Risk Measurement in Open-Source Software

1. Introduction

The use of Open Source Software in large software projects has been growing in the last number of years (Zajdel, Costa, and Mili 2022). The supply chain is a chain of dependencies, which can include open-source software, adding value to a software asset as a network of participants performs activities on these dependencies (K. Singi et al. 2019). As the reliance on open-source software continues to grow, attackers have begun to exploit vulnerabilities within the trusted components in the software supply chain by injecting malicious code into software packages that compromise dependent systems (Ohm et al. 2020). Supply chain attacks are set to increase year after year. Attacks recorded in 2020 accounted for 34% of all recorded attacks and attacks in 2021 accounted for 66% of all recorded attacks (M. Z. Malik and S. Z. A. Bukhari 2023). A data breach from a supply chain attack is a major cost to users of open-source software. The cost of a data breach of 50 million records is $388 million on average (X. Wang 2021). Vulnerable components pose significant risks to the security and integrity of these software supply chains. When developing software that incorporates open-source software, it is important to ensure that the risk is minimised in this supply chain.

Measuring risk is essential in ensuring software systems' security, reliability and compliance. While open-source software is transparent as it is accessible to everyone, it can be susceptible to vulnerabilities that require identification and mitigation. By understanding the risks associated with their open-source supply chain, organisations can safeguard against data breaches and make informed decisions on software adoption when comparing different components. It also encourages collaboration in open-source communities, by encouraging the maintenance of open-source software and ensuring the integrity of software supply chains. Prediction of risk is a technique that can be used to decide whether certain open-source projects are worth using and several methods are used to predict this risk. We will research several methods for predicting risk focusing on three key areas.

In section two, we surveyed a number of papers based on three objects of analysis commonly used in the research literature: the code itself, metadata from the project repository, and vulnerability reports and databases. Some risk measures can be derived from analysing the code itself for vulnerabilities and we survey these in section 2.1 breaking them down into software metrics such as code complexity, program size etc and text mining into tokens to locate possible vulnerabilities. Other measures are drawn from analysing code repository metadata of the open-source projects on GitHub. A selection of papers of this type are surveyed in section 2.2 analysing project activity over time, time lags in the engineering process and prediction of project activity. The final approach we surveyed in section 2.3 focuses on vulnerability reports and databases. Finally, in section three we discuss the gaps in research we identified as well as the conclusions. We identify the research question that we plan to focus on for the duration of our project as well as the reasons why we believe it adds to the current research.

2. Related Work

2.1. Code Analysis

	Cross-Project	Text Mining	Software Metrics
(Theisen and Williams, 2020)		x	x
(Abunadi and Alenezi, 2015)	x	x	x
(Kalouptsoglou et al., 2022)		x	x
(R. Scandariato et al., 2014)	x	x	
(D. Mandal and İ. KÖsesoy, 2023)		x	
(A. Zhou et al., 2021)			x
(K. Z. Sultana et al., 2018)			x
(T. -Y. Chong et al., 2019)	x		x

Table 1: Identification of models analysed in papers on code analysis.

Table 1 is a taxonomy of the studies researched on code analysis to predict vulnerabilities. The main models used were based on text mining of source code and prediction based on software metrics. There are few papers which attempt to predict cross-project after training on another project and even fewer predicting across different languages.

2.1.1. Vulnerability Prediction Studies

In (Theisen and Williams 2020) the authors compare several vulnerability prediction models (VPM). Software metrics, text mining and crash data were used and combined in this study. They aim to identify appropriate features to use in vulnerability prediction and measure the accuracy. The accuracy was calculated using an F-score. Naive Bayes consistently outperformed the other models compared. The authors in (Abunadi and Alenezi 2015) describe an empirical study to clarify how useful cross-project prediction is in predicting software vulnerabilities. The problem is formulated as a classification problem by predicting if a PHP file is vulnerable or not. They trained their model on two different projects and their class labels and then evaluated the model on a third project. Five classifiers were used in this paper - Naive Bayes, Logistic Regression, SVMs, J48 and Random Forest. They evaluate using F-measure as well. A systematic mapping study was done on software vulnerability prediction in (Kalouptsoglou et al. 2023). An extensive literature review of 180 studies was conducted where they mapped studies to variables of interest and identified trends and relationships between studies. They classify the research analysed based on the datasets, the evaluation, techniques and research goals. This review helped to identify research gaps. They claim that more research needs to be done on cross-project vulnerability prediction as many of the papers they analysed were project-specific.

