

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

Kelechi G. Kalu*, Sofia Okorafor*, Betül Durak†, Kim Laine†, Radames C. Moreno†, Santiago Torres-Arias*, and James C. Davis*

*Purdue University

{kalu, sokorafo, santiagotorres, davisjam}@purdue.edu

†Microsoft Research

{betul.durak, kim.laine, radames.cruz}@microsoft.com

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 change request, assessing a change request’s cybersecurity implications is challenging. To help maintainers make this decision, we propose that the open-source ecosystem should incorporate Actor Reputation Metrics (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.

Index Terms—Software Supply Chain, Reputation systems

I. INTRODUCTION

Most commercial software depends on open-source software components [1]. Although this approach reduces development costs, it results in a software supply chain that exposes an organization to cybersecurity risks [2], [3]. Many prior works have examined software supply chain security failures and have developed security techniques [4]–[6], engineering processes and frameworks [7]–[9]. These works have had substantial success, *e.g.*, the now widely-used Sigstore project for provenance [4], and OpenSSF Scorecard project [10] for process. However, prior works have paid little attention to the *actor* element of the software supply chain [8].

In this vision paper, we propose *ARMS*, an Actor Reputation Metric System, to track the security qualifications of engineers in the open-source software supply chain. 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 propose Actor Reputation Metric Systems (ARMS) to support open-source maintainers in vetting contributions from unknown engineers. Focusing on cybersecurity, we identify signals that could indicate cybersecurity expertise, and metrics that could be used to operationalize those signals.
- We design experiments that could be used to evaluate ARMS, and discuss the pros and cons of deploying such a system.

II. BACKGROUND AND MOTIVATION

A. The Open-Source Software Supply Chain

Open-source software is widely integrated into commercial [11] and government [12] systems. Any individual open-source component is developed by a *maintainer team*. With their approval, outsiders may be permitted to contribute code [13]. 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 [14]. 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 [15].

We follow the software supply chain definition of Okafor *et al.* [8]: 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.

B. 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 [13]. As a software package gains popularity, interest from potential contributors increases [16]–[18], often resulting in onboarding new maintainers and merging change requests from new contributors. Evaluating these individuals

may involve reputational factors such as community status and connections [16], [19], [20], but security considerations are rarely enforced due to the challenges OSS teams face in managing security resources effectively [21], [22].

As argued by Okafor *et al.* [8], most cybersecurity work takes an artifact-based perspective. Efforts to develop static and dynamic security analysis tools (SAST, DAST) [23], refine code reviews [24], [25], and assess artifact provenance [4], [26], [27] 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 [28] *e.g.*, Log4j [29].
- **Biased human review.** Manual code reviews and audits are applied inconsistently, frequently favoring contributors familiar to project maintainers [18], [19], [30].
- **Scalability constraints.** As projects grow, sustaining thorough artifact checks becomes resource-intensive, leading to superficial reviews or delayed patching [30], [31].
- **Limited socio-technical insight.** Artifact-centric methods inspect code but ignore the developer behaviors and workflows that often precipitate security issues [32].

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

C. Using Actor Reputation to Establish Trust

Reputation systems establish trust between parties who have not been previously connected [33]. When social systems integrate reputation, incentives associated with positive reputation can encourage good behavior over time [34]. Following Hendriks *et al.* [35], 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 [33], online-auction [36], etc.

Two common examples of reputation systems in software engineering are GitHub’s star system [37] and the Stack Overflow point-based system [38]. Both of these systems can be used by a trustor to quantify the number of users satisfied with a trustee’s projects and contributions [39]. As a result, they might influence which GitHub projects an engineer (trustor) may deem to be reliable [40], and which Stack Overflow answers may be trusted by their readers [41].

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

III. THREAT MODEL

A. Threat Actors

The goal of ARMS is to provide project maintainers with measurements of the security expertise of prospective contributors or maintainers. ARMS operates within a context with 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 [42], [43].
- 2) **Reputation Spoofing:** Malicious actors deliberately craft the appearance of security expertise to gain collaborator or maintainer status.
- 3) **Impersonation.** Impersonation occurs when a malicious actor gains control of a legitimate user’s account (*e.g.*, via key compromise [44], [45]).

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 [46].

B. Examples

We give examples of each kind of threat we outlined above.

1) *Dexcom (Inadvertent Vulnerability):* Dexcom is a medical device company whose products include continuous glucose monitors (CGMs) used by diabetics [47]. 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 [48]. 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.

