Tracing Vulnerability Propagation
Across Open Source Software Ecosystems

Jukka Ruohonen[0000−0001−5147−3084] and Qusai Ramadan[0000−0001−8159−918X]

{juk, qura}@mmmi.sdu.dk

University of Southern Denmark, Sønderborg, Denmark

Abstract. The paper presents a traceability analysis of how over 84 thousand vulnerabilities have propagated across 28 open source software ecosystems. According to the results, the propagation sequences have been complex in general, although GitHub, Debian, and Ubuntu stand out. Furthermore, the associated propagation delays have been lengthy, and these do not correlate well with the number of ecosystems involved in the associated sequences. Nor does the presence or absence of particularly ecosystems in the sequences yield clear, interpretable patterns. With these results, the paper contributes to the overlapping knowledge bases about software ecosystems, traceability, and vulnerabilities.

Keywords: software ecosystems, vulnerabilities, traceability, process mining

1 Introduction

The paper continues and advances the empirical vulnerability coordination research theme [7, 10] in a software ecosystem context; the interest is to better understand how vulnerabilities propagate across ecosystems. The ecosystem context places the paper into a large branch that has examined the software and cyber security of particularly programming language ecosystems, including with respect to a recent surge of malware uploads to these ecosystems [11]. With respect to software traceability research, which too is a large branch, the paper operates in a *post-release* traceability context because all software observed has already been released [4, 9]. Patching and otherwise handling of vulnerabilities is a classical example of typical post-release software engineering activities [6, 15], and, hence, it is not a surprise that also vulnerabilities have been traced [1, 14].

The paper's practical relevance can be motivated by the Cyber Resilience Act (CRA) recently agreed upon in the European Union.[1] Among the regulation's essential cyber security requirements are legal obligations to follow a coordinated vulnerability disclosure policy, supply security patches, and to only ship products without known vulnerabilities. As open source software is widely used also in the commercial software industry, fulfilling the obligations requires tracking, coordinating, and tracing of vulnerabilities also with respect to open source software components, including those distributed through software ecosystems.

---

[1] Regulation (EU) 2024/2847.

2 Data and Methods

2.1 Data

The dataset examined is based on the new Open Source Vulnerabilities (OSV) database.[2] It has been used in recent research [11, 15], and it is ideal for the paper's purposes because it provides cross-ecosystem traceability data. The dataset was limited to vulnerabilities (such that malware entries were omitted) archived with Common Vulnerabilities and Exposures (CVEs). Regarding the ecosystems, some of which use custom identifiers mapped to CVEs, all unique CVEs were included from the `aliases`, `upstream`, and `related` fields in the OSV's JavaScript Object Notation (JSON) schema. Given data retrieval in 19 April 2025, the dataset contains $n = 84,520$ vulnerabilities and 28 ecosystems.

A note should be also made about the ecosystem term; it refers to the corresponding concept used in the OSV database.[3] These ecosystems cover programming language ecosystems, such as PyPI for Python or Packagist for PHP, Linux distributions, such as Red Hat and Ubuntu, software testing frameworks, such as OSS-Fuzz, and large hosting services, such as GitHub for which GitHub Actions were merged with the main GitHub ecosystem entries. This terminological choice can be justified on the grounds that there is no consensus in the literature about definitions [3]. The choice also maintains coherence with the OSV.

2.2 Methods

In overall, the methodology adopted is based on process mining [5, 12]. Accordingly, the propagation of CVEs across software ecosystems are modeled as an event log. An event is understood as a tuple:

$$e = (c, a, t) \in \mathcal{C} \times \mathcal{A} \times \mathbb{T}, \tag{1}$$

where $c \in \mathcal{C}$ is a unique CVE in a set of all CVEs, $|\mathcal{C}| = n$, $a \in \mathcal{A}$ is a unique ecosystem in a set of all ecosystems observed, $|\mathcal{A}| = 28$, and $\mathbb{T} \subseteq \mathbb{N}$ is a set of discrete timestamps (measured in days). In terms of years, the timestamps start from 2000 and end to mid-April 2025. Then, an event log $L \subseteq \mathcal{C} \times \mathcal{A} \times \mathbb{T}$ is a multiset of events. For any $c \in \mathcal{C}$, the trace for the given c is defined as:

$$\sigma_c = \langle (a_1, t_1), (a_2, t_2), \ldots, (a_m, t_m) \rangle, \quad \text{such that} \quad t_1 \leq t_2 \leq \ldots \leq t_m. \tag{2}$$

The trace in (2) represents an ordered sequence through which a given CVE appeared across unique ecosystems. If a length of a given sequence is one, the given CVE-referenced vulnerability $c \in \mathcal{C}$ appeared only in a single ecosystem.

