Is Popularity an Indicator of Software Security?

Miltiadis Siavvas*[†], Marija Jankovic[†], Dionysios Kehagias[†], Dimitrios Tzovaras[†]
* Imperial College London, SW7 2AZ, London, United Kingdom
[†]Centre for Research and Technology Hellas, Thessaloniki, Greece
m.siavvas16@imperial.ac.uk, jankovicm@iti.gr, diok@iti.gr, dimitrios.tzovaras@iti.gr

Abstract—**Although numerous research attempts can be found in the related literature focusing on the ability of software-related factors (e.g. software metrics) to indicate the existence of vulnerabilities in software applications, none of them have demonstrated perfect results. In addition, none of the existing studies have focused on the popularity of software products, which is an important characteristic of open-source software applications and libraries. To this end, in this paper, the ability of popularity (i.e. utilization) to indicate the existence of vulnerabilities and, in turn, to highlight the internal security level of software products is investigated. For this purpose, a relatively large software repository based on well-known libraries retrieved from the Maven Repository was constructed and its security was analyzed using a widely-used open-source static code analyzer. Correlation analysis was employed in order to examine whether a statistically significant correlation exists between the security and popularity of the selected software products. The preliminary results of the analysis suggest that popularity may not constitute a reliable indicator of the security level of software products. To the best of our knowledge, this is the first study that examines the relationship between the popularity of software products and their security level.**

Keywords*—*software security; vulnerability assessment; vulnerability prediction; popularity; static analysis

I. Introduction

Software security is a matter of major concern for software development companies that wish to provide highly secure services to their clients. The term secure software is commonly used to describe software products that encompass as few vulnerabilities as possible [1]. Most of the vulnerabilities are introduced into software products due to insecure choices made by their developers during the implementation phase [2], [3]. Hence, appropriate mechanisms are required to assist developers in avoiding the introduction of security issues, as well as in mitigating vulnerabilities early enough in the software development cycle.

Vulnerability prediction is considered an effective mechanism for facilitating the production of more secure software products [4]. In fact, vulnerability prediction is a relatively new area of research, which focuses on predicting the existence of vulnerabilities in software products (e.g. [5], [6]), or in individual software modules (e.g. [7]–[11]). This information can be leveraged by developers and project managers to aid decision making regarding the product implementation. For instance, it can be used for selecting between software artifacts (products or modules) that provide the same functionality, or for prioritizing testing and inspection efforts by allocating limited test resources to high-risk areas (i.e. potentially vulnerable parts) of the overall software product. The research in the field of vulnerability prediction focuses chiefly on examining the ability of specific software-related factors (e.g. software metrics) to indicate the existence of vulnerabilities in software, as well as on building vulnerability prediction models (e.g. [5]–[11]), based on these factors.

An important factor of software products (especially of open-source software applications and libraries) is their popularity (i.e. utilization). There is a common truism in software engineering community stating that the more popular a software product is, the more bug-free it is expected to be. This belief is based on the fact that, since popular applications are used by a multitude of users, they are expected to be more well-developed and extensively tested [2], [3]. In fact, popularity is commonly used in practice, for facilitating the selection among third-party components that provide similar functionalities [2], [12], [13].

However, although this belief seems to be intuitive and has been evaluated by several empirical studies (e.g. [14]), it is not clear if it also holds for the case of software security. In other words, no empirical evidence exists in the related literature supporting the belief that popular software applications are also highly secure (i.e. vulnerability free). On the contrary, numerous examples of well-known software applications containing severe vulnerabilities exist (e.g. [15], [16]), which render the empirical evaluation of this conviction a topic of high interest.

