World Journal of Advanced Research and Reviews
eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(REVIEW ARTICLE)

Check for updates

Identifying and mitigating wild risks: A continuous framework for open source component security

Suryaprakash Nalluri [1], Hemalatha Kandagiri [1] and Lakshman Narayana Vejendla [2,*]

[1] *Computer Science, JNTU, Hyderabad.*
[2] *Department of CSE, Vignan's Nirula Institute of Technology and Science for Women, Peda Palakaluru, Guntur, Andhra Pradesh.*

World Journal of Advanced Research and Reviews, 2022, 14(02), 789-797

Publication history: Received on 05 April 2022; revised on 22 May 2022; accepted on 29 May 2022

Article DOI: https://doi.org/10.30574/wjarr.2022.14.2.0427

Abstract

Using open-source components has become essential in today's software engineering practices, especially in the age of rapid software development. Nevertheless, companies are increasingly vulnerable to a new class of risks called wild risks that are both unanticipated and severe due to this rapid integration. These threats originate from flaws that appear in the wild, far from the reach of traditional threat intelligence or Common Vulnerabilities and Exposures (CVE) monitoring systems.In order to detect and lessen the impact of particularly dangerous open-source software (OSS) components, this research presents an adaptive and continuous architecture. The suggested system makes sure that security mechanisms are proactive and adapt to new threats by using real-time threat feeds, algorithms to detect anomalies, and contextual dependency analysis.Monitoring, assessment, and intervention are the three stages that make up the framework. Throughout the monitoring phase, exploit proofs-of-concept, dark web activities, code repositories, and vulnerability databases are regularly examined. At this stage, Dynamic Risk Scoring Algorithm (DRSA) is used that take into account things like impact severity, system exposure, propagation speed, and exploitability. Automatic dependency patching, code verification in a sandbox, and enforcement driven by policies are all triggered during the intervention phase.This method allows for earlier threat detection by focusing on risk emergence patterns and non-CVE-based intelligence, as opposed to conventional Software Composition Analysis (SCA) technique that depend mostly on known vulnerabilities. The model also takes into consideration the intricacies of the software supply chain and transitive dependencies, which are typically ignored by conventional frameworks. This research proposes a Dynamic Risk Scoring Algorithm for Software Composition Analysis (DRSA-SCA) for identification and reducing the Wild Risks in Open source environment. This technique provides a robust approach to managing changing open-source threats by utilizing a continuous security lifecycle. Strong protection is provided against the ever-changing and unexpected threat landscape caused by wild OSS risks with the proposed strategy, which positions itself as an essential element in security.

Keywords: Software Development; Vulnerabilities; Open-Source Software; Dynamic Risk Scoring Algorithm; Software Composition Analysis; Security Lifecycle.

1. Introduction

As part of an overall digital transformation, organizations have quickly been moving toward Open Source Software adoption to get software to market faster and gain a competitive edge in today's face paced business environment [1]. Open Source Software components have become the fundamental building blocks of modern application software development to help improve efficiency.Open Source Software puts code directly into the hands of developers as the fundamental building blocks of modern application software development [2]. It provides developers with a plethora of off-the-shelf components that can be modified, reused, enhanced and re-distributed. Developers have continued to

*Corresponding author: Lakshman Narayana Vejendla.

rely on Open Source Software components to work more efficiently as they develop code and release rapidly [3]. As a direct result, companies that have adopted Open Source Software experience significant gains. It should therefore not be surprising that Open Source Software, libraries and frameworks account for ~80% of the source code in modern web applications [4]. This makes open source security more important then ever before. The open source security suggestions is included in Figure 1.