2.1.2. Text Mining

Text mining is another technique often used to predict vulnerabilities where the analysis is done on text tokens of the source code to identify patterns. In (Kalouptsoglou et al. 2022) the authors compare software metrics and text-mining based vulnerability prediction models. Datasets from real-world software products were used and they found that text mining outperformed software metrics models. They followed previous research in terms of datasets and then worked from there. Both deep learning and machine learning models were used and evaluated using F-scores. The authors in (R. Scandariato et al. 2014) explored text mining with machine learning to predict software vulnerabilities using a series of experiments and evaluating using precision and recall. In this study, they refer to software components as Java files. Precision and recall are also used in this study to evaluate the performance of the models. Cross-project experimentation was done in this study and the authors found that it is difficult to determine whether a model built on a single application is applicable to other projects. Another paper studying the prediction of vulnerabilities via text is (D. Mandal and İ. KÖsesoy 2023). Machine learning models were used to predict vulnerabilities in Java code. They found the TF-IDF feature extraction method with SVM and Logistic Regression algorithms was the most accurate after tokenisation. This study had some very promising results and the authors suggest more research is required for other feature extraction methods.

2.1.3. Software Metrics

The authors of (A. Zhou, K. Z. Sultana, and B. K. Samanthula 2021) take an interesting approach to researching software vulnerability prediction models. They track the changes in the behaviour of software metrics when the analysed code is fixed. Open-source files with a low chance of false positives were used as the dataset. After comparing the values of metrics they noted that most of the software metrics did not show any changes in values when the developers removed the vulnerabilities and inferred that complex code is more vulnerable. A comparison of nano-patterns and software metrics in vulnerability prediction is done in (K. Z. Sultana, B. J. Williams, and A. Bosu 2018). The same three machine learning techniques were used to compare the two models and found that nano-patterns had lower false negative rates but software metrics had higher precision. The performance evaluators they used were precision and recall - similar to many other papers. In (T. -Y. Chong, V. Anu, and K. Z. Sultana 2019) the authors conducted experiments on vulnerabilities reported in three Java projects and two Python projects. This cross-project prediction was unusual among the researched papers and they added to the research by focusing on predicting specific functions as vulnerable. The trend of using precision and recall to measure performance continues here. The two machine learning algorithms used found that the performance measures were consistent among the Java projects and the performance measures for the Python projects were quite low.

2.2. Project Metadata Analysis

	Time Lags in Process	Project Activity	Prediction
(A. Ait et al., 2022)		x	

(K. Champion and B. M. Hill, 2021)		x	
(Coelho et al., 2020)		x	x
(Chinthanet et al., 2021)	x		
(N. Imtiaz et al., 2023)	x		
(A. Decan et al., 2018)	x		
(L. Bao et al., 2021)		x	x
(Xia et al., 2022)		x	x

Table 2: Identification of methods analysed in papers on open-source project activity.

Table 2 is a taxonomy of the studies researched that use project metadata to analyse the project activity. There were many different statistical and machine learning algorithms used in this section of papers to identify different activities in projects such as changepoints, underproduction and some time lags in the engineering process.