Although several studies have extensively examined the ability of common software metrics and other factors to indicate the existence of vulnerabilities in software products [5]–[11] (and, thus, their internal security level), no attempts can be found specifically investigating the relationship between the popularity of software applications and their security. To this end, the present research aims to provide an answer to the following research question:

> ***RQ***: *Is the popularity of software products a reliable indicator of their security level?*

For this purpose, a relatively large software repository (comprising approximately 2 million lines of code) was constructed based on well-known Java libraries retrieved from the Maven Repository[1] and analyzed using a popular static code analyzer to determine their security level. Subsequently, statistical anal-

[1] https://mvnrepository.com/

ysis was employed to investigate whether statistically significant positive correlation exists between their popularity and their security level. To the best of our knowledge, the present work, constitutes the first study that specifically investigates the relationship between the popularity of software products and their security level (i.e. existence of vulnerabilities), while it uses a significantly larger repository compared to similar attempts in the field of vulnerability prediction [5]–[11]. The related work, experiment setup, results, and conclusions are presented in the rest of the paper.

II. Related Work

A. Existence of Vulnerabilities in Popular Software Products

As already mentioned, popular software applications are usually expected to be more well-developed, and therefore considerably free from important bugs, including security vulnerabilities [2], [3]. However, as can be seen by the National Vulnerability Database (NVD)[2], a large number of its entries correspond to severe vulnerabilities that belong to popular software applications. Additionally, as discussed in the rest of this section, several real-world examples and empirical studies have highlighted the existence of important security issues even in well-known software products.

HeartBleed [15] and Equifax Breach [16], constitute two of the most representative real-world examples of security issues caused by vulnerabilities located in well-known software products. In particular, HeartBleed [15], is a serious vulnerability found in OpenSSL, which is an open-source library that allows secure communication between two peers based on the well-known SSL/TLS protocol. This vulnerability, which was caused by improper input validation, led to information leakage, allowing malicious individuals to retrieve sensitive information. Similarly, Equifax Breach [16], allowed criminals to expose the data of more than 145 million Equifax customers, and was caused by a vulnerability located in Apache Struts 2, a popular framework for building large-scale web applications. In particular, this vulnerability allowed adversaries to perform remote code execution attack to Apache Servers and steal confidential information. Both Heartbleed and Equifax Breach are registered in the Common Vulnerabilities and Exposure[3] (CVE) database as CVE-2014-0160 and CVE-2017-5638 respectively.

Several empirical studies have highlighted the existence of important vulnerabilities in popular reusable software artifacts, and especially in software libraries. For instance, in [17] Lisvits et al., by evaluating their custom context-sensitive static code analyzer on 9 open-source software applications, identified 29 unreported vulnerabilities, two of which were found in widely-used Java libraries. In addition, an extensive analysis of the Java libraries located at Maven Repository [18] revealed that malicious code issues are highly prevalent in these software artifacts, whereas important security issues, such as lack of input sanitization/validation are also common. The existence of security issues in well-known reusable third-party components is important, since their vulnerabilities are actually vulnerabilities of the products using them [13], [19].

Although several studies have extensively investigated the potential existence of vulnerabilities in popular software applications, none of them examined the relationship between their popularity and the volume of their security issues. *To this end, the present study attempts to determine whether the popularity of software applications is closely related to their security level (i.e. vulnerability density).*

B. Vulnerability Prediction

Several research endeavors have been conducted focusing on the ability of common software metrics to indicate the existence of vulnerabilities in software products, in an attempt to provide a mechanism able to highlight the internal security of software products [6]–[11]. However, existing attempts are hindered by a set of shortcomings that the present study tries to tackle. Firstly, the majority of existing research works focused chiefly on coupling, cohesion and complexity (CCC) metrics [7]–[9], as well as on metrics related to the developers' activity and code churn [10]. Nevertheless, since none of the previous attempts led to perfect results, there is a strong need for incorporating factors that have not been examined before [7], [20]. *Towards this end, our work extends previous studies by including the popularity (i.e. utilizability) of software libraries, which is a previously uninvestigated factor.*

Another issue is the relatively small size of the repositories that were used for the conduction of the previous studies. In fact, the vast majority of the previous empirical studies (e.g. [6]–[11]) were based on a highly limited number of software products (often one or two software products), with the only exception of [11], in which 14 open-source web applications were used for the needs of the analysis. Our study is based on a repository of 20 open-source software libraries, comprising approximately 2 million lines of code. *Hence, the present analysis constitutes one of the largest studies in terms of code base size that can be found in the related literature.*