2) *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 [49]. The vulnerability was introduced by an actor who had built trust within the project through non-malicious contributions, and they were eventually promoted to co-maintainer [49]. In retrospect, this particular actor had several suspicious reputational signals, most notably having no accounts on other sites and no history of contributions to other projects.

3) *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 [45]. Attackers gained access to a maintainer’s npm credentials and published malicious package updates to the npm registry [50]. Because the attacker used the identify of a reputable maintainer, a reputation system (which assume stable identities) could not anticipate this attack.

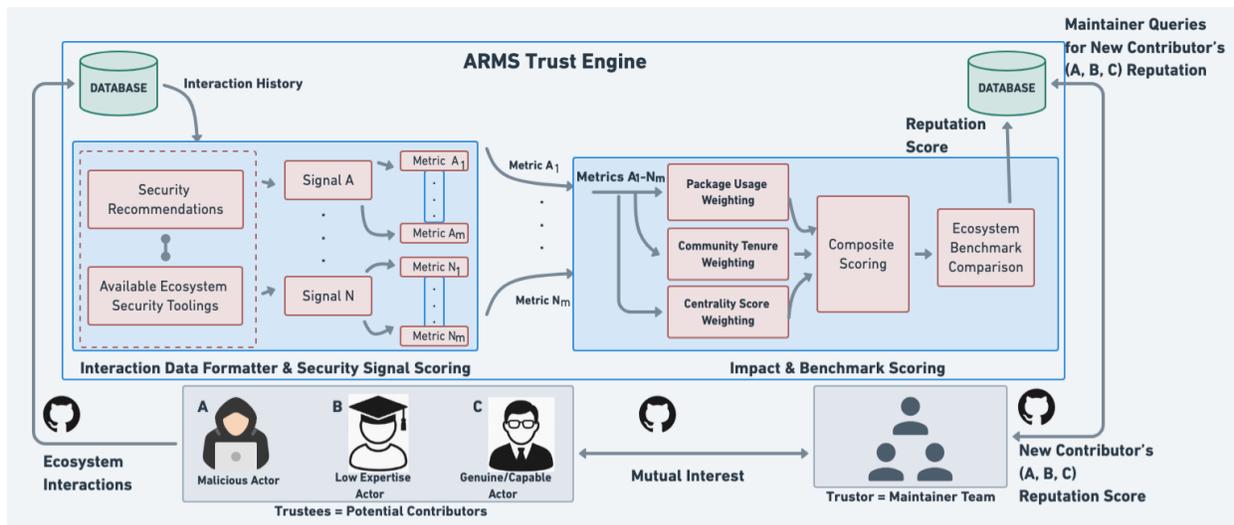


Fig. 1: **Overview of proposed ARMS system and context case study.** Potential contributors (trustees), who may be malicious (*Actor A*), inadequately expertised (*Actor B*), or genuine (and capable) (*Actor C*), express interest or submit change requests. The maintainer team (trustors) requests reputation 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 weights 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 maintainer team.

IV. 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.* [35] described earlier, and adapt it to the OSS supply chain context. The following subsections outline our proposed system, define security metrics for an actor’s security reputation, and propose evaluation metrics to measure the effectiveness of these security metrics.

A. System Overview

Our proposed system follows the three-element model of Hendrikz *et al.* (§II-C) 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 and case study 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 [51], the proposed frameworks are based on defining trust, our work operationalizes trust with reputation systems. In the next section, we define metrics to assess security interactions and history within a typical OSS ecosystem.

B. Interaction Data – Security Signals and Metrics Definitions

Our system computes reputation from an actor’s historical interactions within the ecosystem. We categorize these interactions by security signals and quantify each signal using measurable 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 [57], the CNCF software supply chain security guidelines [58], the NIST SSDF security framework [59], NIST SP 800-204D [60], Openssf S2C2F [61], and the CIS Software Supply Chain Security Guide [62]. Common recommendations across these sources were used to ensure the selection of well-established security 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.

We grouped the resulting metrics into seven categories – Security Signals, based on contributors’ security tool usage and vulnerability management practices. A summary appears in Table I. This resulted in seven security signals

Table I: Proposed security signals. Bolded signal categories are derived from security recommendations (e.g, NIST, CNCF, etc). The proposed evaluation metrics in Table II offer ways to evaluate the signals below.

