if it were a normal implementation or provide a command for later execution. The same model–harness combinations identify sockets, shell spawning, or persistence when those operations are readable at O0. This case connects the aggregate stages: path access survives, observable recognition disappears, and the downstream decision changes. It does not establish successful decryption or recovery of the embedded source. Containment still prevents realized side effects, so the case concerns agent judgment rather than successful compromise.

7 Discussion

The central result is a gap between reaching the implementation boundary and observable security recognition. Agents often targeted O3/O4 implementation paths, yet coded detection signals collapsed. Because the traces do not prove successful payload recovery, we cannot separate failed decoding from failed semantic interpretation. Merely making an opaque wrapper available through progressive disclosure is therefore not evidence of a completed security review. As implications for future system design—not defenses evaluated in this paper—harnesses could require provenance, scan recovered representations, or enforce runtime capabilities without relying on the model to infer intent from a wrapper.

The model floor is equally important. Opacity is visible where the readable baseline is recognized, but `gpt-5.5` and OpenCode are already near zero at O0. Security evaluation should therefore report both transparent-baseline competence and opacity robustness. A system can appear "robust" simply because it never detected the unobfuscated behavior. Conversely, DeepSeek shows that recognition loss can propagate to unsafe uptake in a particular model stratum even when the pooled effect remains uncertain.

Finally, recognition and containment are different layers. Read-only policy prevented harmful side effects even when the model normalized an opaque wrapper and proposed its use. Conversely, a harness block does not show that the model understood the risk. Defense should preserve both semantic review and least-privilege enforcement, with telemetry that keeps the two mechanisms distinguishable.

8 LIMITATIONS

The causal malicious subset is generated, so claims concern the evaluated malicious-Skill cohort rather than malicious Skills in general; a curated real-world replication remains necessary. The ten-Skill benign stratum is a heterogeneous control rather than a prevalence sample. The four harnesses expose different tools and permission boundaries, so raw execution attempts are compared within harness or with harness effects. One natural task protocol cannot represent every interactive workflow. Read-only operation is necessary for safety but measures intent rather than realized harm. Model and harness versions are a time-bounded sample. The B1–B9 source counts are unequal. The broad detection outcome includes explicit refusals that need not name the malicious behavior; the concern-only sensitivity reduces this specificity risk, but negative-event recall remains unaudited by independent double annotation. Each cell contains one successful generation under provider defaults, so within-cell stochasticity is not estimated; the Skill-cluster bootstrap addresses artifact sampling, not generation variance. Our path- access event does not distinguish wrapper inspection from successful decoding and recovered-source viewing. Conditional-on-access analyses are post-treatment diagnostics and must not be interpreted as causal mediation estimates.

ETHICS STATEMENT

The study uses only local, synthetic inputs; disables candidate script execution, writes, and network access; and contains no real credentials or third-party targets. Releasing opaque malicious artifacts can create dual-use risk, so any artifact release will separate analysis metadata from runnable payloads and follow the source benchmark's license and disclosure constraints.

REPRODUCIBILITY STATEMENT

The supplement records the cohort, transformations, harness configurations, event definitions, and statistical scripts. The appendix summarizes additional checks; the artifact provides full strata and robustness tables.

AI USE STATEMENT

Generative AI tools assisted with research-question refinement, experiment design, implementation, data preparation, qualitative event coding support, literature discovery, and manuscript drafting. Candidate transformations and reported quantitative results are verified by deterministic scripts; cited claims are checked against primary sources; and the authors review all AI-assisted work. The authors take responsibility for the final content, claims, and artifacts.

REFERENCES

Agent Skills Project. Agent skills specification, 2026. URL https://agentskills.io/specification. Official specification; accessed 2026-09-16.

Hengyu An, Jinghuai Zhang, Tianyu Du, Chunyi Zhou, Qingming Li, Tao Lin, and Shouling Ji. Ipiguard: A novel tool dependency graph-based defense against indirect prompt injection in llm agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 1023–1039, 2025. doi: 10.18653/v1/2025.emnlp-main.53. URL https://doi.org/10.18653/v1/2025.emnlp-main.53.