2.2.1. Project Activity Over Time

In (A. Ait, J. L. C. Izquierdo, and J. Cabot 2022) the authors conduct an analysis of activity evolution along the lifespan of an open-source project. It also analyses what the survival rate of open-source projects is. A nonparametric PELT algorithm was used which is reproducible and has a large dataset. The Pruned Exact Linear Time (PELT) algorithm is used for detecting change points over time. It could prove useful to flag projects at risk of dying soon as they will no longer be updating/fixing security vulnerabilities. Another method of predicting risk in project activity is underproduction which is explored in (K. Champion and B. M. Hill 2021). Underproduction is when the supply of software engineering labour is not aligned with the demands of the users. This paper presents a new measure to identify at-risk software by measuring underproduction using measures of quality and importance that can be recorded consistently across a gathered collection of software artefacts. They also proposed an ex-ante relationship between two measures and measured deviance from this. This was a good proposal and could benefit from being explored in a more general way as it was Debian-specific. In (Coelho et al. 2020) a data-driven approach to measure the level of maintenance activity of GitHub projects is presented. They train machine learning models to measure the level of maintenance activity on GitHub projects - the evaluation was done on the historical evolution of active projects in a year. This approach does not rely on a threshold as they claim they are not empirically validated. This is another interesting approach to identifying risks in open-source projects.

2.2.2. Time Lags in Engineering Process

In (Chinthanet et al. 2021) the authors discuss lags in the propagation of vulnerability patches in the NPM open-source project. This was an empirical study to identify any of the lags in these three points in the timeline: the release, the adoption and the propagation of vulnerability fixes. This was evaluated using Cramer's V which is the measure of association between two nominal categories. It was a novel analysis which identified a number of lags in

the pipeline with the disadvantage that it only focussed on NPM packages. In (N. Imtiaz, A. Khanom, and L. Williams 2023) the time lag between the security fix being created and the release being published and the time lag between the security release and the security advisory being published were analysed. This paper used a box plot analysis of these time lags. It was a thorough investigation of open source security releases on Snyk although the filtering due to unidentifiable repositories is a weakness in this paper. Another paper (A. Decan, T. Mens, and E. Constantinou 2018) discusses the technical lag in the npm dependency network from different angles. This paper relates the lag to the release type of the dependent packages and analyses the continuous evolution of the technical lag. The evaluation consisted of analysing the distribution of the time lags and possible actions to reduce the time lag. This is an interesting analysis that could certainly be applied to other dependency networks to identify reasons for these technical lags.

2.2.3. Project Activity Predictions

Predicting project activity in open-source software is another way of measuring the success and security of projects. In (L. Bao et al. 2021) they examine specifically the contributors in open source projects and use several machine learning and deep learning techniques to predict their activity. They attempt to differentiate between newcomers who leave the project after a short time and those who stay for a long time. Some of the most important features are the number of followers, the programming language and the average number of commits per developer. A dataset from GitHub was used as well as a survey conducted among developers to see their opinions on factors that affect long-term contributors. Predicting the health of an open-source project is explored in (Xia et al. 2022). Data from over one thousand GitHub projects was used to predict multiple health indicators using machine learning and the error rate was reduced using hyperparameter optimisation. They predict things such as the number of contributors, the number of commits, and the number of pull requests and issues. These metrics are indicative of the project's activity level. They also conducted a survey which indicated that these are real-world concerns for developers.

2.3. Vulnerability Reports and Database Analysis

2.3.1. Time Series Forecasting

Time series forecasting is another method used when predicting vulnerabilities. In (Gencer and Başçiftçi 2021) the authors analyse the use of the ARIMA model to predict vulnerabilities. Data on vulnerabilities from the NVD was used in this study ('Vulnerability APIs', 2024.). The National Vulnerability Database (NVD) is a database maintained by the National Institute for Technology which keeps updated details on known vulnerabilities. The authors aimed to predict the number of vulnerabilities for Android per month. They compared its performance with the long short-term memory model, artificial neural network and recurrent neural network deep learning techniques and found ARIMA gives better results. They found the LTSM to have comparable error rates with ARIMA and to have more successful results than the two deep learning techniques. Another paper discusses the use of ARIMA to predict vulnerabilities (Pokhrel, Rodrigo, and Tsokos 2017). They use a linear and a non-linear approach comparing ARIMA and Artificial Neural Networks ((ANN) and Support Vector Machines (SVM) again using the reported vulnerabilities on the NVD for three different Operating Systems. The results indicated that there were sharp fluctuations in the number of vulnerabilities, the non-linear approaches were a better predictor however, the

ANN did not have enough data to perform well. The authors of (Roumani, Nwankpa, and Roumani 2015) used two methods of time series analysis to predict vulnerabilities for web browsers: ARIMA and exponential smoothing. The trend, level and seasonality of the vulnerabilities were considered to help in prediction. They note that level was the only significant parameter. Datasets from the NVD were also used in this paper.

