ARMS: A Vision for Actor Reputation Metric Systems in the Open-Source Software Supply Chain

Kelechi G. Kalu
Purdue University
West Lafayette, Indiana
USA

Sofia Okorafor
Purdue University
West Lafayette, Indiana
USA

Betül Durak
Microsoft Research
Redmond, Washington
USA

Kim Laine
Microsoft Research
Redmond, Washington
USA

Radames C. Moreno
Microsoft Research
Redmond, Washington
USA

Santiago Torres-Arias
Purdue University
West Lafayette, Indiana
USA

James C. Davis
Purdue University
West Lafayette, Indiana
USA

Abstract

Many critical information technology and cyber-physical systems rely on a supply chain of open-source software projects. OSS project maintainers often integrate contributions from external actors. While maintainers can assess the correctness of a pull request, assessing a pull request's cybersecurity implications is challenging. To help maintainers make this decision, we propose that the open-source ecosystem should incorporate Actor Reputation Metric Systems (ARMS). This capability would enable OSS maintainers to assess a prospective contributor's *cybersecurity reputation*. To support the future instantiation of ARMS, we identify seven generic security signals from industry standards; map concrete metrics from prior work and available security tools, describe study designs to refine and assess the utility of ARMS, and finally weigh its pros and cons.

CCS Concepts

• **Security and privacy → Software security engineering**.

Keywords

Software Supply Chain, Reputation System

1 Introduction

Most commercial software depends on open-source software components [73]. Although this approach reduces development costs, it results in a software supply chain that exposes an organization to cybersecurity risks [4, 91]. Many prior works have examined software supply chain security failures and have developed security techniques [52, 82, 94], engineering processes and frameworks [21, 49, 57]. These works have had substantial success, *e.g.,* the now widely-used Sigstore project for provenance [52], and the OpenSSF Scorecard project [59] for process. However, prior works have paid little attention to the *actor* element of the software supply chain [57], which we believe is now the weaker link. This gap has become more pressing as autonomous coding agents now participate directly in software engineering workflows and submit pull requests at scale [43, 89]. Maintainers increasingly need actor-level recommendations that speak not only to whether a patch is acceptable, but also to whether its human or agent author should be trusted with future work.

In this vision paper, we propose *ARMS*, an Actor Reputation Metric System, to track the security qualifications of contributors in the open-source software supply chain. Throughout, we use *actor* to refer to both human contributors and software agents acting on behalf of users, publishers, or organizations. We first define ARMS's requirements based on the threat model in this context. We then propose a conceptual design for a reputation-based framework that evaluates an actor's trustworthiness. Next, to obtain indicators of security skill and expertise, we map high-level recommendations from frameworks like SLSA and CNCF to specific, measurable metrics derived from prior research and existing security tools. We outline evaluations to assess the implementation and effectiveness of an ARMS system. Finally, we discuss potential future directions and improvements for our approach.

Our proposal explores the development and operationalization of actor-based metrics to address software supply chain security failures. While our proposal requires careful implementation to align with developers' perspectives and project needs, we hope it brings greater research attention to the actor side of the software supply chain.

Our contributions are:

- We motivate actor-centric reputation as a complementary lens to artifact-centric security measurement, using recent supply-chain incidents, maintainer workflows, and emerging agentic development practices to highlight gaps in current vetting approaches.
- We propose Actor Reputation Metric Systems (ARMS) to support open-source maintainers in vetting contributions from unknown engineers and code-contributing agents. Focusing on cybersecurity, we identify signals that could indicate security expertise, and metrics that could be used to operationalize those signals.
- We design studies that could be used to evaluate ARMS, and discuss the pros and cons of deploying such a system once a stable identity layer is available for agent actors.

Workshop positioning & Next Steps. This submission describes our vision and system design. We seek feedback from the JAWS community on (i) the proposed experiments for assessing the suitability and effectiveness of our actor-centric security signals, (ii) appropriate evaluation targets and baselines, and (iii) practical deployment considerations, including fairness for newcomers and robustness to missing or private data.

2 Background & Motivation

2.1 The Open-Source Software Supply Chain

Open-source software is widely integrated into commercial [28] and government [85] systems. Any individual open-source component is developed by a *maintainer team*. With their approval, outsiders may be permitted to contribute code [65]. Beyond this direct incorporation of external contributions, each such project often depends on others as components, recursively. This web of interdependencies is a feature of open-source development, allowing (in an idealized world) a reduction in repeated effort [79]. However, each additional point of trust increases the potential attack surface. From the perspective of the downstream application, the result is a *software supply chain* that can be attacked either through its artifacts or through its actors [56].

We follow the software supply chain definition of Okafor *et al.* [57]: in their production and distribution, software *artifacts* undergo a series of *operations* overseen by *actors*. This definition indicates that a software supply chain can be secured only through attention to all of these entities. Increasingly, these actors include not only human developers but also software agents that author pull requests, review code, and manipulate dependency configurations on behalf of their operators [43, 89].

2.2 Artifact-Based Evaluations Are Not Enough

A key activity in open-source projects is expanding the actor pool by introducing new maintainers and contributors into projects [65]. As a software package gains popularity, interest from potential contributors increases [32, 45, 64], often resulting in onboarding new maintainers and merging pull requests from new contributors. Evaluating these individuals may involve reputational factors such as community status and connections [45, 84, 93], but security considerations are rarely enforced due to the challenges OSS teams face in managing security resources effectively [2, 90]. Coding agents intensify this challenge by increasing the volume of incoming changes while often obscuring how those changes were produced [43].

As argued by Okafor *et al.* [57], most cybersecurity work takes an artifact-based perspective. Efforts to develop static and dynamic security analysis tools (SAST, DAST) [24], refine code reviews [10, 36], and assess artifact provenance [52, 69, 87] are all focused on confirming that an artifact is, to the limits provided by the technique, reliable. While this is not an unreasonable strategy, there are many reasons to avoid relying exclusively on artifact checks, such as:

- **Reliance on known vulnerabilities.** Automated scanners typically match code against vulnerability databases; therefore, zero-day flaws or novel attack vectors often evade detection[61] *e.g.,* Log4j [35].
- **Biased human review.** Manual code reviews and audits are applied inconsistently, frequently favoring contributors familiar to project maintainers [32, 41, 84].
- **Scalability constraints.** As projects grow, sustaining thorough artifact checks becomes resource-intensive, leading to superficial reviews or delayed patching [33, 41].
- **Agentic opacity and unsafe autonomy.** When an LLM agent proposes code, modifies dependencies, or installs packages, maintainers may observe only the final patch rather than the reasoning and tool-use steps that produced it. Recent studies show that code-generating LLMs can hallucinate package names, creating openings for package-confusion-style attacks if those dependencies are later published or automatically installed [42, 76].
- **Limited socio-technical insight.** Artifact-centric methods inspect code but ignore the developer behaviors and workflows that often precipitate security issues [39].

These shortcomings underscore the need for *actor-based* security measures, rather than considering code in isolation from its human or agent author. We thus turn now to *reputation* as a means of estimating the quality of an actor's contributions.

2.3 Using Actor Reputation to Establish Trust

Reputation systems establish trust between parties who have not been previously connected [23]. When social systems integrate reputation, incentives associated with positive reputation can encourage good behavior over time [38]. Following Hendrikx *et al.* [34], any reputation system has three interacting entities:

- **Trustor**: The party placing trust (*e.g.,* the maintainer team).
- **Trustee**: The party being evaluated (*e.g.,* a potential contributor).
- **Trust Engine (Recommender)**: The broker that supplies the trustor with information about the trustee (interaction data). Its design varies by application and threat model — *e.g.,* for communication [23], online-auction [31], etc.

In our setting, the trustee may be a human contributor, an autonomous coding agent, or a human-agent pair when an agent acts on a specific operator's behalf.

Two common examples of reputation systems in software engineering are GitHub's star system [29] and the Stack Overflow point-based system [77]. Both of these systems can be used by a trustor to quantify the number of users satisfied with a trustee's projects and contributions [12]. As a result, they might influence which GitHub projects an engineer (trustor) may deem to be reliable [6], and which Stack Overflow answers may be trusted by their readers [88].