The relationship between the popularity and quality of software components has extensively been studied in the related literature. In [21] the popularity of a wide range of software libraries retrieved from the Maven Repository (i.e. their utilization by open-source Sourcerer [22] applications) was compared to their quality (i.e. defect density). However, only negligible positive correlations were observed between the two factors, indicating that popular software libraries are not necessarily of better quality. Similarly, in a recent study [23], a highly sophisticated hierarchical software quality model was employed to assess the quality of a set of Java libraries retrieved from the Maven Repository. A comparison between their quality score and their reputation (i.e. total number of downloads) revealed that no statistically significant positive correlation exists between their quality and popularity. Finally, in [14], an analysis conducted on a large number of Android applications using static code analysis revealed that a strong relationship exists between specific bug categories and the

[2] https://nvd.nist.gov/

[3] http://cve.mitre.org/cve/

user-defined ratings of the applications, which are sufficient indicators of their reputation. Hence, although significant contributions have been made regarding the relationship between the popularity and quality of software products, no similar attempts exist focusing on software security. *To the best of our knowledge, this is the first study that investigates the relationship between the popularity and the security of software products. This constitutes the main novelty of the present study compared to the aforementioned research attempts.*

III. Experiment Setup and Methodology

A. Benchmark Repository

The first step of the experiment was the construction of the appropriate repository of software products. For this purpose, the top 20 Java libraries (that could be analyzed by the selected tools) were retrieved from the Maven Repository, which is the largest online source of Java libraries, leading to a repository of approximately 2 million lines of code. These libraries are widely used by millions of developers around the world, and therefore they constitute sufficient representatives of real-world software products. In fact, the reasoning behind the selection of the Maven Repository was that it has been widely used in the related literature for similar research purposes (e.g. [18], [21], [23]).

It should be noted that the purpose of the present study is to investigate whether the popularity (i.e. utilization) of software products in general, can be used as an indicator of their internal security. Software libraries were selected as the basis of the present analysis, since they constitute representative examples of popular applications, in which popularity is used as a criterion for their selection [2], [12], [13]. Therefore, software libraries were used only as a proof-of-concept, and, thus, no library-specific conclusions are expected to be reached by the present study.

Modern software development is based on the reusability of third-party components (i.e. software libraries and reusable source code fragments), while their popularity is often used as a criterion for selecting between components that provide similar functionality. This reusability-based engineering approach is commonly adopted for the development of software applications, regardless of their type or their adopted programing language. Hence, the results of the analysis are expected to be independent of the benchmark context, and, thus, generalizable to different types of software applications. In other words, although the present analysis is based on software libraries, its results are expected to be of high interest for software applications in general. Finally, similar observations are expected to be made, if different types of software products (e.g. reusable open-source software artifacts) were used for the construction of the benchmark upon which the present analysis is based.

B. Indicators of Software Security and Popularity

The next step of the experiment was the selection of appropriate indicators for software security and popularity. As an indicator of popularity, the total number of the downloads of each library as reported by the Maven Repository was used. Hence, the popularity of software products corresponds to their utilization, an approach commonly used in the related literature for quantifying product popularity (e.g. [18], [21], [23]).

As an indicator of software security, the Static Analysis Vulnerability Density (SAVD) [11] was selected. The SAVD is the Vulnerability Density metric [24] (i.e. the total number of vulnerabilities that a software product contains per thousand lines of code) calculated based on the security-related results produced by static analysis. The Vulnerability Density is a widely-accepted measurement for quantifying the internal security level of software products. Using static analysis results to quantify the Vulnerability Density is a common approach in the related literature. For instance, in [11] and [6] the authors used the SAVD of software products as an indicator of their security and calculated the correlation between a set of product-level software metrics in an attempt to investigate their ability to indicate the existence of vulnerabilities. In [25], the authors constructed a dataset of software products and measured their security status using the SAVD, which was quantified through a commercial static code analyzer. They subsequently used this dataset to produce vulnerability predictors based on text mining. Finally, in a recent attempt [26], a subset of this dataset was used in order to investigate the ability of text mining-based Deep Neural Networks to indicate the existence of vulnerabilities in software products.

