Draft Security Analysis of Open-Source Software Package Ecosystems

Žiga Trček[a], **Matej Urbas**[a], **and Jan Vasiljević**[a]

[a]University of Ljubljana, Faculty of Computer and Information Science, Večna pot 113, SI-1000 Ljubljana, Slovenia

The manuscript was compiled on May 31, 2024

The use of open-source packages and libraries significantly accelerates software development but simultaneously introduces numerous security risks. Motivated by a recent near-compromise of OpenSSH by a malicious actor, this study aims to investigate the security of open-source software by conceptualizing it as a network and examining transitive vulnerabilities. Our analysis specifically focuses on PyPI, npm, and crates.io, which are the predominant package managers for Python, JavaScript, and Rust, respectively. Through this exploration, we seek to uncover potential security weaknesses within these ecosystems and propose methods to enhance their security posture.

Ni se koncano: Cakam da mamo dejansk neki narejeno. Bolj placeholder.

The date is March 28, 2024. A principal software engineer at Microsoft notices an unusual delay in his login attempts, timing at approximately 500ms—significantly longer than usual by his standards. This prompts an investigation into higher than normal CPU usage, during which he observes anomalous behavior in the SSH daemon process. This unexpected discovery leads to the identification of a vulnerability in the XZ Utils library utilized by OpenSSH. The breach, orchestrated by a malicious actor through a combination of social engineering, code injection, and obfuscation, was set into motion over several years. Interestingly, the core code of OpenSSH remained untouched; instead, a transitive dependency was exploited. Had this vulnerability remained undetected, it could have potentially compromised a vast number of servers relying on OpenSSH, embedding a remote code execution backdoor.

In a manner similar to the exploitation of the XZ Utils library, other open-source software packages and libraries are also vulnerable to malicious attacks. The development of software inherently involves placing trust in the authors of utilized libraries, who are often individual hobbyists or small teams with limited resources. The potential for significant damage is large if a malicious actor targets a lesser-maintained package that, while small, is widely used—either directly or as a transitive dependency in other software projects. The incident involving the left-pad package, which was removed from npm and consequently led to the failure of numerous dependent packages, serves as a reminder of the inherent fragility within the open-source ecosystem.

Int this project we look at three different pacakge ecosystems: PyPI, npm, and crates.io, which are package repositories for Python, JavaScript, and Rust, respectively. We aim to investigate the security of these ecosystems by conceptualizing them as networks.... Zej ne bom pisal dokler ne dejansko necesa nardimo.

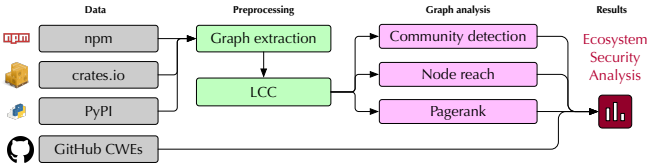

Fig. 1. Mandatory informative illustration highlighting main contributions.

Problem definition, contributions etc.

Related work

Robustness of dependency networks is a topic that has been examined in several studies. (1) explored the robustness of the npm package dependency network and highlighted its vulnerability to targeted attacks by identifying crucial nodes. Attacks on such nodes could gravely affect a large portion of the network, as they are responsible for a significant number of dependencies. Their findings suggest that while the network has crucial nodes, it is not highly vulnerable as they are mostly large and well-maintained projects. Moreover, the network is trending towards a more robust state, as the number of dependencies is decreasing over time.

(2) published a comparative analysis of dependency networks across 7 package ecosystems, including Cargo, CPAN, CRAN, npm, NuGet, Packagist, and RubyGems. They proposed metrics to capture growth, changeability, reusability, and fragility. They revealed that while the ecosystems are growing, a minority of packages are responsible for most updates and dependencies. Furthermore, it is suggested that even transitive dependencies on unmaintained or obsolete packages can have a detrimental effect on the security and maintainability of the ecosystem.

(3) carried out a study on the accessibility of GitHub repositories for npm and PyPI libraries to understand the level of maintenance for them. Their research showed that a significant portion of libraries lacked valid repository URLs, which hinders the ability to monitor vulnerabilities and maintain codebases. An emphasis on the importance of maintaining valid repository URLs was made, as it is crucial for the sustainability of the open-source ecosystem.

Community detection has been proposed as a security analysis tool. The found communities can provide great insight into the structure of the network and the relationships between packages, which helps assess the robustness of the network. Understanding what packages are closely related and how they

All authors contributed equally to this work.

[1]To whom correspondence should be addressed. E-mail: fine.author@email.com.

are connected can help identify potential vulnerabilities and dependencies that could be exploited by attackers (1, 3).