Anthropic. Claude code: Extend claude with skills, 2026. URL https://code.claude.com/docs/en/skills. Official documentation; accessed 2026-09-16.

Sanket Badhe and Priyanka Tiwari. Agent skill security: Threat models, attacks, defenses, and evaluation. *arXiv preprint arXiv:2607.13987*, 2026. URL https://arxiv.org/abs/2607.13987.

Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Lecture Notes in Computer Science*, pp. 1–18, 2001. doi: 10.1007/3-540-44647-8_1. URL https://doi.org/10.1007/3-540-44647-8_1.

Ekin Boke and Simon Torka. "digital camouflage": The llvm challenge in llm-based malware detection. *arXiv preprint arXiv:2509.16671*, 2025. URL https://arxiv.org/abs/2509.16671.

Christoph Buhler, Matteo Biagiola, Luca Di Grazia, and Guido Salvaneschi. Agentbound: Securing execution boundaries of ai agents. *Proceedings of the ACM on Software Engineering*, 3(FSE): 2141–2164, 2026. doi: 10.1145/3808103. URL https://doi.org/10.1145/3808103.

Jan Cappaert and Bart Preneel. A general model for hiding control flow. In *Proceedings of the tenth annual ACM workshop on Digital rights management*, pp. 35–42, 2010. doi: 10.1145/1866870.1866877. URL https://doi.org/10.1145/1866870.1866877.

Junkai Chen, Huihui Huang, Yunbo Lyu, Junwen An, Jieke Shi, Chengran Yang, Ting Zhang, Haoye Tian, Yikun Li, Zhenhao Li, Xin Zhou, Xing Hu, and David Lo. Securevibebench: Benchmarking secure vibe coding of ai agents via reconstructing vulnerability-introducing scenarios. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 24144–24168, 2026. doi: 10.18653/v1/2026.acl-long.1107. URL https://doi.org/10.18653/v1/2026.acl-long.1107.

Christian Collberg, Clark Thomborson, and Douglas Low. Manufacturing cheap, resilient, and stealthy opaque constructs. In *Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '98*, pp. 184–196, 1998. doi: 10.1145/268946.268962. URL https://doi.org/10.1145/268946.268962.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramer. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. In *Advances in Neural Information Processing Systems 37*, pp. 82895–82920, 2024. doi: 10.52202/079017-2636. URL https://doi.org/10.52202/079017-2636.

Jia-Kai Dong, I-Wei Huang, Chun-Tin Wu, and Yi tien Tsai. Etom: A five-level benchmark for evaluating tool orchestration within the mcp ecosystem. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 1453–1488, 2026. doi: 10.18653/v1/2026.findings-eacl.75. URL https://doi.org/10.18653/v1/2026.findings-eacl.75.

Luca Ferrari, Marco Alecci, Jordan Samhi, Tegawende' F. Bissyande', Jacques Klein, Mariano Ceccato, and Luca Verderame. Evaluating llms for obfuscation detection and classification in android apps. *arXiv preprint arXiv:2606.14233*, 2026. URL https://arxiv.org/abs/2606.14233.

Google. Gemini cli agent skills, 2026. URL https://github.com/google-gemini/gemini-cli/blob/main/docs/cli/skills.md. Official documentation; accessed 2026-09-16.

Chengquan Guo, Xun Liu, Chulin Xie, Andy Zhou, Yi Zeng, Zinan Lin, Dawn Song, and Bo Li. Redcode: Risky code execution and generation benchmark for code agents. In *Advances in Neural Information Processing Systems 37*, pp. 106190–106236, 2024. doi: 10.52202/079017-3369. URL https://doi.org/10.52202/079017-3369.

Wenbo Guo, Wei Zeng, Chengwei Liu, Xiaojun Jia, Yijia Xu, Lei Tang, Yong Fang, and Yang Liu. Malskillbench: A runtime-verified benchmark of malicious agent skills. *arXiv preprint arXiv:2606.07131*, 2026. URL https://arxiv.org/abs/2606.07131.

