TraceCaps: Inline Provenance and Risk Enforcement for Agentic Software Engineering

Andre Catarino*
Faculty of Engineering, University of Porto
Porto, Portugal
up202408593@up.pt

Claudia Mamede
Carnegie Mellon University & FEUP
Pittsburgh, USA
cmamede@andrew.cmu.edu

Rui Melo
Carnegie Mellon University & FEUP
Pittsburgh, USA
rmelo@cs.cmu.edu

Rui Abreu
Faculty of Engineering University of Porto
Porto, Portugal
rma@up.pt

Abstract

Large Language Model (LLM) agents are increasingly embedded in software engineering (SE) workflows—planning, coding, testing, and CI/CD. Failures are frequent: prompt injection, unsafe tool use, supply-chain contamination, and memory poisoning. Existing defences —such as static analysers, provenance attestation, and prompt guardrails—are insufficient: they typically audit after-the-fact or operate without cryptographic guarantees or runtime enforcement. We propose *TraceCaps*, a runtime approach that (i) attaches *cryptographically verifiable provenance capsules* to each agent step (e.g. prompt, memory, tool call), and (ii) computes a *monotone, persistent risk score* that gates tool actions inline via policy thresholds (*allow, warn, block*). Capsules hash and sign events, link to parents, and embed risk features; an accumulator prevents "risk laundering" by subsequent benign steps. Early demonstrations on SWE-bench illustrate how *TraceCaps* can expose unsafe behaviors and apply runtime governance through risk accumulation, pointing toward enforceable agent safety. To our knowledge, *TraceCaps* is the first approach to *bind provenance and risk into a single cryptographic substrate*, pointing toward a shift from passive audit to runtime, enforceable safety in agentic SE workflows.

ACM Reference Format:
Andre Catarino, Claudia Mamede, Rui Melo, and Rui Abreu. 2026. TraceCaps: Inline Provenance and Risk Enforcement for Agentic Software Engineering. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE-NIER '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3786582.3786832

1 Introduction

Large Language Models (LLMs) are integrated into software engineering tasks, from code completion in IDEs to automated testing and CI/CD [17, 19, 36]. These integrations are evolving toward AI agents that plan, code, and deploy with greater independence [5, 10].

Unlike deterministic pipelines, AI agents operate with greater autonomy: they perceive inputs, plan multi-step tasks, and invoke external tools at runtime. This makes their executions inherently more complex and unpredictable, complicating efforts to monitor and audit their behavior [6]. It also hinders our ability to verify where each action came from and how it influenced the workflow (i.e., *provenance*), among other problems like prompt injections [11, 21], unsafe tool calls [20, 46], and poisoned memories [7, 30].

Current provenance mechanisms, for example, in-toto [44] and SLSA [43]), as well as standards like the W3C PROV Data Model [1], secure software supply chains by recording how artifacts are produced. While existing provenance mechanisms provide strong guarantees in deterministic pipelines, they are not suited to the dynamic workflows of AI agents. Current provenance models assume a fixed dataflow of inputs and outputs, but agentic execution involves branching (model's decision points) and mutable states. As a result, prompts, intermediate responses, tool calls, or memory updates do not fit into traditional provenance records [42]

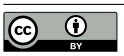
Compounding this, most real-world implementations of provenance frameworks still rely on mutable logs that are not cryptographically bound; consequently, these records can be altered, allowing attackers to disguise malicious actions [27]. The risks of such tampering are not hypothetical. The SolarWinds breach,[1] though unrelated to AI agents, illustrates the stakes: corrupted provenance in the build process allowed a backdoored but digitally signed update to propagate undetected, affecting governmental agencies and diminishing trust across the entire supply chain.

With AI agents, this risk is amplified because (1) agents make dynamic decisions and interact with untrusted external entities, and (2) they are already widely deployed in software engineering and enterprise workflows. Without clear attribution to human controllers, principled limits on agent powers, and tamper-evident observability of actions, attackers can manipulate provenance to conceal malicious behavior within otherwise benign workflows. For instance, if an attacker alters records of an agent's actions, such as which tools it invoked, which content it retrieved, or how its memory was updated, malicious steps could leave downstream users and systems relying on compromised outputs with little or no reliable trail for investigation or accountability.