(4) examine the dependency graph of R packages and the relationship between various metrics. They find that centrality measures have a high correlation with the number of downloads and citations of a package. Furthermore, package attributes such as number of authors and commits also have a positive impact on the number of downloads and citations.

Security vulnerability data analysis has been a large area of research in the recent years. (5) studied around 400 security reports from the npm network to understand how they are discovered and fixed, and how they affect the network. They found that it takes around a year for a discovered vulnerability to be published publically, while it takes around 2 years after the discovery for the vulnerability to be fixed. (6) study vulnerability life cycles on a large software vulnerability dataset, getting similar results.

(7) propose and evaluate novel machine learning approaches to vulnerability prediction in open-source software. Some simple vulnerabilities are easily detected by their approach, but even more complex vulnerabilities can be detected with high accuracy.

(8) use call graphs instead of dependency graphs to detect dependencies in software. They find that call graphs are efficient in aiding preliminary evaluation of security issues and their impact to other applications. Furthermore, (9) find that Cargo packages call only 40% of their dependencies, which can lead to many unneeded security vulnerabilities.

(10) examine around 200000 PyPI packages using static analysis and find that around 46% of the packages have at least one security vulnerability.

Results

We analysed three popular programming languages' ecosystems: npm (11), Crates.io (12) and PyPI (13). Basic stats of each ecosystem's dependency network are shown in Table 1. NPM dependency network is analysed twice, once with combined production and development dependencies, and once with only production dependencies.

Table 1. Original dependency networks for all three ecosystems.

ecosystem	n	m	$\langle k \rangle$	$\langle C \rangle$	#CC
npm	2425767	23006281	19	0.18	561569
npm (prod)	2424918	7920337	7	0.06	870151
PyPI	532386	1649201	6	0.11	179304
Crates.io	145269	830337	11	0.23	25179

For the rest of our analysis, we only considered the largest weakly connected components. Statistics of these networks are shown in Table 2.

Table 2. Largest connected components for all three ecosystems.

ecosystem	n	m	$\langle k \rangle$	$\langle C \rangle$	% nodes
npm	1845838	22979692	24	0.23	76
npm (prod)	1518107	7871828	10	0.10	62
PyPI	349793	1645588	9	0.17	66
Crates.io	119461	829615	13	0.28	82

By enumerating over vulnerabilities from the GitHub's advisory database we examined the reach of every listed vulnerability. The vulnerabilities were analysed as if they all occured on the most recent package dependency graphs as we didn't have the required graph history snapshots. The reach is defined by the size of the affected package's in-component - all the packages that directly or indirectly depend on the affected package. To get a measure, which is proportional to the graph size and package importance, we summed up the page rank scores from the affected component. The reach measure for every vulnerability for these ecosystems is shown in Figure 2.

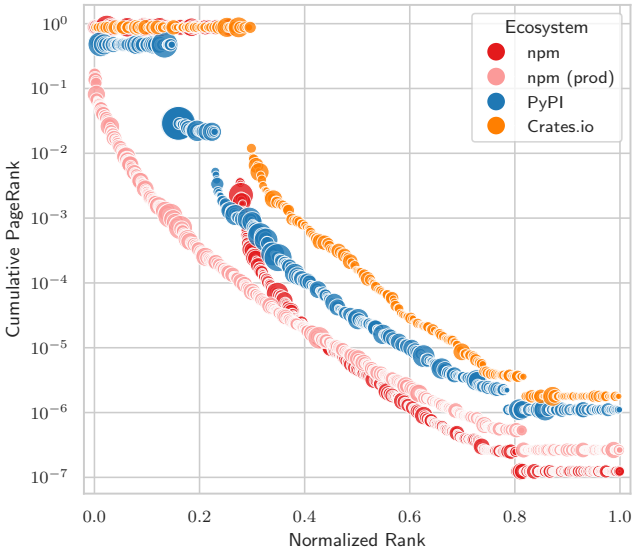

Fig. 2. Vulnerabilities and their reach. Each dot represents a single vulnerability. Dot size represents the vulnerability severity according to CVSS.

Crates.io and npm ecosystems are the most vulnerable. Around 30% of all vulnerabilities could reach almost every package (95% of all). PyPI is a bit less interwined, with *only* 69% of packages being affected by the largest vulnerability. If we exclude all development dependencies from the npm network, we notice a very high decrease in the percentage of affeceted packages. The npm production network consists of around 3 times less edges than the network which also includes development dependencies. This might correlate to 3 times the reduction in affected packages. This also shows that developers are much more exposed than production environemtns, which keep their dependencies to the minimum. Vulnerabilities with biggest reach for every ecosystem are listed in Table 3.