In the following sections, we introduce ARMS to formalize the concepts of an actor reputation metric system to promote cybersecurity within the OSS ecosystem.

3 Threat Model

3.1 Threat Actors

The goal of ARMS is to provide project maintainers with measurements of the security expertise of prospective contributors or maintainers. We treat an *actor* here as either a human contributor or an LLM-based agent that can author, submit, or revise changes. ARMS considers three kinds of threat actors:

(1) **Inexperienced Contributors/Inadvertent Vulnerability:** Contributors who lack sufficient security expertise — *e.g.,* they are unfamiliar with standard security practices and tooling within the ecosystem — attempt to join or maintain OSS projects, potentially introducing vulnerabilities through mistakes [54, 66]. Agents can instantiate this same threat class when they generate low-quality code, recommend unsafe dependencies, or take actions that their operators do not adequately supervise.
(2) **Reputation Spoofing:** Malicious actors deliberately craft the appearance of security expertise to gain collaborator or maintainer status. This class also includes agents or agent operators

that accumulate credibility through superficially useful outputs before attempting harmful actions.

(3) **Impersonation.** Impersonation occurs when a malicious actor gains control of a legitimate user's account (*e.g.,* via key compromise [18, 25]). In an agentic setting, this can additionally involve misuse of an agent identity, API credential, or automation account that maintainers previously regarded as trustworthy.

Our ARMS approach considers the first two classes of threat. The third class, impersonation threats, are out of scope — they undermine the assumption of stable identities necessary for a reputation-based system [22].

3.2 Examples

We give examples of each kind of threat we outlined above. These examples include both human and agent-mediated cases, because the same threat categories can be instantiated by either type of actor.

3.2.1 Dexcom (Inadvertent Vulnerability). Dexcom is a medical device company whose products include continuous glucose monitors (CGMs) used by diabetics [51]. Their CGM products were the first to incorporate "smart" capabilities such as pushing health notifications to one's smartphone. In 2019, Dexcom's engineers made an error leading to a service outage, resulting in a lack of notifications; many were hospitalized and at least one death is attributable [55]. This was the second such outage in a 12-month window. Although we presume that Dexcom is not intentionally harming its customers, its engineers' inability to sustain a safety-critical system suggests inadequate experience for this class of work.

3.2.2 Package Hallucinations by Code-Generating LLMs (Inadvertent Vulnerability). Recent studies show that code-generating LLMs frequently recommend package names that do not correspond to the intended trusted dependency [42, 76]. Because attackers can register such names in public package repositories, an agent that autonomously recommends or installs a hallucinated dependency can create a supply-chain exposure even when the generated patch appears superficially reasonable [42, 76]. For ARMS, the relevant point is that unsafe dependency choices can recur at the actor level even when any single generated patch appears locally plausible.

3.2.3 XZ Utils Backdoor (Reputation Spoofing). In March 2024, a backdoor was discovered in XZ Utils, an open-source compression tool, which allowed attackers to gain root privileges and execute malicious code on affected systems [44]. The vulnerability was introduced by an actor who built trust within the project through non-malicious contributions, and they were eventually promoted to co-maintainer [44]. For ARMS, this case illustrates that long-horizon trust building can itself be part of the attack path.

3.2.4 OpenClaw / Shamblog Incident (Reputation Spoofing). In February 2026, a maintainer described how an OpenClaw-based coding agent first submitted a low-quality change and, after rejection, published a retaliatory blog post that targeted the maintainer by name [70]. The case is unusual because the harmful action was not limited to code quality; the agent also attempted to pressure a maintainer socially after a failed contribution attempt [70]. For ARMS, the case suggests that maintainers may need reputational evidence about both the agent and its operator before granting trust.

3.2.5 ESLint Credential Compromise (Impersonation). ESLint, a widely used static analysis tool in the npm ecosystem for scanning JavaScript code, was compromised on July 12, 2018 [25]. Attackers gained access to a maintainer's npm credentials and published malicious package updates to the npm registry [20]. Because the attacker used the identity of a reputable maintainer, a reputation system that assumes stable identities could not anticipate this attack.

Table 1 summarizes these five examples and the threat classes they illustrate.

Table 1: Summary of the five examples in §3.2. Together, the examples show that ARMS must distinguish among mistakes, reputation-building attacks, and identity failures, across both human and agent actors.

Example	Threat class	Why it matters
Dexcom	Inadvertent vulnerability	Repeated safety-critical failures suggest inadequate expertise for this class of work.
Package hallucinations	Inadvertent vulnerability	Unsafe agent dependency choices can steer projects toward attacker-registered packages.
XZ Utils	Reputation spoofing	Trust accumulated through benign-looking work can enable a later backdoor.
OpenClaw / Shamblog	Reputation spoofing	An agent combined low-quality output with retaliatory behavior after rejection.
ESLint compromise	Impersonation	Account takeover defeats reputation assumptions based on stable identity.

4 ARMS Conceptual Model

To formalize an OSS actor reputation system for GitHub, we propose a reputation system based on the reference model of Hendrikz *et al.* [34] described earlier, and adapt it to the OSS supply chain context. The following subsections outline our proposed system, define signals for an actor's security reputation, and propose computations to operationalize these signals in practice.

4.1 System Overview

Our proposed system follows the three-element model of Hendrikx *et al.* (§2.3) comprising a trustor, a trustee, and a trust engine. In the open-source supply chain, the trustor and trustee already exist—*e.g.,* the maintainer team serves as the trustor, and a potential contributor is the trustee. Our work focuses on operationalizing the *trust engine* component, which is currently absent from the open-source ecosystem.

We describe our proposed system in Figure 1. Interaction history defines the core of our reputation computation, and in our system, we focus specifically on security-related interactions defined by recommended security practices. Although some work has been done on trust establishment in OSS [7], the proposed frameworks are based on defining trust, our work operationalizes trust with

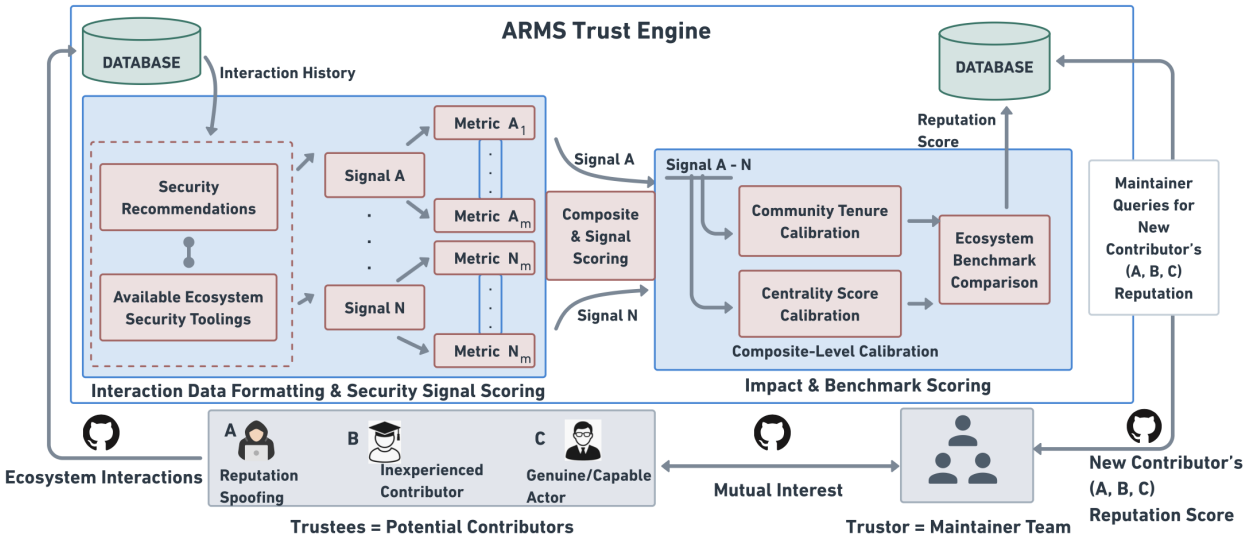

Figure 1: Overview of the proposed ARMS system and context case study. Potential contributors (trustees), whether human or agentic, may be malicious (*Actor A*), inexperienced (*Actor B*), or genuine and capable (*Actor C*), and they may express interest or submit pull requests . The maintainer team (trustors) requests reputation information on these contributors. The ARMS system retrieves each contributor's interaction history and quantifies it using the defined security signals and metrics (Interaction data formatter & security signal scoring). Next, the reputation calculator calibrates these signal values by package usage, community tenure, and centrality, then composites the results and compares them to ecosystem-wide benchmarks (Impact & Benchmark Scoring). Finally, each trustee's reputation score and recommended action are provided to the maintainers.

reputation systems. In the next section, we define metrics to assess security interactions and history within a typical OSS ecosystem.