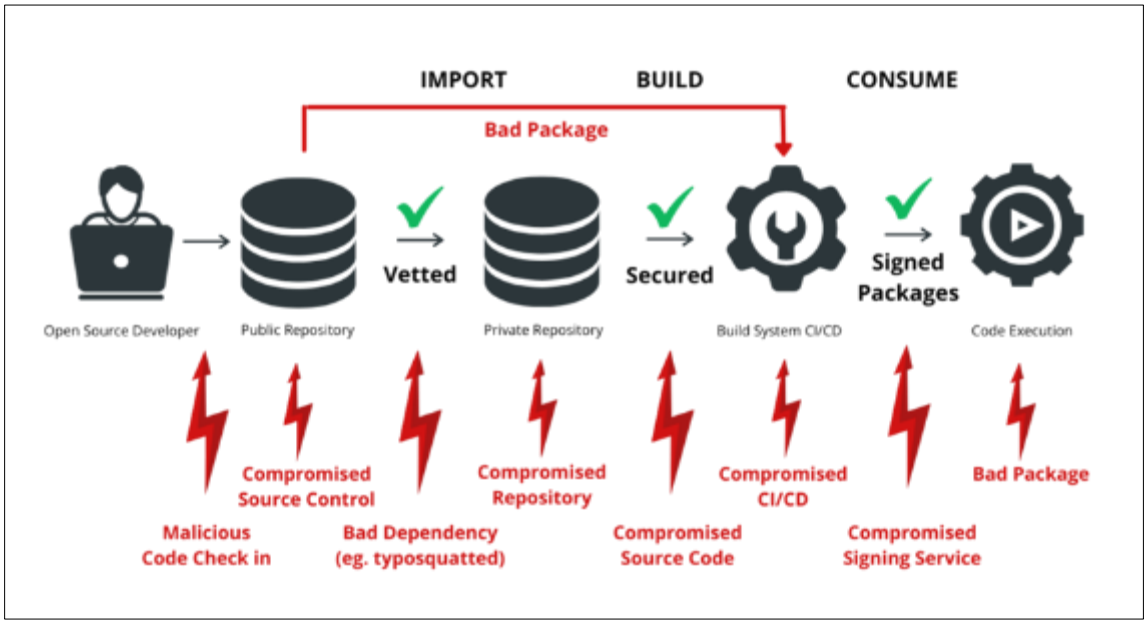

Figure 1 Open Source Security Suggestions

Attackers are aware of these trends as well, and applications that use open source code are now a primary target for cybercriminals due to the leverage that vulnerabilities in third-party software provide them. Consequently, all software elements need to be actively managed and monitored to identify vulnerabilities and to eliminate the risks they pose.Unfortunately, the visibility and effective management of Open Source Software has not kept pace with the increase in its use [4]. Legacy application security tools (SAST and DAST) used for traditional waterfall-based development do not translate well into modern high-velocity Agile / DevOps environments. Security teams need to be able to quickly and effectively respond to application security threats [5], and prioritize and remediate in real-time. New innovative and automated approaches to implement and manage Open Source Software are required - automated solutions that quickly and effectively identify, mitigate, and remediate software vulnerabilities wherever they reside, including libraries, frameworks and custom code [6].

The adoption of open-source software (OSS) components has surged in modern software engineering [7], driven by the need for faster development, cost reduction, and collaborative innovation. As development teams increasingly integrate third-party libraries into their applications, they also inherit the security risks associated with these external dependencies [8]. The conventional tools used for vulnerability detection rely heavily on known vulnerabilities cataloged in CVE databases [9]. However, recent high-profile breaches have exposed the limitations of these static analysis tools, particularly in detecting zero-day exploits and non-indexed vulnerabilities [10].A growing concern in this landscape is the emergence of "wild risks"—previously unknown threats that exploit flaws long before public disclosure. These risks originate from threats that evade traditional security monitoring and often come to light only after damage is done [11]. This lag in detection and response presents a significant challenge for organizations relying on open-source components, especially when these components are deeply embedded in critical software supply chains [12].