Two assumptions are made:

---

[2] https://osv.dev/
[3] https://google.github.io/osv.dev/data/#covered-ecosystems

1. *Independence of CVEs*: each CVE, $c \in \mathcal{C}$, is modeled as an independent case. There are no causal dependencies between different CVEs. Formally, for any two CVEs, $c_1, c_2 \in \mathcal{C}$ with $c_1 \neq c_2$, the traces σ_{c_1} and σ_{c_2} are assumed independent. Furthermore, the propagation of CVEs through ecosystems is treated as a Poisson process over discrete time, implying that time differences between events are realizations of a Poisson-distributed random variable.

2. *Uniqueness of ecosystem observations*: for each $c \in \mathcal{C}$ and each ecosystem $a \in \mathcal{A}$, there exists at most one recorded event for the appearance of c given a. In the presence of multiple observations, only the event with the earliest timestamp is considered. Formally, for duplicated events (c, a, t_i) and (c, a, t_j) with $t_i \neq t_j$, only the event with $\min(t_i, t_j)$ is retained.

The interest is to empirically observe partial ordering over ecosystems based on timestamp comparisons with a trace:

$$a_i \preceq a_j \iff t_i \leq t_j \quad \text{and} \quad |\sigma_c| > 1. \tag{3}$$

This ordering allows observing symbolic representations of propagation sequences across ecosystems. For instance:

$$\text{GitHub} \preceq \text{Debian} \preceq \text{Ubuntu}, \tag{4}$$

which means that a given CVE was observed in GitHub first, then in Debian, and then in Ubuntu, respecting the timestamps observed.

Already observing sequences such as (4) visually with descriptive statistics provides valuable insights into the propagation of CVE-referenced vulnerabilities across the twenty-eight ecosystems tracked in the OSV database. In addition, the interest is to observe traceability delays, as defined by:

$$f(c) = t_{\text{last}} - t_{\text{first}}, \tag{5}$$

where t_{first} and t_{last} are the first and last timestamps in a trace σ_c longer than $|\sigma_c| > 1$. The function measures how long it took for a $c \in \mathcal{C}$ to spread from its first known appearance to its last known appearance across all ecosystems.

The function also allows to formalize two *hypotheses* (H). First (H$_1$), it seems sensible to assume that $\text{Cor}(f(c), |\sigma_c|) > 0$ for the $m = |\sigma_c| > 1$ sequences, meaning that the more there are ecosystems, the longer the traceability delays. Second (H$_2$), it can be hypothesized that $f(c)$ might vary according to some particular ecosystems that appear in the $|\sigma_c| > 1$ sequences. For instance, the presence of some particular ecosystem might shorten the traceability delays, whereas the appearance of some other ecosystem might indicate a bottleneck in terms of delays. To examine this H$_2$, for each $a \in \mathcal{A}$, a t-test (with a correction for unequal variances; [13]) is computed for testing whether the presence of a given a in a σ_c affects the corresponding $f(c)$. Due to repeated testing, Bonferroni correction is applied, meaning that the statistical significance level is set to $0.05 \, / \, |\mathcal{A}|$. Although much debated, the Bonferroni correction seems suitable because many tests are carried out, H$_2$ is fairly loose in theoretical terms, and the purpose is to check whether any ecosystem shows a statistically significant relationship [2]. With these elaborations, the results can be disseminated next.