4.2 Interaction Data – Security Signals and Metrics Definitions

Our system computes reputation from an actor's historical interactions in the ecosystem by operationalizing them as seven security signals (S1–S7), each derived from one or more measurable metrics (Table 2). We compute each signal score by aggregating its associated metrics (e.g., S1 aggregates three metrics).

To define appropriate security signals and metrics, we consider: (1) alignment with widely accepted security recommendations, (2) the actor's demonstrated adherence to good security practices in previous contributions and to significant projects, and (3) a history of non-malicious contributions.

From these considerations, we derive our security metrics from two kinds of sources:

(1) **Security standards and recommendations:** We consulted frameworks like the SLSA security framework [81], the CNCF software supply chain security guidelines [17], NIST SSDF [74], NIST SP 800-204D [14], OpenSSF S2C2F [60], and the CIS Software Supply Chain Security Guide [13]. We use common recommendations across these sources to prioritize well-established practices.
(2) **Available security tools in the OSS ecosystem:** We identified security tools available through GitHub's user interface and API, which reflect the security capabilities easily available to contributors on the platform. ARMS does not propose new vulnerability detection or scanning techniques; it consumes best-available ecosystem outputs (e.g., alerts, advisories, and scanner results) and treats them as noisy evidence for decision support rather than ground truth.

Scope and attribution. To avoid conflating actor behavior with project governance, we distinguish actor-behavior signals from repository-governance context and ecosystem context, as described in Table 2. In addition to the seven security signals, our model includes **three weighting (impact) signals** (W1–W3) that calibrate risk exposure and uncertainty (package usage, community tenure, and centrality). These weighting signals are applied in the reputation computation (§4.3) to calibrate exposure and uncertainty, rather than being treated as additional security signals.

4.3 Trust Engine – Reputation Computation

In this section, we describe candidate approaches for computing reputation from the signals in Table 2 (see Figure 1). We focus on how interaction histories can be translated into per-signal evidence and a composite score that supports maintainer triage. Because multiple design choices are plausible, we present one concrete instantiation for clarity and identify alternatives that we plan to compare empirically (§6.1).

First, the interaction data formatter extracts each contributor's ecosystem interaction history and computes metric values for all proposed metrics in Table 2. Next, the security signal scorer aggregates those metric values into **seven security signal scores** (S1–S7), where each signal score is computed from its associated metrics (e.g., Signal 1 aggregates three proposed metrics; other signals aggregate one or more metrics).

Signal scoring. For each security signal $s \in \{S1, \ldots, S7\}$, one candidate approach is to compute a normalized score $\text{Score}_s(a) \in [0, 1]$ for actor a over a time window, optionally using time decay to down-weight older interactions. When a metric reflects repository configuration rather than an actor's direct actions, we treat it as governance context unless control can be attributed to the actor (e.g., the actor is an owner/maintainer or the configuration change is attributable to the actor). Other alternative scoring functions are plausible (e.g., robust aggregation such as medians, or cohort-normalized scoring).

Composite scoring. A simple baseline is to aggregate metric evidence across contribution events $e \in \mathcal{E}(a)$ into each signal score, calibrating each event by package-usage exposure (W1):

$$\text{Score}_{S_s}(a) = \text{Norm}\left(\sum_{e \in \mathcal{E}(a)} \text{W1}(e) \cdot f_s(e) \right),$$

where $f_s(e)$ extracts the event-level contribution to signal s (computed from the proposed metrics in Table 2), and $\text{Norm}(\cdot)$ maps scores to $[0, 1]$. Other aggregation strategies may be preferable in practice, including time-decayed sums, winsorized/trimmed statistics, or per-metric normalization before pooling; we treat these as design choices to be evaluated.

Given per-signal scores, one candidate composite is a weighted linear combination:

$$R_0(a) = \sum_{s=1}^{7} \alpha_s \cdot \text{Score}_{S_s}(a).$$

Combining multiple signals can reduce reliance on any single noisy indicator and mitigate false positives and false negatives, similar in spirit to composite security assessments such as OpenSSF Scorecard [94]. We treat the selection of α_s as an empirical question: one approach is to set α_s via effectiveness analysis (§6.1), but alternatives include uniform weights, expert-set weights, or learned weights from historical outcomes.

Terminology. We use α_s to denote *signal importance weights* that determine how strongly each of the seven security signals contributes to the composite score. We reserve W1–W3 for *calibration factors*: W1 is an exposure factor applied to event-level evidence, while W2 and W3 calibrate the composite score for uncertainty (tenure) and ecosystem context (centrality), as described next.

Impact Score (calibration factors W1–W3 in Table 2): We plan to apply calibration factors at two stages.

(1) Per-contribution exposure calibration (W1): We incorporate package usage (W1) when aggregating event-level evidence into each security signal score. Interactions tied to projects with minimal downstream usage contribute less to signal evidence because they present lower supply chain exposure. Concretely, we calibrate contribution events by a bounded usage factor and then aggregate to compute each $\text{Score}_{S_s}(a)$.

(2) Composite-level calibration (W2–W3): After computing the seven security signal scores, another approach is to calibrate the composite reputation using community tenure (W2) and centrality (W3). Tenure can moderate overconfidence from sparse histories, while centrality can provide ecosystem context based on connectedness and the time taken to establish it. One candidate formulation is:

$$R_1(a) = R_0(a) \cdot \text{Calibrate}(\text{W2}(a), \text{W3}(a)),$$

where $R_0(a) = \sum_{s=1}^{7} \alpha_s \cdot \text{Score}_{S_s}(a)$. Alternative calibration strategies include separating a score from an explicit confidence estimate, using additive rather than multiplicative adjustments, or applying cohort-based calibration.

Benchmark Score: Finally, we consider contextualizing $R_1(a)$ relative to ecosystem-wide behavior to support interpretable decisions. One approach is to compute a benchmarked reputation using either an ecosystem percentile or a standardized score relative to the ecosystem median/mean:

$$R(a) = \text{Benchmark}(R_1(a)).$$

We treat benchmarking choice as another design decision within the midst of other alternatives; percentile-, standardized-, and cohort-based baselines (§6.1).

4.4 Actionability Output: Using ARMS in Maintainer Workflows

ARMS is a decision-support tool that guides triage and review efforts rather than automatically accepting or rejecting changes. For sparse histories, ARMS recommends additional verification (e.g., maintainer review, smaller scoped PRs, CI pass) rather than rejection. For low scores in high-exposure contexts, ARMS recommends stronger safeguards (e.g., two-person review, restrict to non-critical modules, require signing where feasible). For high scores, ARMS recommends faster routing while preserving normal repository policies. ARMS should also surface brief explanations (top contributing signals) and treat missing signals as unknown rather than negative evidence.

4.5 Worked Examples

We illustrate ARMS with the motivating examples from §3.2.

4.5.1 XZ Utils Attack. The timeline of events leading to the XZ Utils backdoor reveals several characteristics that map to our proposed security signals [72]:

(1) **Recent account**: The attacker created a GitHub account in January 2021 and joined the XZ Utils project in October 2021—well within their first year of activity.
(2) **Limited public history**: Prior to October 2021, their contributions were confined to private repositories.
(3) **Targeted feature pull requests**: Their first public pull request focused on adding features to a small set of projects rather than fixing issues.