Table 3. Biggest reach vulnerability.

ecosystem	package	% packages affected	affected pagerant
npm	lodash	93	0.88
npm (prod)	ms	**30**	**0.18**
PyPI	pyOpenSSL	69	0.54
Crates.io	shlex	95	0.88

Discussion

Summary of results, main contributions, final conclusions, future work etc. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id,

vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Methods

Verjetno manjka se ki

Data Acquisition. The data for `crates.io` was acquired from their official website, where daily data dumps are available. For `npm`, we initially attempted to follow the official guide on their website to clone the CouchDB database. However, this method proved to be outdated and non-functional. Consequently, we utilized data from (14), which provides a weekly data dump of all npm dependencies. The entire dataset was 200GB in size and packaged in a PostgreSQL database. For `PyPI`, we had to manually scrape the data, as there is no official data dump available.

Data Preprocessing. From the data we acquired, we extracted package names, dependencies, and download counts. We then constructed a simple directed graph where nodes represent packages and edges represent dependencies. The edge direction is from the dependent package to the dependency. Graphs were then reduced to the largest connected component and exported in the `GraphML` format for further analysis.

`igraph`. Instead of relying on `networkx` for network analysis, we used the `igraph` library. Using `networkx`, even a simple import of the `npm` dataset took around 300 seconds. Additionally, every other operation was significantly slower compared to `igraph`.

Community Detection. For community detection, we used the `leiden` algorithm, which is an efficient method for uncovering community structures in large networks. The algorithm optimizes the modularity function Q as $Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j)$ where A_{ij} is the weight of the edge between nodes i and j, k_i and k_j are the degrees of nodes i and j, m is the total weight of all edges in the graph, and $\delta(c_i, c_j)$ is the Kronecker delta which is 1 if nodes i and j belong to the same community and 0 otherwise.

Vulnerability analysis.

1. Andrej Hafner, Anže Mur, and Jaka Bernard. Node package manager's dependency network robustness. *arXiv preprint arXiv:2110.11695*, 2021.
2. Alexandre Decan, Tom Mens, and Philippe Grosjean. An empirical comparison of dependency network evolution in seven software packaging ecosystems. *Empirical Software Engineering*, 23(3):1–37, 2018.
3. Alexandros Tsakpinis and Alexander Pretschner. Analyzing the accessibility of github repositories for pypi and npm libraries. In *28th International Conference on Evaluation and Assessment in Software Engineering (EASE 2024)*. ACM, 2024.
4. Gizem Korkmaz, Claire Kelling, Carol Robbins, and Sallie A. Keller. Modeling the impact of r packages using dependency and contributor networks. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 511–514, 2018. .
5. Alexandre Decan, Tom Mens, and Eleni Constantinou. On the impact of security vulnerabilities in the npm package dependency network. In *Proceedings of the 15th International Conference on Mining Software Repositories*, pages 181–191. ACM, 2018.
6. Muhammad Shahzad, Muhammad Zubair Shafiq, and Alex X. Liu. A large scale exploratory analysis of software vulnerability life cycles. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 771–781, 2012. .
7. Hazim Hanif, Mohd Hairul Nizam Md Nasir, Mohd Faizal Ab Razak, Ahmad Firdaus, and Nor Badrul Anuar. The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches. *Journal of Network and Computer Applications*, 179:103009, 2021. ISSN 1084-8045. . URL https://www.sciencedirect.com/science/article/pii/S1084804521000369.
8. Joseph Hejderup, Arie van Deursen, and Georgios Gousios. Software ecosystem call graph for dependency management. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER '18, page 101–104, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356626. . URL https://doi.org/10.1145/3183399.3183417.
9. Jason Hejderup, Moritz Beller, Konstantinos Triantafyllou, et al. Präzi: from package-based to call-based dependency networks. *Empirical Software Engineering*, 27(102):1–34, 2022. .
10. Jukka Ruohonen, Kalle Hjerppe, and Kalle Rindell. A large-scale security-oriented static analysis of python packages in pypi. In *2021 18th International Conference on Privacy, Security and Trust (PST)*, pages 1–10, 2021. .
11. Node package manager, 2024. URL https://www.npmjs.com/.
12. Crates.io, 2024. URL https://crates.io/.
13. Python package index, 2024. URL https://pypi.org/.
14. Donald Pinckney, Federico Cassano, Arjun Guha, and Jonathan Bell. A large scale analysis of semantic versioning in npm. In *Proceedings of the 20th International Conference on Mining Software Repositories*, MSR, 2023. URL https://www.jonbell.net/preprint/msr23-npm.pdf. Acceptance rate: 36%.