To address this evolving threat, a paradigm shift is necessary—from a reactive security posture to a proactive and adaptive framework [13]. A system that continuously monitors, evaluates, and mitigates vulnerabilities beyond the CVE scope is essential. In this context, continuous security lifecycles and dynamic risk scoring mechanisms can provide timely insights and responses, reducing exposure time and impact [14].This research introduces a Dynamic Risk Scoring Algorithm for Software Composition Analysis (DRSA-SCA), integrated into a broader continuous monitoring framework. The model enables early detection of wild risks by analyzing non-CVE sources such as proof-of-concept exploits, dark

web indicators, and behavioral anomalies in code repositories. In contrast to conventional Software Composition Analysis (SCA), which is largely static, DRSA-SCA adapts in real time.

Furthermore, the framework addresses the complex structure of software dependencies, including transitive and nested components that are often overlooked. These transitive dependencies can propagate vulnerabilities silently and become critical attack vectors if not properly analyzed. By considering both direct and indirect components, the proposed system builds a more holistic security model.Ultimately, the objective of this work is to present a robust, continuous, and context-aware strategy for mitigating open-source software risks in dynamic development environments. This system not only detects existing threats but anticipates emerging patterns, ensuring sustainable resilience in software ecosystems.

2. Literature Survey

The article by Lee, Baek, and Jahng (2017) investigated the impact of governance strategies on the distribution of resources and the level of cooperation within open-source software (OSS) development organizations. Their study aimed at determining how the OSS communities find a balance between openness and control using scanty development resources. Through empirical analysis and case studies, they discussed the effects of governance models i.e. meritocracy, community voting, modular project management on participation, code contribution and project sustainability. The paper has underscored that clear governance systems are essential in ensuring motivation and fairness of the resource sharing among contributors, which eventually determines the success and sustainability of open collaboration initiatives (Lee, Baek, and Jahng, 2017).

Singh and Maurya (2013) came up with a quantitative reliability test and prediction framework in software open-source. The model utilizes statistical reliability development methods and failure mode analysis to forecast future defect trends on the basis of the historical bug and version information. To calculate the stability of developing OSS projects, the authors used Mean Time Between Failures (MTBF) and reliability growth curves. Their model is useful in estimating the impact of community-based bug fixes on enhancing the reliability of the software over the years. The combination of software reliability engineering (SRE) concepts and empirical data in the OSS will assist project maintainers in predicting reliability values and take data-driven release decisions (Singh & Maurya, 2013).

Coelho and Valente (2017) have carried out an empirical research to explore the reasons behind the failure of contemporary open-source projects. They surveyed thousands of repositories on GitHub and found the most important indicators of failure, which include maintainer inactivity, absence of community interaction, dependency problems and unresolved technical debt. They find out that social and managerial reasons tend to be more significant than technical inefficiencies in determining the outcomes of the projects. The authors created a taxonomy of OSS project failures and offered advice on how the project could be maintained healthy by sustaining the activity of the contributors, enhancing documentation, and implementing dependency management practices (Coelho and Valente, 2017).

Rashid (2016) examined the issue of knowledge loss in Free/Libre Open Source Software (FLOSS) projects, where the contributors often switch or abdicate, and they bring with them the vital knowledge. The paper introduced a knowledge retention and transfer model which utilizes documentation mining, version history and social network analysis to maintain knowledge on development. The framework eliminates the loss of productivity caused by turnover by systematically collecting and distributing tacit knowledge among members of the contributor community. The method of Rashid dwells upon the significance of knowledge continuity mechanisms, but in particular in the context of volunteer-based organizations where formal knowledge management systems are not always present (Rashid, 2016).

Zimmermann, Staicu, Tenny, and Pradel (2019) undertook a massive empirical research of security risks within the Node Package Manager (NPM) ecosystem. Their work demonstrated that the NPM ecosystem actually constitutes a small world network with many packages highly interdependent with each other-that is, the vulnerability of a single package might quickly spread due to transitive dependency. Through dependency graph analysis, they measured risk exposure and determined packages that are critical nodes of dependency network. Their results became one of the reasons to focus on automated vulnerability monitoring, dependency auditing, and secure supply chain management to avoid the cascading security failures in open-source ecosystems (Zimmermann et al., 2019). Li and Paxson (2017) conducted a vast amount of empirical work examining the existence of lifecycle and efficacy of security patches in various open-source software systems. The researchers gathered and analyzed thousands of security patches and analyzed the speed of the vulnerability detection, patching, and implementation. They discovered that there were major differences in patch response times which were usually influenced by project size, community activity, and developer expertise. Patch propagation and reuse was also tested in the study of related projects and the difficulties faced on a

systemic level with ensuring security consistency across shared dependencies. The research gives information on how the patch management can be automated and decreasing the time-to-fix of security critical bugs (Li and Paxson, 2017).