Static analysis is a testing technique that searches for potential software bugs (including vulnerabilities) in software products by analyzing their source code without requiring its execution [1]. Automatic static analysis (ASA) is considered an important technique for adding security during the software development process. This belief is expressed by several experts in the field of software security (e.g. [27], [28]), while almost all the well-established secure software development lifecycles (SDLCs), including the well-known Microsoft's SDL [29], [30], OWASP's OpenSAAM[4], and Cigital's Touchpoints [1], propose the adoption of static analysis as the main mechanism for adding security during the coding (i.e. implementation phase) of the SDLC. In addition, ASA is a security activity commonly adopted by major technological firms like Google, Microsoft, Adobe and Intel, as reported by the BSIMM[5] initiative. In fact, ASA has been found effective in detecting security bugs that can lead to severe vulnerabilities like cross-site scripting and SQL Injection, early enough in the SDLC [27], [28]. These vulnerability types are included in the lists of the top most dangerous vulnerabilities maintained by OWASP[6] and CWE. Although the major drawback of ASA is the generation of a large number of false positives, the security-related static analysis results have been found effective in indicating the existence of actual vulnerabilities (e.g. [31], [32]). All the information presented above justifies the quantification of the Vulnerability Density metric using static analysis results.

[4] http://www.opensamm.org/

[5] https://www.bsimm.com/

[6] https://www.owasp.org/index.php/Main_Page

In the present work, we decided to quantify SAVD using the security-related bug patterns provided by FindBugs [33], which is an open-source static code analyzer, widely utilized in the related literature for security auditing purposes. In fact, FindBugs has extensively been used both in academia (e.g. [34]) and in the industry (e.g. Google [35]), for detecting software bugs and security issues, as well as for quantifying both the defect and the vulnerability density of software products. For the purposes of the present study, the tool was properly configured in order to detect and report only software bugs that belong to the security-related bug categories provided by FindBugs, which are: Performance, Malicious Code, Multithreaded Correctness, and Security. The latter bug category is provided by FindSecurityBugs[7], which is a popular FindBugs plugin.

C. Statistical Analysis

Finally, two individual rankings of the selected libraries were exported, one based on their popularity and another based on their security (i.e. SAVD). The two rankings were compared using the Spearman's rank correlation coefficient (ρ) [36], which is a non-parametric and non-sensitive to outliers test, commonly used in the related literature for comparing different rankings (e.g. [7], [11], [20], [23]). The thresholds proposed by Cohen et al. [37] were used in order to characterize the strength of the calculated correlation. According to Cohen et al. [37], a correlation value higher than 0.5 is considered strong, between 0.3 and 0.5 is considered moderate, and below 0.3 is considered low. The statistical significance of the results was tested at 95% level of confidence.

IV. Results and Discussion

The selected software libraries along with their final rankings are presented in Table I. As stated previously, the Spearman's rank correlation coefficient between the two rankings was initially calculated, to get an idea of the relationship between their popularity and security. Subsequently, in order to reach safer conclusions regarding the $\boldsymbol{RQ}$ of the present study, the following hypothesis (and its corresponding null hypothesis) was formulated and tested with confidence level 95% ($p = 0.05$):

> H_1: *A statistically significant correlation exists between the two rankings.*
>
> H_0: *No statistically significant correlation exists between the two rankings.*

For the popularity of the libraries to be a reliable indicator of their security level, the null hypothesis has to be rejected.