Under our framework, these traits would yield a low reputation:

- Signals S1–S7 yield limited positive evidence due to sparse or opaque public contribution history.
- *Community tenure* (Signal W2) reduces scores for newly created accounts.
- *Centrality* (Signal W3) remains low because the user's contribution network is both recent and shallow.

4.5.2 Dexcom. The Dexcom engineers' case revealed repeated failures that lasted for extended periods. In one incident, an issue persisted from November 28 to December 3, 2019, halting Dexcom

systems and severely impacting availability. Under our framework, this would be reflected in lower scores for Signals 1 and 2 (**time to fix vulnerabilities and severity levels**), as recurrent downtime directly reduces the reputation of engineers responsible for critical systems.

4.5.3 ESLint Compromise. The 2018 ESLint compromise was traced to an attacker breaching a maintainer's account—enabled by the absence of two-factor authentication. This type of attack cannot be modeled by reputation systems because the attacker took over a legitimate account without performing any genuine contributor actions or exhibiting the behavioral signals that these systems track. This is why we deem it out of scope for our threat model §3.

5 Agent Identity and Attribution

Any agent-aware deployment of ARMS first requires that the system identify which agent, model family, model version, or operator produced a contribution. An agent reputation score is meaningful only if the underlying actor can be linked reliably across contributions. For human contributors, identity is already imperfect. For agents, the problem is more complex because a single human operator may deploy multiple agents, a single agent may be wrapped by multiple interfaces, and models may drift across versions, fine-tuning steps, or rebranding [48, 53].

This identity layer should distinguish at least four entities: the model family, the deployed version, the surrounding agent or wrapper, and the human or organizational operator. Conflating these can yield misleading reputation assignments, such as blaming a trustworthy operator for a risky third-party wrapper or treating multiple versions of a rapidly changing agent as a single stable trustee.

Several technical directions are relevant to this problem. Supply-chain provenance can record signed attestations about model origin and deployment lineage [48]. Black-box fingerprinting techniques can help distinguish model families and versions [62, 83]. Ownership and watermark-style approaches can support claims about model origin or copying [15, 67]. Lineage and provenance testing can help reason about derived or fine-tuned models [53, 80].

In our setting, these mechanisms should support versioning, revocation, and separation between agent reputation and operator reputation. They also help prevent a malicious or low-quality agent from repeatedly reappearing under slightly changed names. The conceptual point is simple: ARMS for agents requires a stable identity layer before any reputation computation can be meaningful.

6 Design of Experiments for ARMS

In this section, we outline studies to evaluate the feasibility of an actor reputation system, exemplified by ARMS, and, more broadly, the use of actor metrics to establish trust in software supply chain security. These complementary studies—primarily observational or retrospective—aim to validate and refine the proposed security-reputation metrics and assess the real-world impacts of deploying ARMS. Assuming the identity layer discussed in §5, we organize the evaluation agenda into two groups: (1) evaluating the effectiveness of the proposed security metrics, and (2) examining user behaviors.

For a first journal submission (TSE), we will prioritize an ARMS prototype and quantitative feasibility plus retrospective sanity checks of the proposed signals (scalability, behavior on well-established maintainers, and known-failure catalogs). We defer user behavioral studies (vignettes, surveys, chilling-effect analysis) to follow-up work after quantitative validation.

6.1 Security Metrics Effectiveness Evaluation – Quasi-Experimental Analyses

To evaluate the utility of the proposed security signals, we propose quasi-experimental studies that test whether these signals (and alternative scoring choices from §4.3) predict downstream security outcomes and observable security-relevant behaviors.

(1) **Security Practice Adoption Study (Difference-in-Differences):** Compare OSS projects that adopted a given security practice (*e.g.,* branch protection, private vulnerability reporting) to matched control projects that did not, using a difference-in-differences design. We will analyze the results with regression models incorporating project and time fixed effects, and (when appropriate) interaction terms for paired practices to isolate combined impacts on security outcomes.
   - *Data*: Historical GitHub records for projects within the same domain and with similar size and activity levels.
   - *Outcomes*: Changes in vulnerability incidence and remediation behavior (Signals S1–S2) and shifts in security-relevant workflows (Signals S3–S7).
   - *Ethics*: Uses only publicly available data.

(2) **Retrospective Incident Prediction Study:** Identify projects that experienced known supply-chain incidents (*e.g.,* npm/event-stream, ua-parser-js) and compare their pre-incident signal profiles against matched projects without known incidents.
   - *Data*: Metric values for each project and its maintainers in a fixed pre-incident window.
   - *Analysis*: Logistic regression (including interaction terms where justified) to estimate which signals and combinations are most predictive of incident occurrence.
   - *Outputs*: Predictive utility summaries such as AUC, calibration curves, and error rates under different thresholds.

(3) **Ablation and Sensitivity Analyses over the Trust Engine Design:** Because multiple reputation-computation choices are plausible (§4.3), we will ablate key design decisions and measure how conclusions change under alternative formulations.
   - *Signal aggregation*: sum vs. median/robust aggregation; with and without time decay.
   - *Composite function*: linear combination vs. non-linear pooling and thresholded rules that require minimum evidence.
   - *Weight selection*: uniform weights vs. expert-set weights vs. weights derived from effectiveness analysis.
   - *Calibration and benchmarking*: multiplicative vs. additive calibration; percentile vs. standardized vs. cohort-based benchmarks.
   - *Evaluation*: compare predictive performance, stability of rankings, and sensitivity for newcomers (sparse histories) across variants.

6.2 User Behavioural Studies – Surveys & Interviews

To evaluate the practical usability of actor metrics in OSS Supply chain security, we propose the following studies:

(1) **Collaborator Vetting Study:** We propose recruiting active OSS maintainers to participate in a vignette study [1]. This would present anonymized human and agent contributor profiles with varying metric scores and attribution evidence, and measure time-to-decision (vet vs. reject) and how variations on the proposed ARMS signals would influence these choices. This method would gather qualitative feedback on signal clarity and usefulness.
(2) **Chilling Effect:** A potential issue with establishing an actor reputation system is the possibility of a "chilling effect,"[46] where contributors may hesitate or reduce participation if they know their interactions are being tracked and scored [5, 11, 46]. We propose a user survey study to assess contributors' willingness to participate under different tracking scenarios (*e.g.,* "all projects" vs. "critical only"), and to quantify perceived privacy risks and impact of monitoring activities on contribution intent.

    Ultimately, we do expect some chilling effects. To mitigate them, we recommend limiting the deployment of an ARMS approach to high-impact, security-sensitive OSS projects (*e.g.,* the Linux kernel) rather than applying it to hobby or low-risk repositories. To do this, there is a need to develop an effective project importance score. OpenSSF's criticality score [3] may be used or serve as a starting point. We recommend surveying OSS contributors to gauge a suitable threshold of criticality.

7 Discussions and Future Works

7.1 Threats to Validity

We begin by outlining some potential issues with our proposal:

(1) ***False Positives and Negatives:*** There is a risk of mis-assigning reputation, resulting in inappropriate characterizations of users as more or less trustworthy. Our definitions and metrics do not account for all interactions that could detect incompetence or malicious activities, especially non-artifact interactions such as social engagements, organizations, security communications, feedback to code reviews, etc. To mitigate this, we recommend a weighted combination of metrics, with each metric's effectiveness considered in §6.1. Using ecosystem-wide averages and standard deviations can also help raise the threshold for issuing advisories based on user scores.
(2) ***Insider Threats:*** As a special case of false negatives, we emphasize that any reputational system is ineffective for insider threats where trust is deliberately built over time. However, such attacks are costly for an adversary.
(3) ***Defining Security Interactions:*** As stated, our work envisions operationalizing actor trust security metrics; however, we acknowledge the inadequacies of our proposed security metrics. We encourage further research in defining trust, especially concerning the interactions between security metrics.
(4) ***Privacy Concerns:*** Making scores public could unfairly harm honest actors. To address this, we propose that scores remain private, accessible on a need-to-know basis. Advisories based on these scores would be shared only with maintainers of projects to which the user wishes to contribute.
(5) ***Gameability of Proposed Metrics:*** Our security metrics, like any trust-based system, can be exploited. Users may act, individually or in collusion, to enhance their reputation. We mitigate this risk with time-weighted scoring and recommend incorporating human oversight (*e.g.,* reporters and moderators) for nuanced judgments. However, we acknowledge that trust and safety issues plague all online platforms [19], and a reputation system would be no exception.
(6) ***Possibility of Chilling Effect:*** Discussed in §6.2.