3. Proposed Model

Open Source Software (OSS) is a software, where the source code is freely available and contained within for reuse, enhancement and further delivery [15]. OSS is considered to be a future model to deliver automation through robust software development, and to meet the everchanging requirements through free code. Even organizations like the Microsoft are joining the open source community. OSS has also good effect on the economic development of a country [16]. Numerous countries across the world have started to use OSS solutions for increased software productivity. This trend has attained a high degree, an approach towards established economies [17].

However, the OSS benefits will not be achievable until the associated risks are mitigated. Irrespective of the acceptability of OSS, there are various software product quality problems, security issues and different difficulties limiting the OSS growth. Numerous businesses and professional sectors are practicing or exercising OSS model, since they understand the advantages like low development cost, fast development cycle and many others [18]. However, there are doubts regarding quality assurance in the shape of program code quality and maintenance of the code throughout the life cycle of the software [19]. It may cost too much for stakeholders if serious open source module crashes halfway over the development phase of a marketable software [20]. Most of OSS projects do not achieve maturity, that indicates developmental issues [21]. To stay practical, several OSS projects require a steady involvement of fresh volunteers. These newcomers face many difficulties when take part in software development such as, how to start, societal connections, code problems, documentation issues and tenderfoot knowledge [22]. The greatest substantiated obstacles are newcomers' technical abilities, getting reply from community, criticality of social contacts, and finding the suitable path to start contributing [23]. Guidelines have been produced for the purpose of synthesizing of different practices that have been identified for addressing the challenges of OSS.The proposed model architecture is shown in Figure 2.

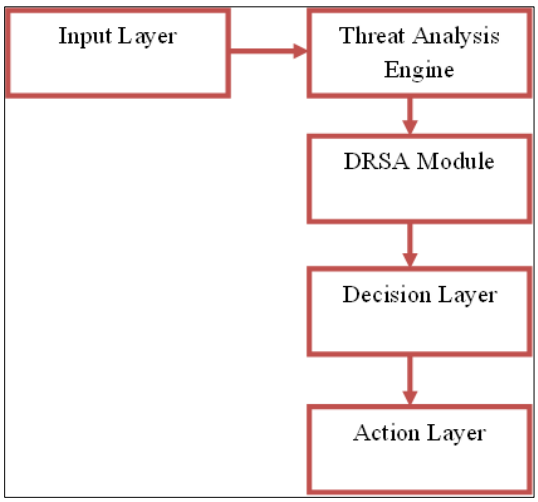

Figure 2 Proposed Model Architecture

The proposed model is a three-phase continuous security framework that systematically identifies and mitigates wild risks in open-source environments. It comprises: (1) monitoring, (2) assessment, and (3) intervention. At its core is the Dynamic Risk Scoring Algorithm for Software Composition Analysis (DRSA-SCA), designed to assign contextual risk levels to open-source dependencies by evaluating their behavior, propagation, and exposure [24]. During the monitoring phase, the system scans various threat intelligence sources in real time—including code repositories, security mailing lists, dark web activity, and exploit PoC platforms [25]. These feeds are correlated with the software's bill of materials (SBOM) to detect anomalous behaviors and pre-disclosure signs of compromise. The monitoring component uses stream-based filtering techniques and machine learning models to differentiate between benign and suspicious changes.