The calculated Spearman's rank correlation coefficient between the two rankings was found to be $\rho = -0.17$. The negative correlation value indicates that more popular libraries (that belong to the studied repository) tend to be more vulnerable and thus less secure. However, this relationship was found to be weak, according to the thresholds proposed by Cohen et al. [37]. *This contradicts the belief that widely-used software products are more likely to be secure, and, in turn, the ability of popularity to indicate security.*

Nevertheless, in order to reach safer conclusions, the null hypothesis was tested. Due to the large p-value (i.e. $p = 0.45 > 0.05$), the null hypothesis cannot be rejected. Since we cannot reject the null hypothesis, we cannot conclude that "there is a statistically significant correlation between the two rankings". Hence, the present study does not provide empirical evidence supporting the belief that the popularity of software products is an indicator of their security. *Therefore, popularity may not be used as a* ***reliable*** *indicator of software security.*

It should be noted that failing to reject the null hypothesis does not necessarily mean that we should accept it. However, in this case, similarly to other related research attempts (e.g. [7]–[9]), failing to reject the null hypothesis puts under question the reliability of the studied factor (here the popularity of software products) in indicating software security (i.e. vulnerabilities).

V. Conclusion

The purpose of the present study was to investigate whether the popularity (i.e. utilization) of software products, can also indicate the existence of vulnerabilities, and therefore highlight their internal security level. For this purpose, a relatively large repository of software products was constructed (comprising approximately 2 million lines of code), based on software libraries retrieved from the Maven Central Repository. The total number of their downloads was used as a measure of their popularity, while their Static Analysis Vulnerability Density (SAVD) [11], calculated based on the security-related static analysis results produced by FindBugs [33], as a measure of their security level. Two individual rankings were produced, one based on the popularity of the studied libraries and another one based on their security level, and were compared using the Spearman's rank correlation coefficient [36]. Hypothesis testing was also applied in order to reach safer conclusions. The preliminary results of the analysis suggest that the popularity of software products may not be used as a **reliable** indicator of their security. To the best of our knowledge, the present work constitutes the first study that investigates the relationship between the popularity of software products and their security level.

Although the present study contributes to the validation of the non-reliability of popularity in indicating software security, in order to reach safer conclusions and investigate the generalizability of the results, further work is required. More specifically, similarly to relevant studies [10], [38], the analysis should be replicated by also considering additional types of software products (along with software libraries). This is expected to enhance the reliability of the produced results, since more application types (for which popularity is an important characteristic) will be examined. Moreover, the results of the present study led to the identification of

[7] https://find-sec-bugs.github.io/

TABLE I: THE RANKINGS OF THE SELECTED SOFTWARE LIBRARIES BASED ON THEIR POPULARITY AND SECURITY (I.E. SAVD).

Software Product	Version	SAVD	Popularity Ranking	Security Ranking
JUnit	4.12	0.963507	1	7
SLF4j	1.7.25	14.77713	2	20
Clojure	1.9.0	0.097005	3	1
Logback Classic	1.2.3	1.976059	4	13
Javax Servlet	4.0.0	5.798502	5	19
Jackson Databind	2.9.3	0.347916	6	2
Commons Logging	1.2	3.644431	7	16
Apache HttpClient	4.5.4	3.854823	8	17
Commons Codec	1.11	0.753417	9	5
Osgi Core	6.0.0	1.855288	10	12
Jackson Core	2.9.3	0.795044	11	6
Hamcrest	1.3	1.023734	12	9
Log4j Core	2.10	2.127574	13	14
Google Guice	4.1.0	1.087039	14	10
Maven Project	3.0	2.544228	15	15
Maven Core	3.5.2	1.506344	16	11
Apache Http Core	4.4.8	0.548506	17	3
Commons Http Client	3.1	3.897551	18	18
Hyper SQL	2.4.0	0.613571	19	4
Javax Mail	1.5.0.b01	1.020057	20	8

an interesting direction for future research, i.e., on whether it is reasonable to use popularity as a criterion for selecting between software libraries that provide similar functionality. Although the present analysis was based on software libraries, no library-specific conclusions can be reached, since this would require more elaborate and finer-grained experiments. For example, this would require the re-execution of the same analysis for different repositories, each one of them containing libraries of the same functionality. Hence, the present work raises the awareness of the aforementioned issue, which is part of our planned future research activities.