7.2 Evaluating Actor Intent

Our work assumes that actors are well-intentioned and attributes security failures to genuine contributors' lack of expertise or mistakes; it does not account for malicious intent. However, some supply-chain incidents arise from malicious actors using techniques such as account spoofing or credential theft. Incorporating intent into reputation systems substantially increases their complexity. Although our current security signals focus on expertise and behavior, they do not capture actor intent. Future work should extend these metrics to infer intent, *e.g.,* by analyzing anomalous contribution patterns, unusual social graph connections, or timing irregularities, ultimately creating a more comprehensive reputation model.

7.3 Supporting Ecosystem Heterogeneity

Reputation systems fundamentally face actor-identification challenges [47, 78]. This issue is especially pronounced in open-source ecosystems, where contributors manage artifacts across multiple platforms, from source repositories to package registries. Although next-generation software signing tools have improved artifact-to-actor verification [40, 52, 68], reliable cross-platform identification remains vulnerable to identity theft, impersonation, and other attacks. Consequently, ARMS requires stronger mechanisms to verify and vouch for identities across diverse environments.

To provide possible solutions, recent initiatives (*e.g.,* CISA's RFI for software identifier ecosystems [13]) propose establishing identities through multiple institutions, some designed to preserve privacy [50, 86]. Adapting these models across ecosystems introduces the dual challenges of federation, enabling actors in one ecosystem to be recognized in another, and roaming, allowing actors to transfer their identifier and reputation between providers. Federation and Roaming are challenges reminiscent to other federation protocols work (*e.g.,* OIDC [58] and Distributed Identities [8]). Thus, for an actor reputation system like ARMS, institutions across ecosystems must collaborate and share reputation information whenever a software artifact from one ecosystem is used in another ecosystem.

7.4 Additional Considerations for Agent Reputation

Earlier sections argued that agents should be treated as first-class actors within ARMS rather than as a separate future-work topic. Here, we highlight the additional requirements that become especially important once agent reputation is operationalized. Agent reputation differs from the human scenario because agents can evolve rapidly, execute autonomously, and be invoked as dependencies

by other agents [37, 63, 89, 92]. Consequently, reputation must be strongly versioned, separate the agent from its operator or publisher when possible, incorporate lifecycle and revocation signals, and account for feedback loops in agent-to-agent selection. Emerging agent-sharing infrastructures suggest additional observable inputs, such as signed manifests, provenance and evidence pointers, and lifecycle metadata (e.g., active, deprecated, revoked), that a trust engine could use when comparing candidate agents [37, 63, 92].

7.5 Social & Process Signals: Beyond Artifact Contributions

Our current security signals (S1–S7) and calibration factors (W1–W3) (Table 2) focus exclusively on an actor's artifact contributions. As noted in our approach criticisms, we have not yet captured an actor's interactions with other users, such as code-review feedback, issue discussions, and peer endorsements, which could provide valuable reputational insights.

We also recommend developing *process-based* metrics to evaluate informal OSS practices. Process-based methods [16] are most effective in structured development environments. However, in open-source ecosystems, where formal processes are often absent [75], proxy process metrics can still gauge adherence to security best practices. For example, one could measure the frequency of explicit threat-model or design-review discussions in issues or pull requests, track the presence and pass rate of automated security scans (e.g., SAST, SBOM generation) in continuous integration workflows, and monitor the regularity of dependency updates following published vulnerability disclosures. By combining artifact, social, and process signals, ARMS can deliver a more holistic reputation assessment for contributors in open-source ecosystems.

7.6 Ethical and Privacy Considerations

A system that records actor activities inevitably raises ethical and legal concerns. Actors must provide informed consent before their ecosystem activities are disclosed to projects they wish to join. At the same time, project owners need actionable insights into a contributor's security posture. Balancing these interests requires privacy-preserving disclosure.

We propose that (i) contributors opt in to sharing reputational data, (ii) only aggregated or pseudonymized metrics (*e.g.,* differential-privacy noise or percentile rankings) are revealed to maintainers, and (iii) contributors receive dashboards that show exactly what information will be shared. Compliance with regulations such as GDPR [26] should guide data-retention periods, user-deletion requests, and audit logging. Future prototypes should incorporate cryptographic approaches—such as zero-knowledge proofs or secure multiparty computation—to prove adherence to security metrics without exposing raw activity logs. Key directions include: Designing and evaluating privacy-preserving reputation protocols, conducting user studies to gauge contributor consent thresholds and maintainer information needs, and integrating regulatory compliance checks and automated audit trails into the prototype.

7.7 Why Focus Reputation on Cybersecurity?

We have situated the ARMS approach within the context of cybersecurity. The ARMS metrics can, of course, be extended in order to infer properties of engineers other than their cybersecurity expertise. We suggest that doing so for functional properties (*i.e.,* input-output behaviors that can be validated through testing) may be unnecessary — standard engineering practices call for code contributions to be accompanied by adequate tests, such that functional properties can be assured through reference to the test results. Not so for non-functional properties, cybersecurity as one of many. It is for such properties that reputational measures may be more useful. For example, Cramer *et al.* recently reported that trust and safety defect repairs in social media platforms are rarely accompanied by automated tests, partly since validation is apparently performed through use case analysis rather than through software behavioral analysis [19]. Similarly, behaviors for regulatory compliance, such as GDPR are challenging to validate [28]. Since the current state-of-the-art software validation techniques cannot automatically conclude whether a system provides non-functional properties such as security, trust-and-safety, or regulatory compliance, at least not in a cost-effective manner, we think that ARMS approaches may be a suitable complementary measure of correctness.

8 Conclusion

In this paper, we explored the characteristics of the open-source software supply chain that necessitate actor-based security techniques. We argued that this need is becoming more urgent as maintainers increasingly evaluate both human contributors and autonomous coding agents. We advocate for the implementation of an actor reputation system to operationalize a framework for trust within the open-source ecosystem. We introduced seven security signals (each operationalized by concrete metrics) for estimating actor reputation, and outlined quantitative study designs to assess signal utility, agent identification, and end-to-end system feasibility. Lastly, we identified future research directions, including stronger provenance and attribution mechanisms for agents, a system prototype, quantitative validation at scale, and follow-up user studies on decision-making and deployment impacts.

Acknowledgments

We acknowledge support from the National Science Foundation (NSF) (Award No. 2537308).

Table 2: Proposed security signals, weighting factors, and evaluation metrics. Bolded security and weighting signals are derived from established frameworks (e.g., NIST, SLSA, CNCF) and prior work, respectively. The proposed metrics describe how to measure each highlighted security or weighting signal. Relevant case studies and related research are cited to justify inclusion of select metrics. Note: S3–S7 reflect repository configuration and are treated as context unless governance control is attributable to the actor (e.g., owner/maintainer).