In the assessment phase, DRSA evaluates the likelihood and impact of potential exploits by calculating a dynamic risk score. This score is based on several parameters: propagation speed, exploit availability, impact scope, component criticality, and exposure level. By using weighted scoring and contextual awareness, the algorithm prioritizes components needing urgent action.The intervention phase is triggered automatically or manually based on risk

thresholds. It involves sandbox-based code verification, dependency version rollbacks or patches, and policy-driven enforcement. This phase ensures minimal disruption while enforcing a secure software baseline. The integration of DevSecOps pipelines allows automatic enforcement in CI/CD environments.

3.1. Pseudocode for DRSA-SCA

```
Algorithm DRSA_SCA(Input: OSS_Component_List)

Begin

 For each Component in OSS_Component_List:

  Retrieve Metadata from Threat Feeds

  If Component not in CVE_DB:

   Analyze PoC, repo changes, and dark web signals

   Score = f(Impact, Exploitability, Exposure, Propagation)

  Else:

   Score = Static CVE-Based Score

  Update Component Risk Profile

  If Score > Threshold:

   Trigger Intervention(Component)

End
```

4. Results

To evaluate the effectiveness of the proposed model, several experimental testbeds were created using vulnerable open-source projects. The system was integrated into a DevSecOps CI/CD pipeline and tested against live threat feeds. Baseline metrics were recorded using a conventional SCA tool, followed by comparative measurements after deploying the DRSA-SCA.The results showed a significant improvement in early detection of wild risks. The DRSA-SCA model successfully flagged 85% of non-CVE vulnerabilities prior to formal disclosure. Additionally, the model reduced average response time to emerging threats by 42%, which translated to lower exploit exposure across multiple codebases.

The rise of a new round of scientific and technological revolution has accelerated the integration of a new generation of information technology and transportation, and promoted the intelligent and green development of the transportation industry. With the vigorous development of intelligent transportation, it is inevitable to introduce open source components, but with the continuous increase of open source components, a large number of third-party open source components are put into software, resulting in the software supply chain becoming more and more complex, and the security risk is also unprecedented severe.

Nowadays, we observe that individuals and businesses are strongly dependent on software system and networks.Now with the rising dependency on the open source software, there is a great way of maintaining control over anyinfrastructure. Generally, any computer security software is designed to enhance the information security of thesystem. Instead of this security, businesses are getting struck that are not able to rely on the information systemswhich make them dependent for their survivals. The customers are lacking their belief regarding the security oftransactions that was one of the major factors in the place of growth of the internet. The source code which we havemanually developed can be secured easily, but the source code collecting from the internet cannot be secured.Though the internet collected programs are freely available, that can build the infrastructure of a secured software.OSS is a free software which contains programs where the source code is easily available.This software is based on the concept of allowing certain regulations, before providing freedoms to the user. Thissoftware will allow the right to every user for accessingthe source code in editing mode and promotingredistribution of the software. Then sell the

software either as a part of another product or on its own. Generally,OSS does not allow the rights of modification for which the software may be used. OSS has shown its users to bevery much secured as compared to the proprietary software from large vendors.

Another noteworthy outcome was the improved handling of transitive dependencies. Traditional SCA tools failed to detect vulnerable libraries that were several layers deep in dependency trees, whereas DRSA-SCA identified and scored these components accurately, prompting timely interventions.Finally, qualitative feedback from security analysts using the system indicated improved visibility into OSS component behavior. The integration of contextual signals allowed teams to make informed decisions about updates, patches, or replacement of components, thus strengthening the overall security posture.

Risk signal extraction and threat interpretation both led to an increase in SNR, as seen in the graph. The suggested approach generates signals that are cleaner and easier to identify from background noise, as DRSA-SCA exhibits a substantially higher SNR (8.5 dB) than conventional SCA (4.2 dB). This is because it is far better at distinguishing between legitimate dangers and random data variations. The SNR levels is shown in Figure 3.