S/N. Proposed Signals & Analysis Metrics	Description
<i>SECURITY SIGNALS</i>	
1. Security vulnerability in artifact – Pull Requests/Commits [51]–[53]	Measures vulnerabilities introduced through pull requests or commits. Introduced either by the user or to a project owned by the user.
2. Security vulnerability in artifact – use of vulnerable dependencies [51], [52]	Assesses the use and introduction of vulnerable dependencies into the artifact.
3. Use of ecosystem code scanning and security analysis features [54]	Evaluates usage of ecosystem’s security tools for code scanning.
4. Use of ecosystem integrity guarantees [27]	Examines use of ecosystem integrity features (e.g., code signing).
5. Use of branch protection [6], [55]	Checks if branch protection is enforced to prevent unauthorized changes.
6. Use of security policies and vulnerability reporting [55]	Determines if the project has security policies and reporting mechanisms.
7. Use of automated workflows [55]	Assesses use of [55] automation in workflows to enforce security.
<i>REPUTATION WEIGHTAGE SIGNALS</i>	
W1. Package Usage [56]	Weighs the utility of the potential contributor’s owned projects
W2. Community Tenure [51]	Weighs the the length of the contributor’s belonging to the ecosystem
W3. Centrality Score [18]	Weighs the degree of connections of the potential contributor to other actors in the ecosystem.

C. Trust Engine – Reputation Computation

In this section, we outline the core functionality of the trust engine’s reputation score computation (see Figure 1). First, the interaction data formatter and security signal scorer extracts each contributor’s ecosystem interaction history and quantify each signal according to the metrics in Table II. These base measurements capture a user’s adherence to the defined security signals and their contribution patterns.

Next, ARMS evaluates each contributor’s reputation. To account for the risk posed by a contributor’s activities within the ecosystem, we refine the initial signal scores by incorporating:

Impact Score – (Weightage Signals Table I (W1-W3)):

- 1) **Package Usage Weighting:** Projects with no users (e.g., private repositories or those with zero forks/downloads) should not contribute to the user’s evaluation score, as they present minimal supply chain risk.
- 2) **Community Tenure Weighting [51]:** This ensures that newer users do not unduly influence their reputation score, addressing concerns like those in the “XZ utils” situation [63], where malicious activity was introduced by new members.
- 3) **Centrality Score Weighting [18]:** Centrality—an actor’s level of connectedness within the ecosystem based on contribution activities—can be crucial for scaling reputation. We suggest evaluating not only the number of connections (edges) but also the time taken to establish them.

Benchmark Score:

- 1) **Combination of multiple security metrics/Composite Scoring:** A holistic evaluation will aggregate a user’s performance across various security metrics, offering a comprehensive view of their adherence to security practices. This approach minimizes false positives and negatives by drawing from multiple metrics, as seen in the OpenSSF

scorecard [64]. Each metric’s weighting will be based on its *effectiveness analysis* §V-A.

- 2) **Ecosystem Benchmarks:** User performance will be benchmarked against the average or median reputation scores across the ecosystem to provide a relative measure of security adherence.

V. DESIGN OF EXPERIMENTS FOR ARMS

In this section, we outline potential studies to establish and evaluate the feasibility of an actor reputation system, exemplified by ARMS, and general 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. We categorize these studies into two groups: (1) evaluating the effectiveness of the proposed security metrics, and (2) examining user behaviors.

A. Security Metrics Effectiveness Evaluation – Quasi-Experimental Analyses

To evaluate the utility of the proposed security signals, we propose the following quasi-experimental studies to determine whether the signals reliably predict poor security practices.

- 1) **Inter-metric Relationship Study:** Compare OSS projects that adopted a given security practice (e.g., branch protection) 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 interaction terms for the paired metrics to isolate their combined impact on security outcomes.
 - *Data:* Historical GitHub records for projects within the same domain, similar size, and activity levels.
 - *Outcomes:* Changes in vulnerability incidence (Signals 1–2) and shifts in contributor behaviors (Signals 3–7).

- *Ethics*: Uses only publicly available data, avoiding additional data collection or privacy concerns.
- 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 similar “clean” projects.
 - *Data*: Metric values for each project’s maintainers before the incident.
 - *Analysis*: Logistic regression (with interaction terms) to determine which signals best predict incident occurrence.

B. 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 a Recruitment of active OSS maintainers to participate in a vignette study [65]. This would present anonymized contributor profiles with varying metric scores, 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 “chilling effect,” [66] where contributors may hesitate or reduce participation if they know their interactions are being tracked and scored [66]–[68]. 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 effectively do this, there is need to develop an effective project importance score. OpenSSF’s criticality score [69] may be used or serve as a starting point. We recommend surveying OSS contributors to gauge a suitable threshold of criticality.