We argue that the solution requires safeguards that work together: *cryptographic binding* to ensure records cannot be altered, *real-time enforcement* so actions are immutably recorded as they occur, and *continuous risk monitoring* to detect and escalate suspicious behavior. None of these alone is sufficient; only their combination can provide the foundations for trustworthy agent systems.

So, we introduce *TraceCaps*, a runtime approach that adapts classical provenance and risk tracking techniques to the dynamic setting of AI agents. Each agent step (e.g., prompt, memory update, or tool call) is logged, signed, and hash-linked, and also contributes to a cumulative monotonic risk score that prevents malicious actions

*Corresponding author.



This work is licensed under a Creative Commons Attribution 4.0 International License.
ICSE-NIER '26, Rio de Janeiro, Brazil
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2425-1/26/04
https://doi.org/10.1145/3786582.3786832

[1] SolarWinds breach: https://www.gao.gov/blog/solarwinds-cyberattack-demands-significant-federal-and-private-sector-response-infographic

from being concealed by subsequent benign ones. Policy thresholds (allow, warn, block) then gate actions inline. In this way, *TraceCaps* transforms provenance from a passive audit trail into an enforceable runtime safety mechanism, ensuring that risky agent behaviors are detectable and bounded in practice.

2 TraceCaps

TraceCaps is a runtime layer interposed between agents and their tools/memory that combines cryptographically verifiable provenance with monotone risk enforcement.

Each agent step is encapsulated in a signed hash-linked capsule (see Section 2.1), while its characteristics contribute to a cumulative risk score that cannot be reduced by later benign actions (see Section 2.2). All records are then stored in a ledger and available for auditing. To guide our design, we draw on three principles for secure AI agents [8] – **(1) well-defined human controllers**, **(2) limits on agent power** and **(3) observability of actions**. These principles motivate the three components of TraceCaps, which we will discuss in the next subsections.

This design overcomes the limitations of prior approaches: provenance frameworks are post-hoc and audit-only [28, 29, 31], logs often lack cryptographic binding and are mutable [14], and current runtime monitors enforce heuristics without guarantees [41].

2.1 Cryptographic Capsules

TraceCaps represent each agent step as a *provenance capsule*

$$C_i = \langle e_i, h_{i-1}, \sigma_i \rangle$$

where e_i is a structured event restricted to one of three types: (1) prompt (i.e., what flows between the agent and the model, such as a user query or AI-generated code fix), (2) tool call (i.e., what flows between the agent and the outside world, like running a shell command or calling an API), or (3) memory operation (i.e., what flows between the agent and its internal state, such as storing a summary for later use or retrieving it when making a decision). h_{i-1} is the digest of the previous capsule, $C_i - 1$, ensuring that each new event is anchored to its predecessor; attempts to remove or modify events break the linkage, much like altering a block in a blockchain invalidates all subsequent blocks [12, 25]. σ_i is a digital signature over $(e_i \parallel h_{i-1})$ that binds the current event to its predecessor using a cryptographic key, ensuring that only an authorized signer can produce valid capsules.

Focusing again on the SolarWinds breach, *TraceCaps* would mitigate the attack in the following way: every intermediate build action would be captured as an event (e_i), making the malicious injection itself visible rather than hidden inside the final binary; the hash linking ($h_i - 1$) ensures that no event can be deleted or reordered without breaking the chain, preventing attackers from silently erasing the injected step; finally, digital signatures (σ_i) would bind each event to a legitimate key holder, so a compromised tool could not forge or regenerate a clean chain without detection.

This design directly advances two of the security principles. First, it supports well-defined human controllers by attributing each action to a valid signer, making responsibility and authorship explicit. Second, it ensures observability of actions by turning the workflow into a tamper-evident chain: any attempt to modify or delete events leaves a detectable break.

2.2 Risk as a Monotone Recurrence

TraceCaps enforces a non-decreasing risk function, adapting classic security principles to agent workflows. Prior work establishes that once a risky action appears in the sequence of events, later actions cannot erase it [23, 37]. We adopt the same stance: each event may raise cumulative risk, but subsequent benign steps cannot lower it.