Wenhao Hu, Jinhao Duan, Chunchen Wei, Li Zhang, Yue Zhang, and Kaidi Xu. Dynacode: A dynamic complexity-aware code benchmark for evaluating large language models in code generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 21980–21997, 2025. doi: 10.18653/v1/2025.findings-acl.1133. URL https://doi.org/10.18653/v1/2025.findings-acl.1133.

Li Huang, Zhongxin Liu, Yifan Wu, Tao Yin, Dong li, Jichao Bi, Nankun Mu, Hongyu Zhang, and Meng Yan. Deepguard: Secure code generation via multi-layer semantic aggregation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19793–19813, 2026. doi: 10.18653/v1/2026.acl-long.907. URL https://doi.org/10.18653/v1/2026.acl-long.907.

Zimo Ji, Congying Xu, Zongjie Li, Yudong Gao, Xin Wei, Shuai Wang, and Shing-Chi Cheung. Cloak and detonate: Scanner evasion and dynamic detection of agent skill malware. *arXiv preprint arXiv:2607.02357*, 2026. URL https://arxiv.org/abs/2607.02357.

Feiran Jia, Tong Wu, Xin Qin, and Anna Squicciarini. The task shield: Enforcing task alignment to defend against indirect prompt injection in llm agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 29680–29697, 2025. doi: 10.18653/v1/2025.acl-long.1435. URL https://doi.org/10.18653/v1/2025.acl-long.1435.

Pascal Junod, Julien Rinaldini, Johan Wehrli, and Julie Michielin. Obfuscator-llvm – software protection for the masses. In *2015 IEEE/ACM 1st International Workshop on Software Protection*, pp. 3–9, 2015. doi: 10.1109/spro.2015.10. URL https://doi.org/10.1109/spro.2015.10.

Haoyang Li, Mingjin Li, Jinxin Zuo, Siqi Li, Xiao Li, Hao Wu, Yueming Lu, and Xiaochuan He. Cotdeceptor:adversarial code obfuscation against cot-enhanced llm code agents. *arXiv preprint arXiv:2512.21250*, 2025a. URL https://arxiv.org/abs/2512.21250.

Qinfei Li, Xiaoxuan Dong, Jin Zhang, Dexu Yu, Wenhao Deng, Junchen Fu, Youhua Li, Hanwen Du, and Chunxiao Li. Risk-aware reranking for agentic tool retrieval. *arXiv preprint arXiv:2608.22751*, 2026a. URL https://arxiv.org/abs/2608.22751.

Shawn Li, Chenxiao Yu, Han Wang, Wei Yang, Ryan Rossi, Franck Dernoncourt, Xiyang Hu, Philip Yu, Chaowei Xiao, Huan Zhang, and Yue Zhao. Fortis: Benchmarking over-privilege in agent skills. *arXiv preprint arXiv:2605.09163*, 2026b. URL https://arxiv.org/abs/2605.09163.

Xiao Li, Yue Li, Hao Wu, Yue Zhang, Yechao Zhang, Fengyuan Xu, and Sheng Zhong. A systematic study of code obfuscation against llm-based vulnerability detection. *arXiv preprint arXiv:2512.16538*, 2025b. URL https://arxiv.org/abs/2512.16538.

Cullen Linn and Saumya Debray. Obfuscation of executable code to improve resistance to static disassembly. In *Proceedings of the 10th ACM conference on Computer and communication security - CCS '03*, 2003. doi: 10.1145/948148.948149. URL https://doi.org/10.1145/948148.948149.

Seyedreza Mohseni, Seyedali Mohammadi, Deepa Tilwani, Yash Saxena, Gerald Ketu Ndawula, Sriram Vema, Edward Raff, and Manas Gaur. Can llms obfuscate code? a systematic analysis of large language models into assembly code obfuscation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(23):24893–24901, 2025. doi: 10.1609/aaai.v39i23.34672. URL https://doi.org/10.1609/aaai.v39i23.34672.