C. Worked Examples

We illustrate how our proposed system could have worked in the earlier described examples §III-B.

1) *XZ Utils Attack*: The timeline of events leading to the XZ Utils backdoor reveals several characteristics that map to our proposed security signals [70]:

- 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 change requests**: Their first public change 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 score:

- *Signal Metrics* (Signals 1-7) penalize sparse or opaque contribution histories.
- *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.

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.

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 in the system threat Model §III.

VI. DISCUSSIONS AND FUTURE WORKS

A. 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 §V-A. 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 [71], and a reputation system would be no exception.
- 6) **Possibility of Chilling Effect:** Discussed in §V-B.

B. 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—for example, by analyzing anomalous contribution patterns, unusual social graph connections, or timing irregularities, ultimately creating a more comprehensive reputation model.

C. Supporting Ecosystem Heterogeneity

Reputation systems fundamentally face actor-identification challenges [72], [73]. 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 [4], [74], [75], 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 [76]) propose establishing identities through multiple institutions—some designed to preserve privacy [77], [78]. 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 [79] and Distributed Identities [80]). Thus, for an actor reputation system like ARMS, institutions across ecosystems must collaborate and share reputation information whenever a software artifact from one ecosystem A is used in another ecosystem B.

D. Social & Process Signals: Beyond Artifact Contributions

Our current security (and weightage) signals (Table I) 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 [81] are most effective in structured development environments. However, in open-source ecosystems—where formal processes are often absent [82]—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 change 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.

E. 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 [83] 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.

F. 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, in part because the validation is apparently performed through use case analysis rather than through software behavioral analysis [71]. Similarly, behaviors for regulatory compliance such as GDPR are challenging to validate [11]. 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.

VII. CONCLUSION

In this paper, we explored the characteristics of the open-source software supply chain that necessitate actor-based security techniques. 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 metrics for measuring reputation, alongside two evaluation methods: one for assessing actor performance and another for determining the effectiveness of these metrics in predicting reputation. Lastly, we identified future research directions, including empirical studies, system prototypes, and initiatives to maintain actor identities across ecosystems.