Recurrence. We define the cumulative risk after each event as:

$$R_i = \max\Big(R_{i-1},\ f(\phi(e_i)) + \lambda \cdot R_{i-1}\Big), \tag{1}$$

where i indexes the current event in the sequence, R_{i-1} is the prior cumulative risk, $\phi(e_i)$ extracts features from the current event and $f(\bullet)$ maps those features into a bounded risk contribution in [0,1], and $\lambda \cdot R_{i-1}$ is a propagation factor (recurrence weight) that accounts for causal amplification: once suspicious behavior is observed, subsequent related actions inherit part of that suspicion. The parameter $\lambda \in [0, 1)$ controls how strongly prior risk propagates forward. When $\lambda = 0$, risk is purely event-local; higher values cause suspicion to persist across subsequent actions, modeling amplification of prior risk across agent steps. In our illustrative example, λ is set conservatively to favor stability over rapid escalation. The risk of the first event e_0 is $R_0 = f(\phi(e_0))$; λ applies only from e_1 onward.

> "By construction, the use of *max* ensures that R_i can only increase or plateau, ensuring monotonicity and preventing adversaries from "laundering" (i.e., hiding) malicious steps among safe-looking ones."

Features for Risk Scoring. We score each event using three complementary metrics. Prior work provides evidence that these metrics are informative for code and natural language, allowing *TraceCaps* to operate across modalities.

- **Negative Log Likelihood (*NLL*)** measures how unnatural an event is for a baseline language model [34]. Buggy or low-quality code tends to have higher NLL [35, 38], while AI-generated text also shows distinct perplexity signatures [24]. In *TraceCaps*, an unusually high NLL may indicate buggy or low-quality code; or, in text events, injected or machine-manipulated passages.
- **Shannon entropy (*E*)** quantifies the unpredictability of tokens within a single event [40]. Prior work shows that high entropy is often associated with obfuscation or hidden payloads in code [9, 33], and has also been used to identify ciphertext or anomalous strings in text streams [47]. In our setting, elevated entropy scores can serve as indicators of constructions that may conceal or smuggle information – e.g., when tool outputs embed base64/hex literals, minified code, or when prompt/memory events contain hashes or symbol-heavy text.
- **Mutual information (*MI*)** measures the statistical dependence between two events [4, 39]. Unlike NLL, which captures how improbable an event is, or entropy, which reflects its internal randomness, MI surfaces correlations with surrounding context. Prior work has shown that variants of MI can help locate vulnerabilities and reveal strong token or API associations that, when misused, indicate risk [26]. In our setting, elevated MI scores can

serve as indicators of suspicious reuse – e.g., when a tool call reintroduces sensitive material from recent context (such as an API key read from memory), or when a prompt event echoes prior instructions or sensitive fragments.

We rely on three complementary metrics as each captures a different dimension of risk: improbability, internal randomness, and contextual dependence. Using them together provides better coverage than any single metric alone and is a balanced and stable way to capture diverse risk patterns across events. To combine the three metrics into a single score, we first normalize them to $[0, 1]$ for comparability, then apply a root-mean-square (RMS) operator:

$$f(\phi(e_i)) = \sqrt{\tfrac{1}{3}\left(NLL^2 + E^2 + MI^2\right)}. \tag{2}$$

RMS emphasizes strong anomalies (so a single suspicious signal cannot be diluted by weaker ones) while still accounting for cumulative effects when multiple features are moderately elevated.

Stability. Because each contribution in Eq. 1 is bounded in $[0, 1]$, the recurrence cannot diverge. If risk features settle around a constant c, the score converges to a stable ceiling:

$$R_i \;\to\; R^\star = \min\left(1,\ \tfrac{c}{1-\lambda}\right), \tag{3}$$

with λ as the recurrence weight defined earlier. Intuitively, long runs of benign events keep the score flat, while occasional high-risk events push it upward but never beyond 1. Risk thus reflects the severity of actions rather than quantity – executing more steps cannot inflate it.