3 Results

The propagation sequences elaborated in (3) and (4) provide a good way to start the dissemination of the empirical results. Thus, the top-30 sequences (in terms of frequency) are shown in Fig. 1. (For presentation purposes, the figure shows also cases with $|\sigma_c| = 1$.) In total, a little over four thousand unique sequences were identified for the about 85 thousand unique CVEs. This amount alone testifies that the propagation can be quite complex among the 28 ecosystems sampled. Both the mean and median are six ecosystems per a CVE on average. However, GitHub alone leads the ranking in Fig. 1; nearly 14% of the CVEs observed were reported only in GitHub and nowhere else. GitHub is also present in many traces with a $|\sigma_c| = 2$, meaning that many other ecosystems, including both programming language ecosystems and Linux distributions, have tended to pick CVE-referenced vulnerabilities that were initially reported on GitHub.

The second place in the ranking is taken by CVEs first reported in Debian and then in Ubuntu. This observation is logical because most of Ubuntu's packages are distributed also in Debian, and largely also maintained therein. Having said that, there are also quite a few CVEs that were reported either in Debian or Ubuntu alone. Although further validation would be required, this observation could be taken to support an argument that there might be still room for better coordination and synchronization. The same point extends toward other Linux distributions who too seem to often pick CVEs that were first reported in Debian. In this regard, it also worth emphasizing that Debian, Ubuntu, Red Hat, and SUSE, among a few others, such as the Python Software Foundation who maintains PyPI, are official CVE numbering authorities (CNAs) unlike Alpine Linux, Mageia, Rocky Linux, and some others [8]. Because the dataset's scope was restricted to CVE-referenced vulnerabilities, and because CVEs require coordination of their own [10], the propagation sequences observed may be partially—but unlikely fully—explained by the CNAs present in the sequences.

Turning to H_1, Fig. 2 shows the traceability delays, as given by (5) for all sequences with lengths larger than one, against the number of ecosystems, m, present in these sequences. The hypothesis is rejected; the Pearson's product moment correlation coefficient is small in magnitude and has a negative sign. In other words—and unlike what was expected and what would seem intuitively logical, the traceability delays do not notably shorten or lengthen according to whether there are many or a few ecosystems present in the traces. Relatively long propagation sequences, such as those with five Linux distributions in Fig. 1, are not necessarily slower than shorter ones. In addition to this observation, Fig. 2 delivers an important point: the traceability delays have generally been long on average. The mean and median are as long as three and two years, respectively. As can be further seen from the figure, there are also a lot of outliers, including even extreme ones indicating delays over a decade. Although the reasons for such outliers are not well-known, similar observations have been made previously [10]. Regardless of the potential explanations, the observation reinforces the earlier remark about potential gains from better coordination and synchronization.
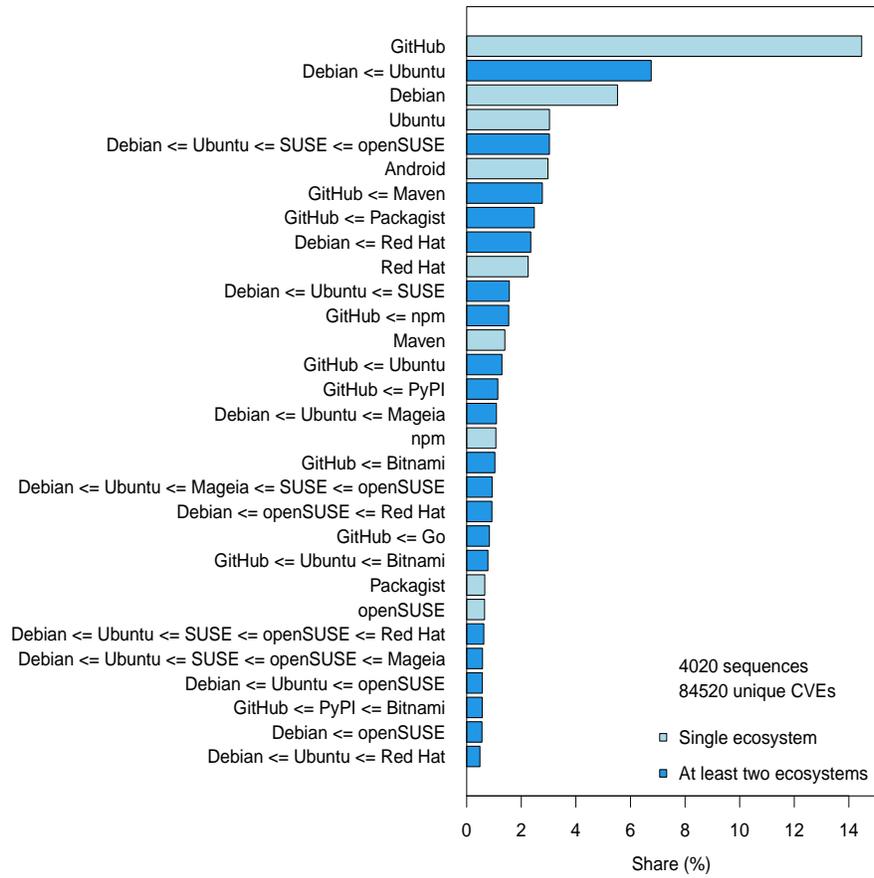
Fig. 1. The Top-30 Sequences Extracted
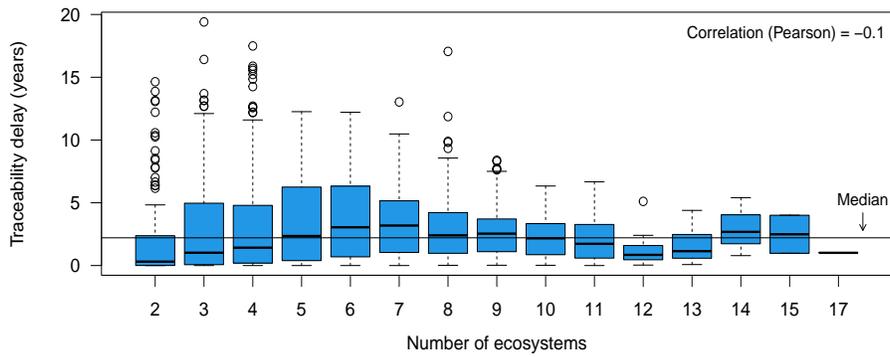
Fig. 2. Traceability Delays ($55,845$ unique CVEs for the $3,997$ sequences with $|\sigma_c| > 1$)