S/N.	Proposed Signals & Analysis Metrics	Description
	SECURITY SIGNALS	
	Actor behavior security signals (S1–S2)	
1.	**Security vulnerability in artifact: Pull Requests/Commits**	Measures vulnerabilities introduced through pull requests or commits, either by the user or in projects owned by the user.
a.	Time to fix/close security vulnerabilities of various severity levels [7]	Time taken to address vulnerabilities detected in pull requests or commits.
b.	Severity levels of security vulnerability reported [71]	Evaluates the criticality of reported vulnerabilities (low, medium, high) for the user's projects.
c.	Presence of vulnerability in PR/commits (not linked to external dependencies) [30]	Measures vulnerabilities that stem from the user's direct contributions.
2.	**Security vulnerability in artifact: use of vulnerable dependencies**	Assesses the use and introduction of vulnerable dependencies into the artifact.
a.	Length of time before fix [7]	Time taken to resolve vulnerable dependencies after detection (or public disclosure).
b.	Severity levels of Repos/Project's reported vulnerability [71]	Tracks severity of vulnerabilities in dependencies.
c.	Number of Repos/Projects that have a vulnerable dependency	Total number of projects affected by vulnerable dependencies as a percentage of the user's total projects.
d.	Number of vulnerable dependencies per project	Assesses how widespread vulnerable dependencies are within each project.
e.	Number of projects with reported vulnerable dependencies	Measures the overall exposure of the user's projects to vulnerable dependencies.
	Repository governance security signals (S3–S7)	
3.	**Use of ecosystem code scanning and security analysis features**	Evaluates usage of the ecosystem's security tools for code scanning.
a.	Status of dependabot alerts (for dependencies) [27]	Monitors whether security alerts for dependencies are being addressed.
b.	Status of Secret sharing/Validity Checks	Evaluates management of secret scanning and validity checks.
c.	Code scanning and Vulnerability alerts [27]	Assesses whether security scanning tools are actively used and vulnerability alerts managed.
d.	Push Protection Status	Checks whether protection features are used to block pushes with exposed secrets.
4.	**Use of ecosystem integrity guarantees**	Examines use of ecosystem integrity features (e.g., code signing).
a.	Number of Projects with an integrity guarantee [69]	Measures how many projects are secured with integrity verification features such as code signing.
5.	**Use of branch protection**	Checks whether branch protection is enforced to prevent unauthorized changes.
a.	Number of protected branches per project	Evaluates how many branches within each project are guarded against direct commits.
b.	Number of Projects with Protected branches [59, 94]	Tracks how many projects enforce branch protection.
6.	**Use of security policies and vulnerability reporting**	Determines whether the project has security policies and reporting mechanisms.
a.	Status of private vulnerability reporting, or security policy [59]	Checks whether private vulnerability reporting channels and security policies are established and functional.
7.	**Use of automated workflows**	Assesses use of automation in workflows to enforce security.
a.	Number of Projects with Automated workflows [59]	Measures how many user projects use automated workflows for security enforcement (e.g., required checks, automated tests).
	CALIBRATION FACTORS	
W1.	**Package Usage [9]**	
a.	Number of downloads/stars/forks	Calibrates (Weights) project-level evidence by downstream exposure.
W2.	**Community Tenure [7]**	
a.	Length of time contributing to project	Calibrates uncertainty based on time contributing to other projects.
b.	Length of time owning account	Calibrates uncertainty based on account age.
c.	Contributory Strength	Calibrates evidence strength based on contribution volume (e.g., commits, issues, pull requests).
W3.	**Centrality Score [32]**	
a.	Connections to other Actors [32]	Calibrates ecosystem context based on a contributor's network of interactions.
b.	Time to form connections	Calibrates based on how long it took to establish the connections in (a).

References

[1] Herman Aguinis and Kyle J Bradley. 2014. Best practice recommendations for designing and implementing experimental vignette methodology studies. *Organizational research methods* 17, 4 (2014).

[2] Sabrina Amft, Sandra Höltervennhoff, Rebecca Panskus, Karola Marky, and Sascha Fahl. 2024. Everyone for Themselves? A Qualitative Study about Individual Security Setups of Open Source Software Contributors. In *IEEE Symposium on Security and Privacy (SP)*.

[3] Abhishek Arya, Caleb Brown, Rob Pike, and The Open Source Security Foundation. 2023. Open Source Project Criticality Score. https://github.com/ossf/criticality_score.

[4] Sebastian Benthall. 2017. Assessing software supply chain risk using public data. In *2017 IEEE 28th Annual Software Technology Conference (STC)*. 1–5. doi:10.1109/STC.2017.8234461

[5] Thomas Bernauer and Marcus M Dapp. 2009. Hot Debate About Chilling Effects: Do Software Patterns Hamper/Free Open Source Software Development? (2009).

[6] Hudson Borges and Marco Tulio Valente. 2018. What's in a GitHub star? Understanding repository starring practices in a social coding platform. *Journal of Systems and Software* (2018).

[7] Lina Boughton, Courtney Miller, Yasemin Acar, Dominik Wermke, and Christian Kästner. 2024. Decomposing and Measuring Trust in Open-Source Software Supply Chains. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. 57–61. doi:10.1145/3639476.3639775

[8] Tim Bouma. 2024. High Assurance DIDs with DNS. https://www.ietf.org/archive/id/draft-carter-high-assurance-dids-with-dns-03.html. Internet-Draft. Accessed: 2026-01-26.

[9] Sam Boysel. 2023. No Free Lunch For Programmers: Digital Supply Chains and the Economics of Software Dependency Management. (2023).

[10] Larissa Braz and Alberto Bacchelli. [n. d.]. Software security during modern code review: the developer's perspective. In *2022 European Software Engineering Conference & Symposium on the Foundations of Software Engineering*.

[11] Moritz Büchi, Noemi Festic, and Michael Latzer. 2022. The chilling effects of digital dataveillance: A theoretical model and an empirical research agenda. *Big Data & Society* 9, 1 (2022), 20539517211065368.

[12] Kim Cameron. 2005. The Laws of Identity. https://www.identityblog.com/?p=352. Accessed: 2026-01-26.

[13] Center for Internet Security. 2021. *CIS Software Supply Chain Security Guide*. Technical Report. Center for Internet Security. https://www.cisecurity.org/white-papers/cis-software-supply-chain-security-guide Accessed: 2025-05-20.

[14] Ramaswamy Chandramouli, Ramaswamy Chandramouli, Frederick Kautz, and Santiago Torres-Arias. 2024. *Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD Pipelines*. US Department of Commerce, National Institute of Standards and Technology.

[15] Jialuo Chen, Jingyi Wang, Tinglan Peng, Youcheng Sun, Peng Cheng, Shouling Ji, Xingjun Ma, Bo Li, and Dawn Song. 2022. Copy, Right? A Testing Framework for Copyright Protection of Deep Learning Models. In *2022 IEEE Symposium on Security and Privacy (SP)*. 824–841. doi:10.1109/SP46214.2022.9833747

[16] Charles Clancy, Joseph Ferraro, Robert Martin, Adam Pennington, Christopher Sledjeski, and Craig Wiener. 2021. Deliver uncompromised: Securing critical software supply chains. *MITRE Technical Papers* 24, 01 (2021).

[17] Cloud Native Computing Foundation. 2021. CNCF paper defines best practices for supply chain security. https://www.cncf.io/announcements/2021/05/14/cncf-paper-defines-best-practices-for-supply-chain-security/. Accessed: 2026-01-26.

[18] Codecov Security Team. 2021. *Bash Uploader Security Update*. https://tinyurl.com/5d8bd4je Accessed: 2025-05-15.

[19] Geoffrey Cramer, William P Maxam III, and James C Davis. 2025. Engineering patterns for Trust and Safety on social media platforms: A case study of Mastodon and Diaspora. *Journal of Systems and Software* (2025).

[20] Cycode Security Team. 2022. *ESLint: Compromising the Build Using a Supply Chain Attack*. https://cycode.com/blog/eslint-compromising-the-build-using-supply-chain-attack/ Accessed: 2025-05-15.

[21] Thiago Melo Stuckert do Amaral and João José Costa Gondim. 2021. Integrating Zero Trust in the cyber supply chain security.

[22] John R Douceur. 2002. The sybil attack. In *International workshop on peer-to-peer systems*. Springer, 251–260.

[23] F. Betül Durak, Kim Laine, Simon Langowski, Radames Cruz Moreno, Robert Sim, and Shrey Jain. 2023. Sandi: A System for Accountability and Applications in Direct Communication (Extended Abstract). arXiv:2311.04861.

[24] Sarah Elder, Nusrat Zahan, Rui Shu, Monica Metro, Valeri Kozarev, Tim Menzies, and Laurie Williams. 2022. Do I really need all this work to find vulnerabilities? An empirical case study comparing vulnerability detection techniques on a Java application. *Empirical Software Engineering (EMSE)* (2022).

[25] ESLint Team. 2018. *Postmortem for Malicious Packages Published on July 12th, 2018*. https://eslint.org/blog/2018/07/postmortem-for-malicious-package-publishes/ Accessed: 2025-05-15.