2.3.2. Time to Fix

The authors of (Ben Othmane et al. 2017) investigate the time for addressing software security issues. They look at prediction models and impacting factors. SAP's automatically collected data was used and "time to fix" is the duration between the reported issue and the time at which it is marked as closed. Logistic regression outperformed the other models on the prediction and time to fix had high variability. They found the most important factor to be the component itself in estimating the time to fix. Predicting bug fix time is investigated in (Y. Lee et al. 2020). Deep learning in activity stream embedding was used to predict the time. The time was recalculated based on the log messages when the bug reports changed. They used bug tracking systems of open source projects to investigate the fix time of these bugs. The model was found to be competitive with other available models even with a small dataset. A replication study using an open-source project was done to predict bug-fixing time (Akbarinasaji, Caglayan, and Bener 2018). A Markov-based model was used in this paper to predict the number of bugs that can be fixed per month and the model was found to be robust enough to be generalised. The model was capable of predicting the number of fixed bugs in three consecutive months.

3. Conclusions and Research Question

Research Question: Can we combine features extracted from each of the risk measurement methods to provide developers with a more effective risk measure that allows them to minimise risk when choosing between multiple candidate open-source components?

The studies we have surveyed above explore risk from only one metric, from the code, the project activity or data from vulnerability databases. We would like to investigate if a combination of these risk measurements will give a more accurate risk measurement when choosing open-source components. A combination of features from each risk measurement method could give a more rounded view of the open-source projects. Using a risk metric from the source code will help to identify risky components in open-source projects, particularly if the user would like to use them. The project activity metrics will indicate whether an open-source project is actively maintained and still being updated so if vulnerabilities are found they will likely be fixed promptly. Combining these two metrics with the final metric of analysing vulnerability databases to predict the number of vulnerabilities could help in identifying a more accurate view of whether an open-source project is risky to use.

In conclusion, the growing reliance on open-source software has introduced significant challenges regarding the security and integrity of software supply chains. Supply chain attacks, characterised by the exploitation of vulnerabilities within trusted components, have escalated in frequency and severity, posing substantial risks to organisations and users

alike. The average cost of a major data breach resulting from such attacks underscores the critical need for effective risk management strategies. Throughout this review, we have explored various approaches to measuring and predicting risks associated with open-source software. From vulnerability prediction methods through analysing source code to analysing project activity of an open-source project, each section offers valuable insights into the complex landscape of software security. We also explore the measurement of risk through the use of vulnerability database information.

Separating the literature into the three different risk measurement methods helped us to identify gaps in the research such as predicting the time lag between vulnerability release and fix release as a risk measure of open-source software. Supply chain attacks are increasing as time goes on and as more vulnerabilities are reported the more secure a supply chain is the better thus measuring the risk in the use of proposed open-source software is increasingly important. Another gap we have identified is the use of VPMs to predict vulnerabilities across projects. There is data available freely on GitHub and the NVD to investigate some of the above-mentioned measurements of risk.

4. References

A. Ait, J. L. C. Izquierdo, and J. Cabot. 2022. ‘An Empirical Study on the Survival Rate of GitHub Projects’. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, 365–75. https://doi.org/10.1145/3524842.3527941.

A. Decan, T. Mens, and E. Constantinou. 2018. ‘On the Evolution of Technical Lag in the Npm Package Dependency Network’. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 404–14. https://doi.org/10.1109/ICSME.2018.00050.

A. Zhou, K. Z. Sultana, and B. K. Samanthula. 2021. ‘Investigating the Changes in Software Metrics after Vulnerability Is Fixed’. In *2021 IEEE International Conference on Big Data (Big Data)*, 5658–63. https://doi.org/10.1109/BigData52589.2021.9671334.