Regarding H_2, the Bonferroni-corrected and variance-adjusted t-tests indicate that for ten ecosystems out of 27 the differences between the means are not different from zero, meaning that a presence or an absence of a given a does not affect the averages of the given traceability delays.[4] Among these ten ecosystems are GitHub, Maven, openSUSE, Packagist, PyPI, RubyGems, and Ubuntu. In contrast, statistically significant differences are present for Debian, Mageia, Red Hat, and SUSE for which their presences indicate longer delays. If the propagation sequences have involved Go or npm, the delays have been shorter. In general, these test results indicate that there is no clear-cut pattern among the ecosystems with respect to the traceability delays, including regarding the CNAs.

4   Conclusion

This short paper presented an empirical traceability analysis of the propagation of CVEs across popular open source software ecosystems, including Linux distributions and programming language ecosystems. The analysis presented demonstrates the value offered by the new OSV database also for research purposes.

Regarding the results, (1) the propagation has generally been rather complex, as demonstrated by over four thousand unique propagation sequences for the 28 ecosystems and about 85 thousand unique CVEs observed. In terms of a frequency ranking, however, (2) GitHub alone, Debian and Ubuntu together, and Debian and Ubuntu alone lead the ranking, meaning that also many CVEs have only been reported in one ecosystem without a propagation to others. The frequent propagation from Debian to Ubuntu is also expected, given the close resemblance and collaboration between these two Linux distributions. Furthermore, (3) the traceability delays, as measured by time differences between the last and first appearances of CVEs in given ecosystems, do not correlate well with the number of ecosystems present in the corresponding propagation sequences. That is, longer (shorter) sequences do not imply lengthier (faster) delays. Although reporting was omitted for brevity, it can be noted that neither the severity of the CVE-referenced vulnerabilities, as measured by the Common Vulnerability Scoring System (v. 3.1), affects the traceability delays statistically.[5] In addition, (4) there is no clear—or at least easily interpretable—pattern in the traceability delays with respect to some particular ecosystems appearing in the sequences. Last but not least, (5) the traceability delays have been lengthy on average; the median is about two years. There are also extremely outlying traceability delays.