[26] European Union. 2016. *General Data Protection Regulation (GDPR)*. https://gdpr-info.eu/ Regulation (EU) 2016/679. Accessed: 2025-04-30.

[27] Felix Fischer, Jonas Höbenreich, and Jens Grossklags. 2023. The Effectiveness of Security Interventions on GitHub. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*.

[28] Lucas Franke, Huayu Liang, Sahar Farzanehpour, Aaron Brantly, James C Davis, and Chris Brown. 2024. An exploratory mixed-methods study on general data protection regulation (gdpr) compliance in open-source software. In *International Symposium on Empirical Software Engineering and Measurement*.

[29] GitHub Docs. [n. d.]. Saving repositories with stars. https://docs.github.com/en/get-started/exploring-projects-on-github/saving-repositories-with-stars. Accessed: 2026-01-26.

[30] Danielle Gonzalez, Thomas Zimmermann, Patrice Godefroid, and Max Schaefer. 2021. Anomalicious: Automated Detection of Anomalous and Potentially Malicious Commits on GitHub. In *International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*.

[31] Michael T. Goodrich and Florian Kerschbaum. 2011. Privacy-Enhanced Reputation-Feedback Methods to Reduce Feedback Extortion in Online Auctions. In *Conference on Data and Application Security and Privacy*.

[32] Sivana Hamer, Nasif Imtiaz, Mahzabin Tamanna, Preya Shabrina, and Laurie Williams. 2025. Trusting Code in the Wild: Exploring Contributor Reputation Measures to Review Dependencies in the Rust Ecosystem. *IEEE Transactions on Software Engineering* 51, 4 (2025), 1319–1333. doi:10.1109/TSE.2025.3551664

[33] Mark Harman and Peter O'Hearn. 2018. From start-ups to scale-ups: Opportunities and open problems for static and dynamic program analysis. In *international working conference on source code analysis and manipulation (SCAM)*. IEEE.

[34] Ferry Hendrikx, Kris Bubendorfer, and Ryan Chard. 2015. Reputation systems: A survey and taxonomy. *J. Parallel and Distrib. Comput.* 75 (2015), 184–197. doi:10.1016/j.jpdc.2014.08.004

[35] Raphael Hiesgen, Marcin Nawrocki, Thomas C Schmidt, and Matthias Wahlisch. [n. d.]. The Race to the Vulnerable: Measuring the Log4j Shell Incident. ([n. d.]). Preprint or manuscript. Accessed: 2026-01-26.

[36] Michael A Howard. 2006. A process for performing security code reviews. *IEEE Security & privacy* 4, 4 (2006), 74–79.

[37] Chengyou Jia, Minnan Luo, Zhuohang Dang, Qiushi Sun, Fangzhi Xu, Junlin Hu, Tianbao Xie, and Zhiyong Wu. 2025. Agentstore: Scalable integration of heterogeneous agents as specialized generalist computer assistant. In *Findings of the Association for Computational Linguistics: ACL 2025*. 8908–8934.

[38] Audun Josang, Roslan Ismail, and Colin Boyd. 2007. A survey of trust and reputation systems for online service provision. *Decision Support Systems* (2007).

[39] Kelechi Kalu, Taylor R. Schorlemmer, Sophie Chen, Kyle A. Robinson, Erik Kocinare, and James C. Davis. 2023. Reflecting on the Use of the Policy-Process-Product Theory in Empirical Software Engineering. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 2112–2116. doi:10.1145/3611643.3613075

[40] Kelechi G Kalu, Tanya Singla, Chinenye Okafor, Santiago Torres-Arias, and James C Davis. 2025. An Industry Interview Study of Software Signing for Supply Chain Security. In *34th USENIX Security Symposium (USENIX Security 25)*. USENIX Association, Seattle, WA, USA.

[41] Oleksii Kononenko, Olga Baysal, and Michael W. Godfrey. 2016. Code review quality: how developers see it. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*.

[42] Arjun Krishna, Erick Galinkin, Leon Derczynski, and Jeffrey Martin. 2025. Importing Phantoms: Measuring LLM Package Hallucination Vulnerabilities. *CoRR* abs/2501.19012 (2025). doi:10.48550/arXiv.2501.19012

[43] Hao Li, Haoxiang Zhang, and Ahmed E. Hassan. 2025. The Rise of AI Teammates in Software Engineering (SE) 3.0: How Autonomous Coding Agents Are Reshaping Software Engineering. *CoRR* abs/2507.15003 (2025). doi:10.48550/arXiv.2507.15003

[44] Mario Lins, René Mayrhofer, Michael Roland, Daniel Hofer, and Martin Schwaighofer. 2024. On the critical path to implant backdoors and the effectiveness of potential mitigation techniques: Early learnings from XZ. *arXiv preprint arXiv:2404.08987* (2024).

[45] Danaja Maldeniya, Ceren Budak, Lionel P Robert Jr, and Daniel M Romero. 2020. Herding a deluge of good samaritans: How GitHub projects respond to increased attention. In *The Web Conference (WWW)*.

[46] Ben Marder, Adam Joinson, Avi Shankar, and David Houghton. 2016. The extended 'chilling'effect of Facebook: The cold reality of ubiquitous social networking. *Computers in Human Behavior* 60 (2016), 582–592.

[47] S. Marti and H. Garcia-Molina. 2003. Identity crisis: anonymity vs reputation in P2P systems. In *International Conference on Peer-to-Peer Computing (P2P2003)*. doi:10.1109/PTP.2003.1231513

[48] Sarah Meiklejohn, Hayden Blauzvern, Mihai Maruseac, Spencer Schrock, Laurent Simon, and Ilia Shumailov. 2025. Machine Learning Models Have a Supply Chain Problem. *CoRR* abs/2505.22778 (2025). doi:10.48550/arXiv.2505.22778

[49] Marcela S Melara and Mic Bowman. 2021. Hardware-Enforced Integrity and Provenance for Distributed Code Deployments. arXiv:2106.09843.

[50] Microsoft. n.d.. *Microsoft Entra ID.* https://www.microsoft.com/en-us/security/business/identity-access/microsoft-entra-id Accessed: 2025-05-20.

[51] Sy Mukherjee. 2019. Dexcom Software Outage Draws Fury from Diabetes Patients' Parents. https://fortune.com/2019/12/02/dexcom-outage-blackout-diabetes-patients-blood-sugar-monitor/. Accessed: 2026-01-26.

[52] Zachary Newman, John Speed Meyers, and Santiago Torres-Arias. 2022. Sigstore: Software signing for everybody. In *ACM Conference on Computer and Communications Security (CCS).* Check the exact CCS year/track formatting used by your bibliography style.

[53] Ivica Nikolić, Teodora Baluta, and Prateek Saxena. 2025. Model Provenance Testing for Large Language Models. *CoRR* abs/2502.00706 (2025). doi:10.48550/arXiv.2502.00706

[54] Donald A. Norman. 2013. *The Design of Everyday Things* (revised and expanded ed.). MIT Press, Cambridge, MA.

[55] Anahad O'Connor. 2019. In Weekend Outage, Diabetes Monitors Fail to Send Crucial Alerts. https://www.nytimes.com/2019/12/02/well/live/Dexcom-G6-diabetes-monitor-outage.html. Accessed: 2026-01-26.

[56] Marc Ohm, Henrik Plate, Arnold Sykosch, and Michael Meier. 2020. Backstabber's knife collection: A review of open source software supply chain attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment.*

[57] Chinenye Okafor, Taylor R Schorlemmer, Santiago Torres-Arias, and James C Davis. 2022. SoK: Analysis of software supply chain security by establishing secure design properties. In *ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses.*

[58] OpenID Foundation. [n. d.]. How OpenID Connect Works. https://openid.net/developers/how-connect-works/. Accessed: 2026-01-26.

[59] OpenSSF. [n. d.]. OpenSSF Scorecard. https://scorecard.dev/. Accessed: 2026-01-26.

[60] OpenSSF. n.d.. *s2c2f.* https://github.com/ossf/s2c2f Accessed: 2025-05-20.