2.3 Policies

TraceCaps policies turn cumulative risk scores into three runtime decisions – allow, warn, block. Formally, for event e_i with risk R_i, *TraceCaps* compares R_i against thresholds T_{warn} and T_{block} to decide wether to allow, warn or block an action. Thresholds can be set globally, per state, or per tool, giving operators fine-grained control without breaking the cryptographic chain of provenance.

The novelty lies not in these mechanisms themselves, but in how *TraceCaps* binds them into a monotone, tamper-evident chain. Once an action is deemed risky, that judgment cannot be erased by later benign steps, and every denial is cryptographically recorded. This prevents "risk laundering" and ensures that policy outcomes remain verifiable and auditable at runtime.

Together with the monotone recurrence, these policies enforce the principle of limiting agent powers: cumulative risk cannot be reset, and once thresholds are crossed, further actions are cut off.

False-Positive and False-Negative Tradeoff. Inevitably, different thresholds shift the balance between false positives and false negatives: stricter settings reduce undetected attacks but may disrupt workflows, while permissive ones keep developers moving but risk letting manipulations go unnoticed. We make this trade-off explicit and tunable, moving enforcement from opaque heuristics to verifiable, auditable decisions.

3 Illustrative Example

To illustrate TraceCaps in practice, we instrumented RepairAgent [2]. RepairAgent cycles through three phases: understanding a bug, collecting information, and proposing fixes. Each phase mixes code-oriented actions (e.g., reading or modifying source files, running tests) with text-oriented actions (e.g., generating hypotheses in natural language, reasoning about errors). This dual modality motivates our use of risk metrics that capture both text and code.

Fig 1 summarizes an execution of the instrumented agent for the `django__django-11964` task in SWE-Bench.[2] Although the actual run produced 23 capsules across 8 iterations, we highlight a subset of representative events (E3, E5, E7) to show how the risk score evolved. The dashed branch (E9) illustrates how TraceCaps would enforce a stop once the block threshold is crossed.

Each tool call is encapsulated in a capsule and contributes to a risk score. Consequently, at E3, the risk score (0.393) is below the warning threshold and action proceeds. At E5 (0.508) and E7 (0.651), the score is above the warning threshold but below the block threshold, so actions continue under warning. The dashed path shows a hypothetical E9, where the score (0.851) would cross the block threshold. In this case, *TraceCaps* would signal the agent to stop and record the denial in the provenance chain. Because the score is monotone, later steps cannot reduce earlier risks. Capsule creation and hash chaining are linear in the number of agent steps and require no global synchronization, avoiding bottlenecks in long-running agent trajectories.

Audit Trail. The session produced a verifiable ledger of 6 chained capsules, each cryptographically binding an event to its predecessor. This ledger encodes both the sequence of actions and the corresponding security decisions.

```
"tool_name": "goal_accomplished",
"summary": "Repair process completed after 8 tool
↪  executions with 6 successful operations",
"agent_state": "try_fix",
"risk_score": 0.651,
"capsule_hash": "0813230660c34c9e...",
"parent_capsule": "bf4f5d67...",
"decision": "warn"
```

The capsule is then referenced in the audit log, which records its occurrence in the agent's timeline:

```
"event_type": "tool",
"agent_id": "llm_django_11964_repair",
"event_id": "f1782ae5...",
"parent_capsule": "bf4f5d67...",
"payload_summary": "{'tool_name': 'goal_accomplished',
↪  ...}",
"risk_score": "0.651",
"capsule_hash": "0813230660c34c9e..."
```

These artifacts demonstrate how TraceCaps records each step of the repair workflow in a cryptographically linked chain. If the pipeline completes, we treat the repair as successful; if it halts, it is because TraceCaps signaled a stop once the block threshold was exceeded. In either case, the outcome is accompanied by a verifiable ledger of actions and decisions, turning opaque agent behavior into an auditable and enforceable process.