Abunadi, Ibrahim, and Mamdouh Alenezi. 2015. ‘Towards Cross Project Vulnerability Prediction in Open Source Web Applications’. In *Proceedings of the The International Conference on Engineering & MIS 2015*. ICEMIS ’15. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2832987.2833051.

Akbarinasaji, Shirin, Bora Caglayan, and Ayse Bener. 2018. ‘Predicting Bug-Fixing Time: A Replication Study Using an Open Source Software Project’. *Journal of Systems and Software* 136 (February): 173–86. https://doi.org/10.1016/j.jss.2017.02.021.

Ben Othmane, Lotfi, Golriz Chehrazi, Eric Bodden, Petar Tsalovski, and Achim D. Brucker. 2017. ‘Time for Addressing Software Security Issues: Prediction Models and Impacting Factors’. *Data Science and Engineering* 2 (2): 107–24. https://doi.org/10.1007/s41019-016-0019-8.

Bilge, Leyla, and Tudor Dumitras. 2012. *Before We Knew It: An Empirical Study of Zero-Day Attacks in the Real World*. *Proceedings of the ACM Conference on Computer and Communications Security*. https://doi.org/10.1145/2382196.2382284.

Chinthanet, Bodin, Raula Gaikovina Kula, Shane McIntosh, Takashi Ishio, Akinori Ihara, and Kenichi Matsumoto. 2021. ‘Lags in the Release, Adoption, and Propagation of Npm Vulnerability Fixes’. *Empirical Softw. Engg.* 26 (3). https://doi.org/10.1007/s10664-021-09951-x.

Coelho, Jailton, Marco Tulio Valente, Luciano Milen, and Luciana L. Silva. 2020. ‘Is This GitHub Project Maintained? Measuring the Level of Maintenance Activity of Open-Source Projects’. *Information and Software Technology* 122 (June): 106274. https://doi.org/10.1016/j.infsof.2020.106274.

D. Mandal and İ. KÖsesoy. 2023. ‘Prediction of Software Security Vulnerabilities from Source Code Using Machine Learning Methods’. In *2023 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 1–6. https://doi.org/10.1109/ASYU58738.2023.10296747.

Gencer, Kerem, and Fatih Başçiftçi. 2021. ‘Time Series Forecast Modeling of Vulnerabilities in the Android Operating System Using ARIMA and Deep Learning Methods’. *Sustainable Computing: Informatics and Systems* 30 (June): 100515. https://doi.org/10.1016/j.suscom.2021.100515.

K. Champion and B. M. Hill. 2021. ‘Underproduction: An Approach for Measuring Risk in Open Source Software’. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 388–99. https://doi.org/10.1109/SANER50967.2021.00043.

K. Singi, J. C. B. R P, S. Podder, and A. P. Burden. 2019. ‘Trusted Software Supply Chain’. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 1212–13. https://doi.org/10.1109/ASE.2019.00141.

K. Z. Sultana, B. J. Williams, and A. Bosu. 2018. ‘A Comparison of Nano-Patterns vs. Software Metrics in Vulnerability Prediction’. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, 355–64. https://doi.org/10.1109/APSEC.2018.00050.

Kalouptsoglou, Ilias, Miltiadis Siavvas, Apostolos Ampatzoglou, Dionysios Kehagias, and Alexander Chatzigeorgiou. 2023. ‘Software Vulnerability Prediction: A Systematic Mapping Study’. *Information and Software Technology* 164 (December): 107303. https://doi.org/10.1016/j.infsof.2023.107303.

Kalouptsoglou, Ilias, Miltiadis Siavvas, Dionysios Kehagias, Alexandros Chatzigeorgiou, and Apostolos Ampatzoglou. 2022. ‘Examining the Capacity of Text Mining and Software Metrics in Vulnerability Prediction.’ *Entropy (Basel, Switzerland)* 24 (5). https://doi.org/10.3390/e24050651.