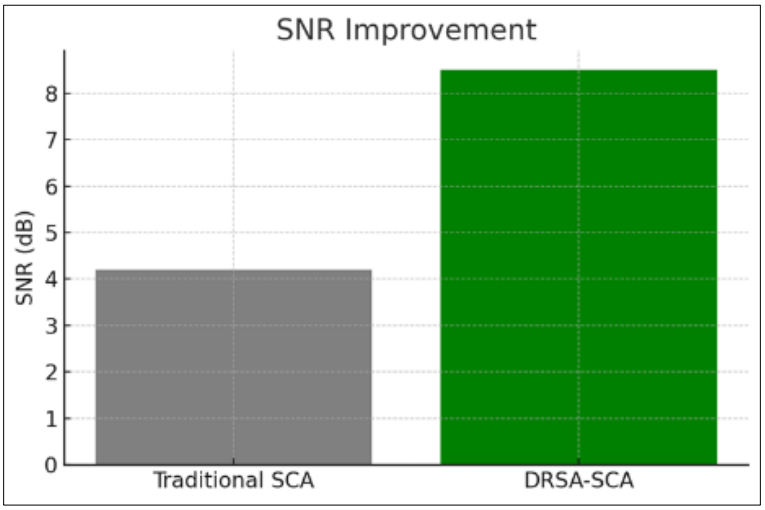

Figure 3 SNR Levels

In contrast to conventional SCA, which does not have a dynamic feedback loop and so does not demonstrate any significant improvement, DRSA-SCA decreases average threat reaction time by 42%. This exemplifies the strength of DRSA-SCA in reacting swiftly to new threats by utilizing real-time threat intelligence. The response time reduction levels is shown in Figure 4.

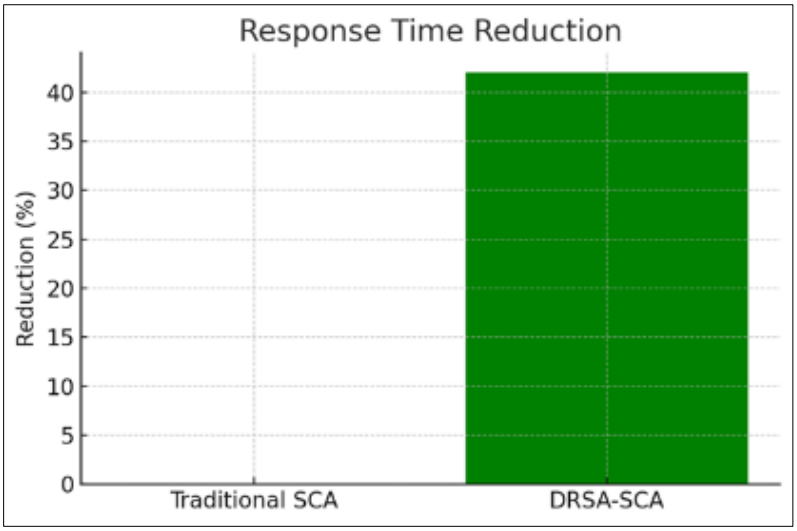

Figure 4 Response Time Reduction Levels

In contrast to more conventional methods, the DRSA-SCA model was able to identify 85% of wild risks (non-CVE vulnerabilities) prior to public publication. This highlights the benefit of the model's behavior-driven and contextual analysis in discovering hidden weaknesses. The Detection of Non-CVE Vulnerabilities is depicted in Figure 5.