[2]The sample can be found at https://huggingface.co/datasets/SWE-bench/SWE-bench

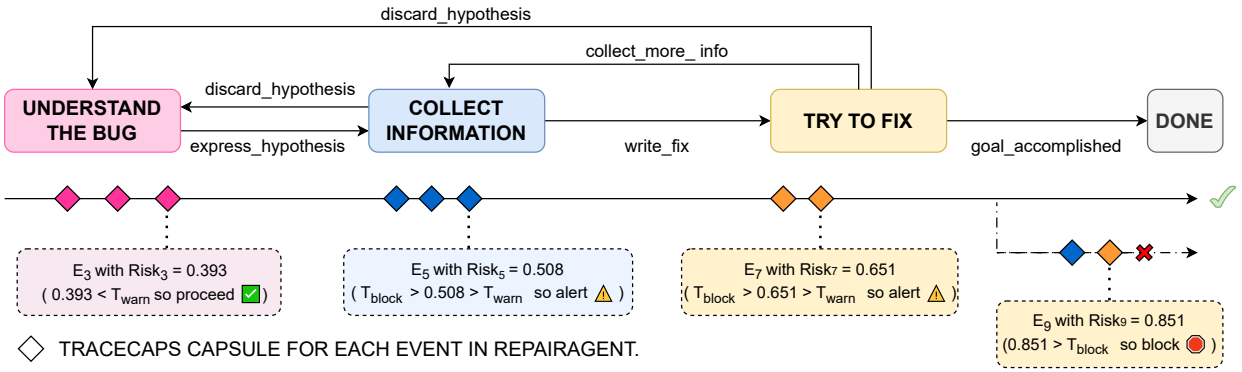


Figure 1: RepairAgent workflow with TraceCaps overlay. Solid lines show the observed execution, where each tool call is encapsulated in a cryptographically chained capsule and contributes to a monotone cumulative risk score. The dashed line illustrates a hypothetical scenario in which an event exceeds the block threshold. TraceCaps would then enforce a stop, preventing further tool use and recording the denial in the provenance chain.

4 Related Work

TraceCaps combines provenance with runtime risk enforcement, extending beyond static analysis, heuristic monitors, and post-hoc attestation to provide tamper-evident governance for the dynamic workflows of LLM-based SE agents.

Comparison with Conventional Tools. Static analysis tools (e.g., SonarQube, Bandit) excel at identifying insecure code fragments, but they operate on isolated snapshots of code. These tools are ill-suited to agentic workflows, where risk emerges from plans, evolving prompts, and tool orchestrations [45]. Moreover, these tools implicitly trust the integrity of the logs or metrics they consume, offering no guarantees against tampering or omission. In contrast, *TraceCaps* binds each step into a tamper-evident chain, ensuring audit trails remain trustworthy even under manipulation.

Provenance and secure logging. Systems such as in-toto [44], SLSA [43], and Sigstore [22] attest to supply chain integrity, while secure provenance and logging frameworks [15, 48] prevent forgery through tamper-evident trails. These efforts, however, target post-hoc accountability in static workflows. *TraceCaps* is complementary: it brings cryptographically bound provenance to *runtime actions of LLM-based SE agents*, coupling them with monotone risk enforcement so unsafe behaviors are governed inline rather than post-hoc.

Runtime Enforcement Mechanisms. Because AI agents operate in open-ended environments, several works propose runtime monitors. *LlamaFirewall* [3] inspects final outputs with jailbreak detectors and code analyzers, filtering unsafe content. *Thought-Aligner* [18] corrects risky reasoning steps before they execute, while rule-based monitors restrict usage or flag suspicious commands [45]. These efforts promise runtime enforcement, but they lack cryptographic guarantees and do not preserve a tamper-evident audit trail. *TraceCaps* advances this line by binding each action into a cryptographically linked chain and propagating a monotone risk score, making risky behavior both enforceable and auditable.

Guardrails and LLM safety. Guardrail mechanisms attempt to constrain unsafe behaviors through prompt filters [13] and LLM-based judges [32, 49]. More recently, *TrustAgent* [16] proposes an Agent-Constitution framework that injects safety rules at multiple stages of agent planning to moderate unsafe decisions before execution. These approaches remain heuristic and opaque, lacking cryptographic verifiability. *TraceCaps* advances this line by making each action both *cryptographically bound to its predecessors* and *subject to monotone risk enforcement*, bridging the provenance guarantees of SE pipelines with the real-time guardrail requirements.