REFERENCES

- [1] Sonatype, “2021 state of the software supply chain report,” Sonatype, Tech. Rep., 2021, whitepaper; accessed: 2025-05-18. [Online]. Available: <https://www.sonatype.com/resources/whitepapers/2021-state-of-the-software-supply-chain-report-2021>
- [2] S. Benthall, “Assessing software supply chain risk using public data,” in *2017 IEEE 28th Annual Software Technology Conference (STC)*, Sep. 2017, pp. 1–5.
- [3] M. Willett, “Lessons of the solarwinds hack,” in *Survival April–May 2021: Facing Russia*. Routledge, 2023, pp. 7–25.
- [4] Z. Newman, J. S. Meyers, and S. Torres-Arias, “Sigstore: Software signing for everybody,” in *Conf. on Computer and Comms. Security*, 2022.
- [5] S. Torres-Arias, H. Afzali, T. K. Kuppasamy, R. Curtmola, and J. Cappos, “in-toto: Providing farm-to-table guarantees for bits and bytes,” in *USENIX Security*, 2019.
- [6] N. Zahan, P. Kanakiya, B. Hambleton, S. Shohan, and L. Williams, “Openssf scorecard: On the path toward ecosystem-wide automated security metrics,” *IEEE Security & Privacy*, 2023.
- [7] T. M. S. do Amaral and J. J. C. Gondim, “Integrating zero trust in the cyber supply chain security,” in *Workshop on Communication Networks and Power Systems*. IEEE, 2021.
- [8] C. Okafor, T. R. Schorlemmer, S. Torres-Arias, and J. C. Davis, “Sok: Analysis of software supply chain security by establishing secure design properties,” in *ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, 2022.
- [9] M. S. Melara and M. Bowman, “Hardware-enforced integrity and provenance for distributed code deployments,” *arXiv:2106.09843*, 2021.
- [10] openssf, “Openssf scorecard.” [Online]. Available: <https://securityscorecards.dev/>
- [11] L. Franke, H. Liang, S. Farzanehpour, A. Brantly, J. C. Davis, and C. Brown, “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*, 2024.
- [12] U.S. Department of Defense, “Clarifying guidance regarding open source software (oss),” Memorandum, Oct 2009. [Online]. Available: <https://dodcio.defense.gov/Portals/0/Documents/FOSS/2009OSS.pdf>
- [13] E. Raymond, “The cathedral and the bazaar,” *Knowledge, Technology & Policy*, 1999.
- [14] M. Sweeney, “What motivates a developer to contribute to open-source software?” Nov 2023. [Online]. Available: <https://clearcode.cc/blog/why-developers-contribute-open-source-software/>
- [15] M. Ohm, H. Plate, A. Sykosch, and M. Meier, “Backstabber’s knife collection: A review of open source software supply chain attacks,” in *Detect. of Intrusions and Malware, and Vuln. Assess.*, 2020.
- [16] D. Maldeniya, C. Budak, L. P. Robert Jr, and D. M. Romero, “Herding a deluge of good samaritans: How github projects respond to increased attention,” in *The Web Conference (WWW)*, 2020.
- [17] H. Qiu, Y. Li, H. Padala, A. Sarma, and B. Vasilescu, “The signals that potential contributors look for when choosing open-source projects,” *ACM Proceedings on Human-Computer Interaction*, 2019.
- [18] S. Hamer, N. Imtiaz, M. Tamanna, P. Shabrina, and L. Williams, “Trusting code in the wild: Exploring contributor reputation measures to review dependencies in the rust ecosystem,” *IEEE Transactions on Software Engineering*, vol. 51, no. 4, pp. 1319–1333, 2025.
- [19] J. Tsay, L. Dabbish, and J. Herbsleb, “Influence of social and technical factors for evaluating contribution in github,” in *International Conference on Software Engineering (ICSE)*, 2014.
- [20] Y. Yu, G. Yin, H. Wang, and T. Wang, “Exploring the patterns of social behavior in github,” in *1st International Workshop on...*, 2014.
- [21] D. Wermke, N. Wöhler, J. H. Klemmer, M. Fourné, Y. Acar, and S. Fahl, “Committed to Trust: A Qualitative Study on Security & Trust in Open Source Software Projects,” in *IEEE Symposium on Security and Privacy (SP)*, May 2022.
- [22] S. Amft, S. Höltervennhoff, R. Panskus, K. Marky, and S. Fahl, “Everyone for themselves? a qualitative study about individual security setups of open source software contributors,” in *IEEE Symposium on Security and Privacy (SP)*, 2024.
- [23] S. Elder, N. Zahan, R. Shu, M. Metro, V. Kozarev, T. Menzies, and L. Williams, “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.
- [24] M. A. Howard, “A process for performing security code reviews,” *IEEE Security & privacy*, vol. 4, no. 4, pp. 74–79, 2006.
- [25] L. Braz and A. Bacchelli, “Software security during modern code review: the developer’s perspective,” in *2022 European Software Engineering Conference & Symposium on the Foundations of Software Engineering*.
- [26] D.-L. Vu, F. Massacci, I. Pashchenko, H. Plate, and A. Sabetta, “Last-pymile: identifying the discrepancy between sources and packages,” in *Foundations of Software Engineering (ESEC/FSE)*, 2021.
- [27] T. R. Schorlemmer, K. G. Kalu, L. Chigges, K. M. Ko, E. A. Ishgair, S. Bagchi, S. Torres-Arias, and J. C. Davis, “Signing in four public software package registries: Quantity, quality, and influencing factors,” in *IEEE Symposium on Security and Privacy (SP)*, 2024.
- [28] S. Pan, L. Bao, J. Zhou, X. Hu, X. Xia, and S. Li, “Towards More Practical Automation of Vulnerability Assessment,” in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE ’24. New York, NY, USA: Association for Computing Machinery, Apr. 2024, pp. 1–13. [Online]. Available: <https://dl.acm.org/doi/10.1145/3597503.3639110>
- [29] R. Hiesgen, M. Nawrocki, T. C. Schmidt, and M. Wahlisch, “The Race to the Vulnerable: Measuring the Log4j Shell Incident.”
- [30] O. Kononenko, O. Baysal, and M. W. Godfrey, “Code review quality: how developers see it,” in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE ’16, May 2016.
- [31] M. Harman and P. O’Hearn, “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, 2018.
- [32] K. Kalu, T. R. Schorlemmer, S. Chen, K. A. Robinson, E. Kocinare, and J. C. Davis, “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*, ser. ESEC/FSE 2023. New York, NY, USA: Association for Computing Machinery, Nov. 2023, pp. 2112–2116. [Online]. Available: <https://dl.acm.org/doi/10.1145/3611643.3613075>
- [33] F. B. Durak, K. Laine, S. Langowski, R. C. Moreno, R. Sim, and S. Jain, “Sandi: A System for Accountability and Applications in Direct Communication (Extended Abstract),” Nov. 2023.
- [34] A. Josang, R. Ismail, and C. Boyd, “A survey of trust and reputation systems for online service provision,” *Decision support systems*, 2007.