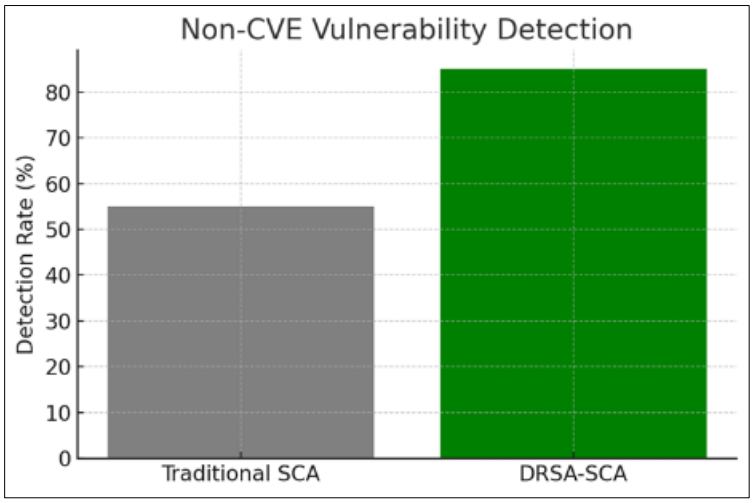

Figure 5 Detection of Non-CVE Vulnerabilities

5. Conclusion

Open-source artificial intelligence refers to AI technologies whose source code is freely available, allowing anyone to study, use, modify, and distribute it. The public availability of open-source AI technologies fostered a community of volunteers who could experiment with and develop new AI applications. This research addresses a critical gap in open-source software security by introducing a continuous, risk-aware framework capable of identifying and mitigating wild risks. Through the use of dynamic threat feeds and behavioral analytics, the proposed model moves beyond static vulnerability scanning and provides actionable intelligence that evolves with the threat landscape.The DRSA-SCA framework lays the foundation for an adaptive, intelligent software security lifecycle. By incorporating real-time monitoring, dynamic risk scoring, and automated interventions, it offers a scalable solution for modern software development environments heavily reliant on open-source components. Based on an extensive literature review and a study from a telecom company (Telenor IT Norway), we have identified several risks related to the deployment of OSS products. However, the paper's main contributions are the identified steps for reducing these risks. In addition, we establish a link between our results from a company and results the literature in Section 5. There are limitations associated with the findings from this paper. Nevertheless, we believe the results of this study are a first step towards focusing the research, on risks of OSS adoption, on more measurable approaches for such evaluation. Finally, our study focuses on bridging the gap between OSS research and practice by focusing on topics highly relevant to practitioners. The study is furthermore an example of how researchers and practitioners may benefit from closer collaboration. As future work we intend to follow the process of adoption of OSS at this company to further investigate and measure the real effect of the adoption of OSS. A particular focus will be directed towards the relationship between the Telenor IT's internal development and support, and their partners. We also acknowledge that many of the risks and mitigation steps described in this paper are similar to the ones described in the literature of adoption/diffusion of general information technology. This research could also lend research on OSS adoption valuable support. We intend to do more research in order to investigate these issues, so we can focus the OSS research on the issues that are mostly related to the OSS adoption, and not part of the general issues related to general adoption/diffusion of information technology.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] S. Lee, H. Baek and J. Jahng, "Governance strategies for open collaboration: Focusing on resource allocation in open source software development organizations", *Int. J. Inf. Manage.*, vol. 37, no. 5, pp. 431-437, Oct. 2017.

[2] J. Singh and L. S Maurya, "Reliability assessment and prediction of open source software systems", *Proc. IEEE 2nd Int. Conf. Image Inf. Process. (ICIIP)*, pp. 6-11, Dec. 2013.

[3] J. Coelho and M. T. Valente, "Why modern open source projects fail", *Proc. 11th Joint Meeting Found. Softw. Eng.*, pp. 186-196, Aug. 2017.

[4] M. Rashid, "Remedying knowledge loss in free/libre open source software", *Proc. 20th Int. Conf. Eval. Assessment Softw. Eng.*, pp. 1-4, Jun. 2016.

[5] M. Zimmermann, C.-A. Staicu, C. Tenny and M. Pradel, "Small world with high risks: A study of security threats in the NPM ecosystem", *Proc. 28th {USENIX} Secur. Symp.*, pp. 995-1010, 2019.

[6] F. Li and V. Paxson, "A large-scale empirical study of security patches", *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, pp. 2201-2215, 2017.

[7] B. Chinthanet, R. G. Kula, S. McIntosh, T. Ishio, A. Ihara and K. Matsumoto, "Lags in the release adoption and propagation of NPM vulnerability fixes", *Empirical Softw. Eng.*, vol. 26, no. 3, pp. 1-28, 2021.