5 Conclusion

TraceCaps introduces a new way of governing agentic workflows by coupling tamper-evident provenance with a monotone risk enforcement that gates agent actions in real time. Rather than treating provenance as a passive audit trail, *TraceCaps* turns it into an active runtime mechanism that makes risky decisions visible and enforceable. However, open challenges remain, particularly around interaction paradigms for exposing provenance to developers, achieving adoption in real workflows, and the broader balance between safety, productivity, and trust. These challenges present opportunities for future research into how provenance-based enforcement can evolve into deployable infrastructure for agentic safety.

6 Future Plans

We will extend *TraceCaps* to systematically characterize how it behaves in realistic agent executions. A first step is a comprehesive empirical evaluation on representative agent trajectories, including benchmarks such as SWE-bench, quatifying runtime overhead (e.g., per-step latency and per-trace memory growth), enforcement behavior (e.g., time-to-enforcement, blocked actions in benign workflows), and task-level outcomes, including completion rates and retry frequency with and without *TraceCaps* enabled. In parallel, the risk model will be refined beyond linear recurrence to better reflect the structure of agent executions. This includes richer action taxonomies and propagation mechanisms that follow dependencies between related steps. In particular, realistic executions expose concurrency, where multiple tool invocations branch from a single parent action and require explicit parent–child tracking for consistent risk propagation. Similarly, retries after failure introduce the need to delimit prior execution context; these cases will be handled through *reset capsules* that seal earlier history and establish a clean baseline once no residual unsafe state is carried forward.

References

[1] Khalid Belhajjame, Reza B'Far, James Cheney, Sam Coppens, Stephen Cresswell, Yolanda Gil, Paul Groth, Graham Klyne, Timothy Lebo, Jim McCusker, et al. 2013. Prov-dm: The prov data model. *W3C Recommendation* 14 (2013), 15–16.

[2] Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. 2024. Repairagent: An autonomous, llm-based agent for program repair. *arXiv preprint arXiv:2403.17134* (2024).

[3] Sahana Chennabasappa, Cyrus Nikolaidis, Daniel Song, David Molnar, Stephanie Ding, Shengye Wan, Spencer Whitman, Lauren Deason, Nicholas Doucette, Abraham Montilla, et al. 2025. Llamafirewall: An open source guardrail system for building secure ai agents. *arXiv preprint arXiv:2505.03574* (2025).

[4] Thomas M Cover. 1999. *Elements of information theory*. John Wiley & Sons.

[5] Cursor AI, Inc. 2024. Cursor. https://www.cursor.sh. AI-powered code editor.

[6] Zehang Deng, Yongjian Guo, Changzhou Han, Wanlun Ma, Junwu Xiong, Sheng Wen, and Yang Xiang. 2025. Ai agents under threat: A survey of key security challenges and future pathways. *Comput. Surveys* 57, 7 (2025), 1–36.

[7] Shen Dong, Shaochen Xu, Pengfei He, Yige Li, Jiliang Tang, Tianming Liu, Hui Liu, and Zhen Xiang. 2025. A practical memory injection attack against llm agents. *arXiv preprint arXiv:2503.03704* (2025).

[8] Santiago (Sal) Díaz, Christoph Kern, and Kara Olive. 2025. *Google's Approach for Secure AI Agents*. Technical Report.

[9] Shouki A Ebad, Abdulbasit A Darem, and Jemal H Abawajy. 2021. Measuring software obfuscation quality–a systematic literature review. *IEEE Access* 9 (2021), 99024–99038.

[10] GitHub, Inc. and Microsoft Corporation. 2024. GitHub Copilot. https://github.com/features/copilot. AI-powered code completion tool.

[11] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security* (Copenhagen, Denmark) *(AISec '23)*. Association for Computing Machinery, New York, NY, USA, 79–90. doi:10.1145/3605764.3623985

[12] Stuart Haber and W. Scott Stornetta. 1991. How to time-stamp a digital document. *Journal of Cryptology* 3, 2 (01 Jan 1991), 99–111. doi:10.1007/BF00196791

[13] William Hackett, Lewis Birch, Stefan Trawicki, Neeraj Suri, and Peter Garraghan. 2025. Bypassing Prompt Injection and Jailbreak Detection in LLM Guardrails. *arXiv preprint arXiv:2504.11168* (2025).