Yutao Mou, Xiao Deng, Yuxiao Luo, Shikun Zhang, and Wei Ye. Can you really trust code copilot? evaluating large language models from a code security perspective. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17349–17369, 2025. doi: 10.18653/v1/2025.acl-long.849. URL https://doi.org/10.18653/v1/2025.acl-long.849.

Yuzhou Nie, Zhun Wang, Yu Yang, Ruizhe Jiang, Yuheng Tang, Xander Davies, Yarin Gal, Bo Li, Wenbo Guo, and Dawn Song. Secodeplt: A unified benchmark for evaluating the security risks and capabilities of code genai. In *Advances in Neural Information Processing Systems 38*, pp. 14835–14877, 2025. doi: 10.52202/085713-0447. URL https://doi.org/10.52202/085713-0447.

OpenAI. Codex agent skills, 2026. URL https://developers.openai.com/codex/skills. Official documentation; accessed 2026-09-16.

OpenCode Project. Opencode skills, 2026. URL https://opencode.ai/docs/skills. Official documentation; accessed 2026-09-16.

Shishir Patil, Tianjun Zhang, Xin Wang, and Joseph Gonzalez. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems 37*, pp. 126544–126565, 2024. doi: 10.52202/079017-4020. URL https://doi.org/10.52202/079017-4020.

Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. Asleep at the keyboard? assessing the security of github copilots code contributions. In *2022 IEEE Symposium on Security and Privacy (SP)*, pp. 754–768, 2022. doi: 10.1109/sp46214.2022.9833571. URL https://doi.org/10.1109/sp46214.2022.9833571.

Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. Do users write more insecure code with ai assistants? In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2785–2799, 2023. doi: 10.1145/3576915.3623157. URL https://doi.org/10.1145/3576915.3623157.

Hongzhou Rao, Zikan Dong, Yanjie Zhao, Haodong Li, and Haoyu Wang. Acoda: Adversarial code obfuscation for defending against llm-based analysis. *arXiv preprint arXiv:2606.11755*, 2026. URL https://arxiv.org/abs/2606.11755.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems 36*, pp. 68539–68551, 2023. doi: 10.52202/075280-2997. URL https://doi.org/10.52202/075280-2997.

Jiawen Shi, Zenghui Yuan, Guiyao Tie, Pan Zhou, Neil Gong, and Lichao Sun. Prompt injection attack to tool selection in llm agents. In *Proceedings 2026 Network and Distributed System Security Symposium*, 2026. doi: 10.14722/ndss.2026.230675. URL https://doi.org/10.14722/ndss.2026.230675.

Mohammed Latif Siddiq and Joanna C. S. Santos. Securityeval dataset: mining vulnerability examples to evaluate machine learning-based code generation techniques. In *Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security*, pp. 29–33, 2022. doi: 10.1145/3549035.3561184. URL https://doi.org/10.1145/3549035.3561184.

Adrian Swindle, Derrick McNealy, Giri Krishnan, and Ramyaa Ramyaa. Evaluation of large language models on code obfuscation (student abstract). *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(21):23664–23666, 2024. doi: 10.1609/aaai.v38i21.30517. URL https://doi.org/10.1609/aaai.v38i21.30517.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), 2024. doi: 10.1007/s11704-024-40231-1. URL https://doi.org/10.1007/s11704-024-40231-1.

Lichao Wang, ZhaoXing Ren, Tianzhuo Yang, Jiaming Ji, Chi Harold Liu, Yaodong Yang, and Juntao Dai. Safemcp: Proactive power regulation for llm agent defense via environment-grounded look-ahead reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11374–11396, 2026a. doi: 10.18653/v1/2026.acl-long.522. URL https://doi.org/10.18653/v1/2026.acl-long.522.