- [35] F. Hendriks, K. Bubendorfer, and R. Chard, "Reputation systems: A survey and taxonomy," *Journ. of Parallel and Distr. Computing*, 2015.
- [36] M. T. Goodrich and F. Kerschbaum, "Privacy-Enhanced Reputation-Feedback Methods to Reduce Feedback Extortion in Online Auctions," in *Conference on Data and Application Security and Privacy*, 2011.
- [37] [Online]. Available: <https://docs.github.com/en/get-started/exploring-projects-on-github/saving-repositories-with-stars>
- [38] [Online]. Available: <https://stackoverflow.com/help/whats-reputation>
- [39] K. Cameron, "The laws of identity," May 2005. [Online]. Available: <https://www.identityblog.com/?p=352>
- [40] H. Borges and M. T. Valente, "What's in a github star? understanding repository starring practices in a social coding platform," *Journal of Systems and Software*, 2018.
- [41] S. Wang, D. M. German, T.-H. Chen, Y. Tian, and A. E. Hassan, "Is reputation on stack overflow always a good indicator for users' expertise? no!" in *International Conference on Software Maintenance and Evolution (ICSME)*, 2021.
- [42] D. A. Norman, *The Design of Everyday Things*, revised and expanded ed. Cambridge, MA: MIT Press, 2013.
- [43] J. Reason, "The contribution of latent human failures to the breakdown of complex systems," *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, vol. 327, no. 1241, pp. 475–484, 1990.
- [44] Codex Security Team. (2021, Apr) Bash uploader security update. Accessed: 2025-05-15. [Online]. Available: <https://tinyurl.com/5d8bd4je>
- [45] ESLint Team. (2018, Jul) Postmortem for malicious packages published on july 12th, 2018. Accessed: 2025-05-15. [Online]. Available: <https://eslint.org/blog/2018/07/postmortem-for-malicious-package-publishes/>
- [46] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*. Springer, 2002, pp. 251–260.
- [47] S. Mukherjee, "Dexcom software outage draws fury from diabetes patients' parents," Dec. 2019. [Online]. Available: <https://fortune.com/2019/12/02/dexcom-outage-blackout-diabetes-patients-blood-sugar-monitor/>
- [48] A. O'Connor, "In weekend outage, diabetes monitors fail to send crucial alerts," Dec 2019. [Online]. Available: <https://www.nytimes.com/2019/12/02/well/live/Dexcom-G6-diabetes-monitor-outage.html>
- [49] M. Lins, R. Mayrhofer, M. Roland, D. Hofer, and M. Schwaighofer, "On the critical path to implant backdoors and the effectiveness of potential mitigation techniques: Early learnings from xz," *arXiv preprint arXiv:2404.08987*, 2024.
- [50] Cocode Security Team. (2022, Feb) Eslint: Compromising the build using a supply chain attack. Accessed: 2025-05-15. [Online]. Available: <https://cocode.com/blog/eslint-compromising-the-build-using-supply-chain-attack/>
- [51] L. Boughton, C. Miller, Y. Acar, D. Wermke, and C. Kästner, "Decomposing and Measuring Trust in Open-Source Software Supply Chains," in *International Conf. on Software Eng—NIER (ICSE-NIER)*, Apr. 2024.
- [52] A. Shukla, B. Katt, and L. O. Nweke, "Vulnerability discovery modelling with vulnerability severity," in *IEEE Conference on Information and Communication Technology*, 2019.
- [53] D. Gonzalez, T. Zimmermann, P. Godefroid, and M. Schaefer, "Anomalous: Automated Detection of Anomalous and Potentially Malicious Commits on GitHub," in *International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2021.
- [54] F. Fischer, J. Höbenreich, and J. Grossklags, "The effectiveness of security interventions on github," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023.
- [55] [Online]. Available: <https://scorecard.dev/>
- [56] S. Boysel, "No Free Lunch For Programmers: Digital Supply Chains and the Economics of Software Dependency Management."
- [57] The Linux Foundation, "Supply-chain levels for software artifacts," <https://slsa.dev/>.
- [58] C. N. Computing, "CNCF paper defines best practices for supply chain security — cncf.io," 2021. [Online]. Available: <https://www.cncf.io/announcements/2021/05/14/cncf-paper-defines-best-practices-for-supply-chain-security/>
- [59] M. Souppaya, K. Scarfone, and D. Dodson, "Secure Software Development Framework (SSDF) Version 1.1," National Institute of Standards and Technology, Tech. Rep., Feb. 2022. [Online]. Available: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218.pdf>