[14] Ragib Hasan, Radu Sion, and Marianne Winslett. 2009. The Case of the Fake Picasso: Preventing History Forgery with Secure Provenance. 1–14.

[15] Ragib Hasan, Radu Sion, and Marianne Winslett. 2009. Preventing history forgery with secure provenance. *ACM Transactions on Storage (TOS)* 5, 4 (2009), 1–43.

[16] Wenyue Hua, Xianjun Yang, Mingyu Jin, Zelong Li, Wei Cheng, Ruixiang Tang, and Yongfeng Zhang. 2024. Trustagent: Towards safe and trustworthy llm-based agents. *arXiv preprint arXiv:2402.01586* (2024).

[17] Kush Jain, Gabriel Synnaeve, and Baptiste Roziere. 2024. Testgeneval: A real world unit test generation and test completion benchmark. *arXiv preprint arXiv:2410.00752* (2024).

[18] Changyue Jiang, Xudong Pan, and Min Yang. 2025. Think Twice Before You Act: Enhancing Agent Behavioral Safety with Thought Correction. *arXiv preprint arXiv:2505.11063* (2025).

[19] Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. 2023. Inferfix: End-to-end program repair with llms. In *Proceedings of the 31st ACM joint european software engineering conference and symposium on the foundations of software engineering*. 1646–1656.

[20] Daniel Jones, Giorgio Severi, Martin Pouliot, Gary Lopez, Joris de Gruyter, Santiago Zanella-Beguelin, Justin Song, Blake Bullwinkel, Pamela Cortez, and Amanda Minnich. 2025. A Systematization of Security Vulnerabilities in Computer Use Agents. arXiv:2507.05445 [cs.CR] https://arxiv.org/abs/2507.05445

[21] Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, et al. 2023. Prompt injection attack against llm-integrated applications. *arXiv preprint arXiv:2306.05499* (2023).

[22] Dan Lorenc et al. 2021. Sigstore: Software signing for everybody. https://www.sigstore.dev.

[23] Pratyusa K Manadhata and Jeannette M Wing. 2010. An attack surface metric. *IEEE Transactions on Software Engineering* 37, 3 (2010), 371–386.

[24] Alberto José Gutiérrez Megías, L Alfonso Urena Lopez, and Eugenio Martínez-Cámara. 2024. The influence of the perplexity score in the detection of machine-generated texts. In *Proceedings of the First International Conference on Natural Language Processing and Artificial Intelligence for Cyber Security*. 80–85.

[25] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. 2016. *Bitcoin and Cryptocurrency Technologies*. Princeton University Press, Princeton, NJ.

[26] Van Nguyen, Trung Le, Chakkrit Tantithamthavorn, Michael Fu, John Grundy, Hung Nguyen, Seyit Camtepe, Paul Quirk, and Dinh Phung. 2022. Statement-Level Vulnerability Detection: Learning Vulnerability Patterns Through Information Theory and Contrastive Learning. *arXiv preprint arXiv:2209.10414* (2022).

[27] Bofeng Pan, Natalia Stakhanova, and Suprio Ray. 2023. Data provenance in security and privacy. *Comput. Surveys* 55, 14s (2023), 1–35.

[28] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David Eyers, Margo Seltzer, and Jean Bacon. 2017. Practical whole-system provenance capture. In *Proceedings of the 2017 symposium on cloud computing*. 405–418.

[29] Thomas Pasquier, Xueyuan Han, Thomas Moyer, Adam Bates, Olivier Hermant, David Eyers, Jean Bacon, and Margo Seltzer. 2018. Runtime analysis of whole-system provenance. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*. 1601–1616.

[30] Atharv Singh Patlan, Ashwin Hebbar, Pramod Viswanath, and Prateek Mittal. 2025. Context manipulation attacks : Web agents are susceptible to corrupted memory. arXiv:2506.17318 [cs.CR] https://arxiv.org/abs/2506.17318

[31] Kexin Pei, Zhongshu Gu, Brendan Saltaformaggio, Shiqing Ma, Fei Wang, Zhiwei Zhang, Luo Si, Xiangyu Zhang, and Dongyan Xu. 2016. Hercule: Attack story reconstruction via community discovery on correlated log graph. In *Proceedings of the 32Nd Annual Conference on Computer Security Applications*. 583–595.