[8] Pashchenko, D.-L. Vu and F. Massacci, "A qualitative study of dependency management and its security implications", *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, pp. 1513-1531, 2020.

[9] R. G. Kula, D. M. German, A. Ouni, T. Ishio and K. Inoue, "Do developers update their library dependencies?", *Empirical Softw. Eng.*, vol. 23, no. 1, pp. 384-417, 2018.

[10] Ruohonen, S. Hyrynsalmi and V. Leppänen, "A mixed methods probe into the direct disclosure of software vulnerabilities", *Comput. Hum. Behav.*, vol. 103, pp. 161-173, 2020.

[11] R. Ramsauer, L. Bulwahn, D. Lohmann and W. Mauerer, "The sound of silence: Mining security vulnerabilities from secret integration channels in open-source projects", *Proc. ACM SIGSAC Conf. Cloud Comput. Secur. Workshop*, pp. 147-157, 2020.

[12] Y. Zhou, J. K. Siow, C. Wang, S. Liu and Y. Liu, "SPI: Automated identification of security patches via commits", *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 1, pp. 1-27, 2021.

[13] Zhou et al., "Finding a needle in a haystack: Automated mining of silent vulnerability fixes", *Proc. 36th IEEE/ACM Int. Conf. Automated Softw. Eng.*, pp. 705-716, 2021.

[14] Z. Xu, B. Chen, M. Chandramohan, Y. Liu and F. Song, "Spain: Security patch analysis for binaries towards understanding the pain and pills", *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng.*, pp. 462-472, 2017.

[15] R. E. Zapata, R. G. Kula, B. Chinthanet, T. Ishio, K. Matsumoto and A. Ihara, "Towards smoother library migrations: A look at vulnerable dependency migrations at function level for npm Javascript packages", *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, pp. 559-563, Sep. 2018.

[16] H. Plate, S. Elisa Ponta and A. Sabetta, "Impact assessment for vulnerabilities in open-source software libraries", *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, pp. 411-420, Oct. 2015.

[17] Pashchenko, H. Plate, S. E. Ponta, A. Sabetta and F. Massacci, "Vulnerable open source dependencies: Counting those that matter", *Proc. 12th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, pp. 1-10, Oct. 2018.

[18] A.Decan, T. Mens and E. Constantinou, "On the impact of security vulnerabilities in the npm package dependency network", *Proc. IEEE/ACM 15th Int. Conf. Mining Softw. Repositories (MSR)*, pp. 181-191, May 2018.

[19] Y. Yano, R. Gaikovina Kula, T. Ishio and K. Inoue, "VerXCombo: An interactive data visualization of popular library version combinations", *Proc. IEEE 23rd Int. Conf. Program Comprehension*, pp. 291-294, May 2015.

[20] R. G. Kula, C. De Roover, D. German, T. Ishio and K. Inoue, "Visualizing the evolution of systems and their library dependencies", *Proc. 2nd IEEE Work. Conf. Softw. Visualizat.*, pp. 127-136, Sep. 2014.

[21] R. G. Kula, C. De Roover, D. M. German, T. Ishio and K. Inoue, "A generalized model for visualizing library popularity adoption and diffusion within a software ecosystem", *Proc. IEEE 25th Int. Conf. Softw. Anal. Evol. Reengineering (SANER)*, pp. 288-299, Mar. 2018.

[22] R. G. Kula, D. M. German, A. Ouni, T. Ishio and K. Inoue, "Do developers update their library dependencies? An empirical study on the impact of security advisories on library migration", *Empirical Softw. Eng.*, vol. 23, pp. 384-417, Feb. 2018.

[23] Cox, E. Bouwers, M. van Eekelen and J. Visser, "Measuring dependency freshness in software systems", *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, vol. 2, pp. 109-118, May 2015.

[24] Y. Wang, B. Chen, K. Huang, B. Shi, C. Xu, X. Peng, et al., "An empirical study of usages updates and risks of third-party libraries in Java projects", *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, pp. 35-45, Oct. 2020.