- [60] R. Chandramouli, R. Chandramouli, F. Kautz, and S. Torres-Arias, *Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD Pipelines*. US Department of Commerce, National Institute of Standards and Technology, 2024.
- [61] OpenSSF. (n.d.) s2c2f. Accessed: 2025-05-20. [Online]. Available: <https://github.com/ossf/s2c2f>
- [62] Center for Internet Security, "CIS Software Supply Chain Security Guide," Center for Internet Security, Tech. Rep., 2021, accessed: 2025-05-20. [Online]. Available: <https://www.cisecurity.org/white-papers/cis-software-supply-chain-security-guide>
- [63] D. Bors, "CVE-2024-3094 The XZ Utils Backdoor, a critical SSH vulnerability in Linux," <https://pentest-tools.com/blog/xz-utils-backdoor-cve-2024-3094>, [Accessed 09-10-2024].
- [64] N. Zahan, P. Kanakiya, B. Hambleton, S. Shohan, and L. Williams, "OpenSSF Scorecard: On the Path Toward Ecosystem-wide Automated Security Metrics," Jan. 2023, arXiv:2208.03412.
- [65] H. Aguinis and K. J. Bradley, "Best practice recommendations for designing and implementing experimental vignette methodology studies," *Organizational research methods*, vol. 17, no. 4, 2014.
- [66] B. Marder, A. Joinson, A. Shankar, and D. Houghton, "The extended 'chilling' effect of facebook: The cold reality of ubiquitous social networking," *Computers in Human Behavior*, vol. 60, pp. 582–592, 2016.
- [67] M. Büchi, N. Festic, and M. Latzer, "The chilling effects of digital dataveillance: A theoretical model and an empirical research agenda," *Big Data & Society*, vol. 9, no. 1, p. 20539517211065368, 2022.
- [68] T. Bernauer and M. M. Dapp, "Hot debate about chilling effects: Do software patterns hamper/free open source software development?" 2009.
- [69] A. Arya, C. Brown, R. Pike, and T. O. S. S. Foundation, "Open Source Project Criticality Score," https://github.com/ossf/criticality_score, 2023.
- [70] H. Smith, "Where the wild things are: A complete analysis of jia tan's github history and the xz utils software supply chain breach," Apr. 2024. [Online]. Available: <https://tinyurl.com/6rush4hd>
- [71] G. Cramer, W. P. Maxam III, and J. C. Davis, "Engineering patterns for trust and safety on social media platforms: A case study of mastodon and diaspora," *Journal of Systems and Software*, 2025.
- [72] G. Swamynathan, K. C. Almeroth, and B. Y. Zhao, "The design of a reliable reputation system," *Electronic Commerce Research*, Dec. 2010.
- [73] S. Marti and H. Garcia-Molina, "Identity crisis: anonymity vs reputation in p2p systems," in *International Conference on Peer-to-Peer Computing (P2P2003)*, 2003.
- [74] K. G. Kalu, T. Singla, C. Okafor, S. Torres-Arias, and J. C. Davis, "An industry interview study of software signing for supply chain security," in *34th USENIX Security Symposium (USENIX Security 25)*. Seattle, WA, USA: USENIX Association, aug 2025.
- [75] T. R. Schorlemmer, E. H. Burmane, K. G. Kalu, S. Torres-Arias, and J. C. Davis, "Establishing provenance before coding: Traditional and next-generation software signing," *IEEE Security & Privacy*, 2025.
- [76] cisa.gov, "Software identification ecosystem option analysis," <https://www.cisa.gov/sites/default/files/2023-10/Software-Identification-Ecosystem-Option-Analysis-508c.pdf>, 2023.
- [77] Microsoft. (n.d.) Microsoft entra id. Accessed: 2025-05-20. [Online]. Available: <https://www.microsoft.com/en-us/security/business/identity-access/microsoft-entra-id>
- [78] Vidos.id. (n.d.) Google wallet adds digital ids: What does this mean for the future of identity? Accessed: 2025-05-20. [Online]. Available: <https://vidos.id/blog/google-wallet-adds-digital-ids-what-does-this-mean-for-the-future-of-identity>
- [79] O. Foundation, "How OpenID Connect Works - OpenID Foundation," <https://openid.net/developers/how-connect-works/>.
- [80] T. Bouma, "High Assurance DIDs with DNS," <https://www.ietf.org/archive/id/draft-carter-high-assurance-dids-with-dns-03.html>, 2024.
- [81] C. Clancy, J. Ferraro, R. Martin, A. Pennington, C. Sledjeski, and C. Wiener, "Deliver uncompromised: Securing critical software supply chains," *MITRE Technical Papers*, vol. 24, no. 01, 2021.
- [82] D. Spinellis and C. Szyperski, "How is open source affecting software development?" *IEEE software*, vol. 21, no. 1, p. 28, 2004.
- [83] European Union. (2016) General data protection regulation (gdpr). Regulation (EU) 2016/679. Accessed: 2025-04-30. [Online]. Available: <https://gdpr-info.eu/>