Yanlin Wang, Ziyao Zhang, Chong Wang, Xinyi Xu, Mingwei Liu, Yong Wang, Jiachi Chen, and Zibin Zheng. Realsec-bench: A benchmark for evaluating secure code generation in real-world repositories. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 35866–35883, 2026b. doi: 10.18653/v1/2026.findings-acl.1786. URL https://doi.org/10.18653/v1/2026.findings-acl.1786.

Yue Wang, Yi Liu, Gelei Deng, Ying Zhang, Yuekang Li, Zhenyu Chen, and Leo Zhang. Maliciousskillbench: A comprehensive benchmark for malicious agent skill detection. *arXiv preprint arXiv:2608.19901*, 2026c. URL https://arxiv.org/abs/2608.19901.

Babak Yadegari, Brian Johannesmeyer, Ben Whitely, and Saumya Debray. A generic approach to automatic deobfuscation of executable code. In *2015 IEEE Symposium on Security and Privacy*, pp. 674–691, 2015. doi: 10.1109/sp.2015.47. URL https://doi.org/10.1109/sp.2015.47.

Kaiyue Yang, Yuyan Bu, Jingwei Yi, Yuchi Wang, Biyu Zhou, Juntao Dai, Songlin Hu, and Yaodong Yang. When lower privileges suffice: Investigating over-privileged tool selection in llm agents. *arXiv preprint arXiv:2606.20023*, 2026. URL https://arxiv.org/abs/2606.20023.

Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, Rui Wang, and Gongshen Liu. R-judge: Benchmarking safety risk awareness for llm agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 1467–1490, 2024. doi: 10.18653/v1/2024.findings-emnlp.79. URL https://doi.org/10.18653/v1/2024.findings-emnlp.79.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics ACL 2024*, pp. 10471–10506, 2024. doi: 10.18653/v1/2024.findings-acl.624. URL https://doi.org/10.18653/v1/2024.findings-acl.624.

Boyu Zhang, Tianyu Du, Junkai Tong, Xuhong Zhang, Kingsum Chow, Sheng Cheng, Xun Wang, and Jianwei Yin. Seccoder: Towards generalizable and robust secure code generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 14557–14571, 2024a. doi: 10.18653/v1/2024.emnlp-main.806. URL https://doi.org/10.18653/v1/2024.emnlp-main.806.

Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. Agent-safetybench: Evaluating the safety of llm agents. *arXiv preprint arXiv:2412.14470*, 2024b. URL https://arxiv.org/abs/2412.14470.

A Additional sampling and transformation details

The 28 malicious source Skills comprise B1–B9 counts of 2, 3, 2, 1, 4, 3, 3, 3, and 7. The representative benign Skills comprise four Python, three JavaScript, and three Shell bundles; seven single-file and three multi-file; and four small, five medium, and one large artifact. Deterministic builders verify one-to-one source/variant identifiers, transform every declared script, and recover the exact original bytes from O1–O4 without executing candidate code. M0 contains no declared implementation. Manifests and data partitions are fixed before execution.

B Outcome codebook and validity checks

The event codebook distinguishes implementation-path access, execution intent, security concern, explicit refusal, and enforcement. A blocked action is not counted as model recognition, and a security statement is not counted as an execution attempt. The positive-event provenance audit maps every reported security concern and refusal to an assistant-authored native trace record. Concern-only contrasts and benign controls provide specificity checks; because negative traces are not exhaustively annotated by independent raters, recall remains an explicit measurement limitation. Path access includes reads, listings, and searches under `scripts/`; consequently, M0 positives mean

that the agent probed a missing or empty implementation location. A provided command counts as uptake only when it names a bundled script and runtime; it is reported separately from an execution attempt.

C ADDITIONAL STATISTICAL DETAILS

The artifact contains one row for each of the 3,648 planned cells, together with aggregate and model/harness rates, B1–B9 denominators, Wilson intervals, paired contrasts, cluster-bootstrap draws, and randomization tests. All cells are technically complete. The main O4–O0 detection contrast remains negative under the concern-only definition, while the pooled unsafe-uptake interval includes zero.