In light of the rather long delays, it could be perhaps contemplated whether the OSV database could (or should) be developed further into a more general platform for coordinating and otherwise handling of vulnerabilities in the open source software context [15]. Although much of development nowadays happens on GitHub, it remains unclear how well the platform works in facilitating coordination, especially when keeping in mind the large amounts of software distributed through the twenty-eight open source software ecosystems covered in the paper.

---

[4]   CRAN was excluded because only a single CVE has been reported for it.
[5]   https://www.first.org/cvss/v3-1/

References

[1] Alqahtani, S.S., Eghan, E.E., Rilling, J.: Recovering Semantic Traceability Links Between APIs and Security Vulnerabilities: An Ontological Modeling Approach. In: Proceedings of the IEEE International Conference on Software Testing, Verification and Validation (ICST 2017). pp. 80–91. IEEE, Tokyo

[2] Armstrong, R.A.: When to Use the Bonferroni Correction. Ophthalmic & Physiological Optics 34(5), 502–508 (2014)

[3] Burström, T., Lahti, T., Parida, V., Wartiovaara, M.: Software Ecosystems Now and in the Future: A Definition, Systematic Literature Review, and Integration Into the Business and Digital Ecosystem Literature. IEEE Transactions on Engineering Management 71, 12243–12258 (2024)

[4] Dakkak, A., Bosch, J., Olsson, H.H.: The Role of Post-Release Software Traceability in Release Engineering: A Software-Intensive Embedded Systems Case Study From the Telecommunications Domain. In: Proceedings of the 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2022). pp. 169–176. IEEE, Gran Canaria (2022)

[5] Ghahderijani, A.Y., Turgay, H.N., Karastoyanova, D.: Change Logging and Mining of Change Logs of Business Processes – A Literature Review (2025), archived manuscript, available online: https://arxiv.org/abs/2503.02804

[6] Jayatilaka, A., Zahedi, M., Babar, M.A.: Software Security Patch Management – A Systematic Literature Review of Challenges, Approaches, Tools and Practices. Information and Software Technology 144, 106771 (2022)

[7] Lin, J., Adams, B., Hassan, A.E.: On the Coordination of Vulnerability Fixes: An Empirical Study of Practices From 13 CVE Numbering Authorities. Empirical Software Engineering 28, 1–34 (2023)

[8] MITRE: CVE Numbering Authorities (CNAs) (2025), Available online in May 2025: https://www.cve.org/ProgramOrganization/CNAs

[9] Ruohonen, J., Leppänen, V.: How PHP Releases Are Adopted in the Wild? In: Proceedings of the 24th Asia-Pacific Software Engineering Conference (APSEC 2017). pp. 71–80. IEEE, Nanjing (2017)

[10] Ruohonen, J., Rauti, S., Hyrynsalmi, S., Leppänen, V.: A Case Study on Software Vulnerability Coordination. Information and Software Technology 103, 239–257 (2018)

[11] Ruohonen, J., Saddiqa, M.: A Time Series Analysis of Malware Uploads to Programming Language Ecosystems (2025), archived manuscript, available online: https://arxiv.org/abs/2504.15695

[12] van der Aalst, W.M.P.: Process Mining: A 360 Degree Overview. In: van der Aalst, W.M.P., Josep Carmona (eds.) Process Mining Handbook, pp. 3–34. Springer, Cham (2022)

[13] Welch, B.L.: The Generalization of 'Student's' Problem when Several Different Population Variances are Involved. Biometrika 34(1/2), 28–35 (1947)

[14] Yang-Smith, C., Abdellatif, A.: Tracing Vulnerabilities in Maven: A Study of CVE Lifecycles and Dependency Networks (2025), archived manuscript, available online: https://arxiv.org/abs/2502.04621

[15] Zhang, L., Wu, J., Liu, C., Li, K., Sun, X., Zhao, L., Wang, C., Liu, Y.: Fixing Outside the Box: Uncovering Tactics for Open-Source Security Issue Management (2025), archived manuscript, available online: https://arxiv.org/abs/2503.23357