APPENDIX

OUTLINE OF APPENDICES

The appendix contains the following material:

- Appendix A: A comprehensive Signal Metrics Table.

Table II: 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.

S/N. Proposed Signals & Analysis Metrics	Description
<i>SECURITY SIGNALS</i>	
1. Security vulnerability in artifact – Pull Requests/Commits	Measures vulnerabilities introduced through pull requests or commits. Introduced either by the user or to a project owned by the user.
a. Time to fix/close security vulnerabilities of various severity levels [51]	Time taken to address vulnerabilities detected in pull requests or commits.
b. Severity levels of security vulnerability reported [52]	Evaluates the criticality of reported vulnerabilities (low, medium, high), for user’s projects.
c. Presence of vulnerability in PR/commits (not linked to external dependencies) [53]	Measures severity of vulnerabilities that stem from 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 [51]	Time taken to resolve vulnerable dependencies, after they have been detected (or publicized).
b. Severity levels of package’s reported vulnerability [52]	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 users total projects.
d. Number of vulnerable dependencies/project	Assesses how widespread vulnerable dependencies are within each project.
e. Number of projects with reported vulnerable dependencies	Measures the overall exposure of projects to vulnerable dependencies.
3. Use of ecosystem code scanning and security analysis features	Evaluates usage of ecosystem’s security tools for code scanning.
a. Status of dependabot alerts (for dependencies) [54]	Monitors if security alerts for dependencies are being addressed.
b. Status of Secret sharing/Validity Checks	Evaluates the management of secret-sharing mechanisms and validity checks.
c. Code scanning and Vulnerability alerts [54]	Assesses if security scanning tools are actively used and vulnerability alerts managed.
d. Push Protection Status	Checks if protection features are used to block pushes with exposed secrets (e.g., unscanned code).
4. Use of ecosystem integrity guarantees	Examines use of ecosystem integrity features (e.g., code signing).
a. Number of Projects with an integrity guarantee [27]	Measures how many projects are secured with integrity verification features such as code signing.
5. Use of branch protection	Checks if branch protection is enforced to prevent unauthorized changes.
a. Number of protected branches per project	Evaluates how many branches within each project are safeguarded against direct commits.
b. Number of Projects with Protected branches [55], [64]	Tracks how many projects enforce branch protection.
6. Use of security policies and vulnerability reporting	Determines if the project has security policies and reporting mechanisms.
a. Status of private vulnerability reporting, or security policy [55]	Checks if private vulnerability reporting channels and security policies are established and functional.
7. Use of automated workflows	Assesses use of [55] automation in workflows to enforce security.
a. Number of Projects with Automated workflows [55]	Measures how many user’s projects use automated workflows for security enforcement, such as auto-deployments or auto-tests.
<i>REPUTATION WEIGHTAGE SIGNALS</i>	
W1. Package Usage [56]	
a. Number of downloads/stars/forks	Weights the potential contributor’s ownership of useful projects to the ecosystem.
W2. Community Tenure [51]	
a. Length of time contributing to project	Weights how much time user has been a contributor to other projects.
b. Length of time owning account	Weights how long user has actually owned an account
c. Contributory Strength	Weights the strength of contributions to other projects by lines of code, issues, pull requests
W3. Centrality Score [18]	
a. Connections to other Actors [18]	Weights a potential contributor’s network of interactions/contributory activities.
b. Time to form connections	Length of time taken to form the connections in ‘a’ above.