L. Bao, X. Xia, D. Lo, and G. C. Murphy. 2021. ‘A Large Scale Study of Long-Time Contributor Prediction for GitHub Projects’. *IEEE Transactions on Software Engineering* 47 (6): 1277–98. https://doi.org/10.1109/TSE.2019.2918536.

M. Z. Malik and S. Z. A. Bukhari. 2023. ‘Protection Mechanism against Software Supply Chain Attacks through Blockchain’. In *2023 International Conference on Communication Technologies (ComTech)*, 73–78. https://doi.org/10.1109/ComTech57708.2023.10164932.

N. Imtiaz, A. Khanom, and L. Williams. 2023. ‘Open or Sneaky? Fast or Slow? Light or Heavy?: Investigating Security Releases of Open Source Packages’. *IEEE Transactions on Software Engineering* 49 (4): 1540–60. https://doi.org/10.1109/TSE.2022.3181010.

Ohm, Marc, Henrik Plate, Arnold Sykosch, and Michael Meier. 2020. ‘Backstabber’s Knife Collection: A Review of Open Source Software Supply Chain Attacks’. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 17th International Conference, DIMVA 2020, Lisbon, Portugal, June 24–26, 2020, Proceedings*, 23–43. Berlin, Heidelberg: Springer-Verlag. https://doi.org/10.1007/978-3-030-52683-2_2.

Pokhrel, Nawa Raj, Hansapani Rodrigo, and Chris Tsokos. 2017. ‘Cybersecurity: Time Series Predictive Modeling of Vulnerabilities of Desktop Operating System Using Linear and Non-Linear Approach’. *Journal of Information Security* 08 (January): 362–82. https://doi.org/10.4236/jis.2017.84023.

R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen. 2014. ‘Predicting Vulnerable Software Components via Text Mining’. *IEEE Transactions on Software Engineering* 40 (10): 993–1006. https://doi.org/10.1109/TSE.2014.2340398.

Roumani, Yaman, Joseph K. Nwankpa, and Yazan F. Roumani. 2015. ‘Time Series Modeling of Vulnerabilities’. *Comput. Secur.* 51 (C): 32–40. https://doi.org/10.1016/j.cose.2015.03.003.

T. -Y. Chong, V. Anu, and K. Z. Sultana. 2019. ‘Using Software Metrics for Predicting Vulnerable Code-Components: A Study on Java and Python Open Source Projects’.

In *2019 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, 98–103. https://doi.org/10.1109/CSE/EUC.2019.00028.

Theisen, Christopher, and Laurie Williams. 2020. 'Better Together: Comparing Vulnerability Prediction Models'. *Information and Software Technology* 119 (March): 106204. https://doi.org/10.1016/j.infsof.2019.106204.

'Vulnerability APIs'. n.d. Accessed 19 February 2024. https://nvd.nist.gov/developers/vulnerabilities.

X. Wang. 2021. 'On the Feasibility of Detecting Software Supply Chain Attacks'. In *MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM)*, 458–63. https://doi.org/10.1109/MILCOM52596.2021.9652901.

Xia, Tianpei, Wei Fu, Rui Shu, Rishabh Agrawal, and Tim Menzies. 2022. 'Predicting Health Indicators for Open Source Projects (Using Hyperparameter Optimization)'. *Empirical Software Engineering* 27 (6): 122. https://doi.org/10.1007/s10664-022-10171-0.

Y. Lee, S. Lee, C. -G. Lee, I. Yeom, and H. Woo. 2020. 'Continual Prediction of Bug-Fix Time Using Deep Learning-Based Activity Stream Embedding'. *IEEE Access* 8: 10503–15. https://doi.org/10.1109/ACCESS.2020.2965627.

Zajdel, Stan, Diego Elias Costa, and Hafedh Mili. 2022. 'Open Source Software: An Approach to Controlling Usage and Risk in Application Ecosystems'. In *Proceedings of the 26th ACM International Systems and Software Product Line Conference - Volume A*, 154–63. SPLC '22. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3546932.3547000.