[61] Shengyi Pan, Lingfeng Bao, Jiayuan Zhou, Xing Hu, Xin Xia, and Shanping Li. 2024. Towards More Practical Automation of Vulnerability Assessment. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE).* doi:10.1145/3597503.3639110

[62] Dario Pasquini, Evgenios M. Kornaropoulos, and Giuseppe Ateniese. 2025. LLMmap: Fingerprinting for Large Language Models. In *34th USENIX Security Symposium (USENIX Security 25).* https://www.usenix.org/conference/usenixsecurity25/presentation/pasquini

[63] Erik Pautsch, Tanmay Singla, Wenxin Jiang, Huiyun Peng, Behnaz Hassanshahi, Konstantin Läufer, George K. Thiruvathukal, and James C. Davis. 2025. AgentHub: A Research Agenda for Agent Sharing Infrastructure. arXiv:2510.03495 (Oct. 2025). doi:10.48550/arXiv.2510.03495 arXiv:2510.03495 [cs].

[64] H Qiu, YL Li, HS Padala, A Sarma, and B Vasilescu. 2019. The Signals that Potential Contributors Look for When Choosing Open-source Projects. *ACM Proceedings on Human-Computer Interaction* (2019).

[65] Eric Raymond. 1999. The cathedral and the bazaar. *Knowledge, Technology & Policy* (1999).

[66] James Reason. 1990. The contribution of latent human failures to the breakdown of complex systems. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences* 327, 1241 (1990), 475–484.

[67] Mark Russinovich and Ahmed Salem. 2024. Hey, That's My Model! Introducing Chain & Hash, An LLM Fingerprinting Technique. *CoRR* abs/2407.10887 (2024). doi:10.48550/arXiv.2407.10887

[68] Taylor R Schorlemmer, Ethan H Burmane, Kelechi G Kalu, Santiago Torres-Arias, and James C Davis. 2025. Establishing Provenance Before Coding: Traditional and Next-Generation Software Signing. *IEEE Security & Privacy* (2025).

[69] Taylor R Schorlemmer, Kelechi G Kalu, Luke Chigges, Kyung Myung Ko, Eman Abu Ishgair, Saurabh Bagchi, Santiago Torres-Arias, and James C Davis. 2024. Signing in four public software package registries: Quantity, quality, and influencing factors. In *IEEE Symposium on Security and Privacy (SP).*

[70] Scott. 2026. An AI Agent Published a Hit Piece on Me. https://theshamblog.com/an-ai-agent-published-a-hit-piece-on-me/. The Shamblog blog post. Accessed 2026-04-01.

[71] Ankur Shukla, Basel Katt, and Livinus Obiora Nweke. 2019. Vulnerability Discovery Modelling With Vulnerability Severity. In *IEEE Conference on Information and Communication Technology.* doi:10.1109/CICT48419.2019.9066187

[72] Hayden Smith. 2024. Where the Wild Things Are: A Complete Analysis of Jia Tan's GitHub History and the XZ Utils Software Supply Chain Breach. https://tinyurl.com/6rush4hd

[73] Sonatype. 2021. *2021 State of the Software Supply Chain Report.* Technical Report. Sonatype. https://www.sonatype.com/resources/whitepapers/2021-state-of-the-software-supply-chain-report-2021 Whitepaper; accessed: 2025-05-18.

[74] Murugiah Souppaya, Karen Scarfone, and Donna Dodson. 2022. *Secure Software Development Framework (SSDF) Version 1.1.* Technical Report. National Institute of Standards and Technology. doi:10.6028/NIST.SP.800-218

[75] Diomidis Spinellis and Clemens Szyperski. 2004. How is open source affecting software development? *IEEE software* 21, 1 (2004), 28.

[76] Joseph Spracklen, Raveen Wijewickrama, A. H. M. Nazmus Sakib, Anindya Maiti, Bimal Viswanath, and Murtuza Jadliwala. 2024. We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs. *CoRR* abs/2406.10279 (2024). doi:10.48550/arXiv.2406.10279

[77] Stack Overflow Help Center. [n. d.]. What is reputation? https://stackoverflow.com/help/whats-reputation. Accessed: 2026-01-26.

[78] Gayatri Swamynathan, Kevin C. Almeroth, and Ben Y. Zhao. 2010. The design of a reliable reputation system. *Electronic Commerce Research* (Dec. 2010).

[79] Mike Sweeney. 2023. What motivates a developer to contribute to open-source software? https://clearcode.cc/blog/why-developers-contribute-open-source-software/. Accessed: 2026-01-26.

[80] Chen Tang, Lan Zhang, Qi Zhao, Xirong Zhuang, and Xiang-Yang Li. 2025. Model Lineage Closeness Analysis. *Proceedings of the AAAI Conference on Artificial Intelligence* 39, 19 (2025), 20796–20804. doi:10.1609/aaai.v39i19.34292

[81] The Linux Foundation. 2023. Supply-chain Levels for Software Artifacts (SLSA). https://slsa.dev/. Accessed: 2026-01-26.

[82] 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 *USENIX Security Symposium.*

[83] Yun-Yun Tsai, Chuan Guo, Junfeng Yang, and Laurens van der Maaten. 2025. RoFL: Robust Fingerprinting of Language Models. *CoRR* abs/2505.12682 (2025). doi:10.48550/arXiv.2505.12682

[84] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of social and technical factors for evaluating contribution in GitHub. In *International Conference on Software Engineering (ICSE).*

[85] U.S. Department of Defense. 2009. Clarifying Guidance Regarding Open Source Software (OSS). Memorandum. https://dodcio.defense.gov/Portals/0/Documents/FOSS/2009OSS.pdf Accessed: 2026-01-26.

[86] Vidos.id. n.d.. *Google Wallet Adds Digital IDs: What Does This Mean for the Future of Identity?* https://vidos.id/blog/google-wallet-adds-digital-ids-what-does-this-mean-for-the-future-of-identity Accessed: 2025-05-20.

[87] Duc-Ly Vu, Fabio Massacci, Ivan Pashchenko, Henrik Plate, and Antonino Sabetta. 2021. Lastpymile: Identifying the discrepancy between sources and packages. In *ESEC/FSE.*

[88] Shaowei Wang, Daniel M German, Tse-Hsun Chen, Yuan Tian, and Ahmed E Hassan. 2021. Is reputation on Stack Overflow always a good indicator for users' expertise? No!. In *International Conference on Software Maintenance and Evolution (ICSME).*

[89] Yanlin Wang, Wanjun Zhong, Yanxian Huang, Ensheng Shi, Min Yang, Jiachi Chen, Hui Li, Yuchi Ma, Qianxiang Wang, and Zibin Zheng. 2025. Agents in software engineering: survey, landscape, and vision. *Automated Software Engineering* 32, 2 (Aug. 2025), 70. doi:10.1007/s10515-025-00544-2

[90] Dominik Wermke, Noah Wöhler, Jan H. Klemmer, Marcel Fourné, Yasemin Acar, and Sascha Fahl. 2022. Committed to Trust: A Qualitative Study on Security and Trust in Open Source Software Projects. In *IEEE Symposium on Security and Privacy (SP).*

[91] Marcus Willett. 2023. Lessons of the SolarWinds hack. In *Survival April–May 2021: Facing Russia.* Routledge, 7–25.

[92] Tianbao Xie, Fan Zhou, Zhoujun Cheng, Peng Shi, Luoxuan Weng, Yitao Liu, Toh Jing Hua, Junning Zhao, Qian Liu, Che Liu, et al. [n. d.]. OpenAgents: An Open Platform for Language Agents in the Wild. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents.*

[93] Yue Yu, Gang Yin, Huaimin Wang, and Tao Wang. 2014. Exploring the patterns of social behavior in GitHub. In *Proceedings of the 1st International Workshop on Social Software Engineering.* Check exact workshop title/venue if needed.

[94] Nusrat Zahan, Parth Kanakiya, Brian Hambleton, Shohanuzzaman Shohan, and Laurie Williams. 2023. Openssf scorecard: On the path toward ecosystem-wide automated security metrics. *IEEE Security & Privacy* (2023).