[32] Melissa Kazemi Rad, Huy Nghiem, Andy Luo, Sahil Wadhwa, Mohammad Sorower, and Stephen Rawls. 2025. Refining input guardrails: Enhancing llm-as-a-judge efficiency through chain-of-thought fine-tuning and alignment. *arXiv preprint arXiv:2501.13080* (2025).

[33] Tomer Raitsis, Yossi Elgazari, Guy E Toibin, Yotam Lurie, Shlomo Mark, and Oded Margalit. 2025. Code obfuscation: A comprehensive approach to detection, classification, and ethical challenges. *Algorithms* 18, 2 (2025), 54.

[34] Daniel Ramos, Claudia Mamede, Kush Jain, Paulo Canelas, Catarina Gamboa, and Claire Le Goues. 2025. Are large language models memorizing bug benchmarks?. In *2025 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code)*. IEEE, 1–8.

[35] Baishakhi Ray, Vincent Hellendoorn, Saheel Godhane, Zhaopeng Tu, Alberto Bacchelli, and Premkumar Devanbu. 2016. On the" naturalness" of buggy code. In *Proceedings of the 38th International Conference on Software Engineering*. 428–439.

[36] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).

[37] Fred B Schneider. 2000. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)* 3, 1 (2000), 30–50.

[38] Srinivasan Sengamedu and Hangqi Zhao. 2022. Neural language models for code quality identification. In *Proceedings of the 6th International Workshop on Machine Learning Techniques for Software Quality Evaluation*. 5–10.

[39] Claude E Shannon. 2001. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review* 5, 1 (2001), 3–55.

[40] Claude E Shannon and Warren Weaver. 1998. *The mathematical theory of communication*. University of Illinois press.

[41] Robin Sommer and Vern Paxson. 2010. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. *Proceedings - IEEE Symposium on Security and Privacy*, 305–316. doi:10.1109/SP.2010.25

[42] Renan Souza, Amal Gueroudji, Stephen DeWitt, Daniel Rosendo, Tirthankar Ghosal, Robert Ross, Prasanna Balaprakash, and Rafael Ferreira da Silva. 2025. PROV-AGENT: Unified Provenance for Tracking AI Agent Interactions in Agentic Workflows. *arXiv preprint arXiv:2508.02866* (2025).

[43] Mahzabin Tamanna, Sivana Hamer, Mindy Tran, Sascha Fahl, Yasemin Acar, and Laurie Williams. 2024. Unraveling challenges with supply-chain levels for software artifacts (SLSA) for securing the software supply chain. *Available at SSRN 5119626* (2024).

[44] Santiago Torres-Arias, Hammad Afzali, Trishank Karthik Kuppusamy, Reza Curtmola, and Justin Cappos. 2019. In-toto: providing farm-to-table guarantees for bits and bytes. In *Proceedings of the 28th USENIX Conference on Security Symposium* (Santa Clara, CA, USA) *(SEC'19)*. USENIX Association, USA, 1393–1410.

[45] Haoyu Wang, Christopher M Poskitt, and Jun Sun. 2025. Agentspec: Customizable runtime enforcement for safe and reliable llm agents. *arXiv preprint arXiv:2503.18666* (2025).

[46] Haowei Wang, Rupeng Zhang, Junjie Wang, Mingyang Li, Yuekai Huang, Dandan Wang, and Qing Wang. 2024. From allies to adversaries: Manipulating llm tool-calling through adversarial injection. *arXiv preprint arXiv:2412.10198* (2024).

[47] Adam Dorian Wong. 2023. Detecting Domain-Generation Algorithm (DGA) Based Fully-Qualified Domain Names (FQDNs) with Shannon Entropy. arXiv:2304.07943 [cs.CR] https://arxiv.org/abs/2304.07943

[48] Shams Zawoad, Amit Kumar Dutta, and Ragib Hasan. 2015. Towards building forensics enabled cloud through secure logging-as-a-service. *IEEE Transactions on Dependable and Secure Computing* 13, 2 (2015), 148–162.

[49] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems* 36 (2023), 46595–46623.