ACKNOWLEDGMENT

This work is partially funded by the European Union's Horizon 2020 Research and Innovation Programme through SDK4ED project under Grant Agreement No. 780572.

REFERENCES

[1] G. McGraw, *Software Security: Building Security In*. Addison-Wesley Prof., 2006.

[2] G. Wurster and P. C. van Oorschot, "The developer is the enemy," *NSPW '08: Proceedings of the 2008 Workshop on New Security Paradigms*, pp. 89–97, 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1595676.1595691

[3] M. Green and M. Smith, "Developers are Not the Enemy!: The Need for Usable Security APIs," *IEEE Security and Privacy*, vol. 14, no. 5, pp. 40–46, 2016.

[4] M. Siavvas, E. Gelenbe, D. Kehagias, and D. Tzovaras, "Static analysis-based approaches for secure software development," in *Security in Computer and Information Sciences*. Cham: Springer International Publishing, 2018, pp. 142–157.

[5] Y. Roumani, J. K. Nwankpa, and Y. F. Roumani, "Examining the relationship between firm's financial records and security vulnerabilities," *International Journal of Information Management*, vol. 36, no. 6, Part A, pp. 987 – 994, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0268401215302607

[6] J. Walden and M. Doyle, "SAVI: Static-Analysis vulnerability indicator," *IEEE Security and Privacy*, vol. 10, no. 3, pp. 32–39, 2012.

[7] Y. Shin and L. Williams, "Is complexity really the enemy of software security?" in *Proceedings - ACM Conference on Computer and Communications Security*, 2008.

[8] I. Chowdhury and M. Zulkernine, "Can complexity, coupling, and cohesion metrics be used as early indicators of vulnerabilities?" in *Proc. of the 2010 ACM Symposium on Applied Comp.*, 2010.

[9] H. Alves, B. Fonseca, and N. Antunes, "Software Metrics and Security Vulnerabilities: Dataset and Exploratory Study," *Proceedings - 2016 12th European Dependable Computing Conference, EDCC 2016*, 2016.

[10] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities," *IEEE Transactions on Software Engineering*, vol. 37, no. 6, pp. 772–787, 2011.

[11] J. Walden, M. Doyle, G. A. Welch, and M. Whelan, "Security of open source web applications," *2009 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009*, pp. 545–553, 2009.

[12] H. Plate, S. E. Ponta, and A. Sabetta, "Impact assessment for vulnerabilities in open-source software libraries," *2015 IEEE 31st International Conference on Software Maintenance and Evolution, ICSME 2015 - Proceedings*, 2015.

[13] M. Cadariu, E. Bouwers, J. Visser, and A. Van Deursen, "Tracking known security vulnerabilities in proprietary software systems," *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering*, 2015.

[14] H. Khalid, M. Nagappan, and A. E. Hassan, "Examining the relationship between FindBugs warnings and app ratings," *IEEE Software*, vol. 33, no. 4, 2016.

[15] M. Carvalho, J. Demott, R. Ford, and D. A. Wheeler, "Heartbleed 101," *IEEE Security and Privacy*, vol. 12, no. 4, pp. 63–67, 2014.

[16] J. Luszcz, "Apache Struts 2: how technical and development gaps caused the Equifax Breach," *Network Security*, vol. 2018, no. 1, pp. 5–8, jan 2018. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1353485818300059

[17] V. B. Livshits and M. S. Lam, "Finding Security Vulnerabilities in Java Applications with Static Analysis," *Architecture*, p. 18, 2005.

[18] D. Mitropoulos, G. Gousios, P. Papadopoulos, V. Karakoidas, P. Louridas, and D. Spinellis, "The Vulnerability Dataset of a Large Software Ecosystem," *Proceedings - 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, BADGERS 2014*, pp. 69–74, 2016.

[19] N. H. Pham, T. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Detection of recurring software vulnerabilities," *Proceedings of the IEEE/ACM international conference on Automated software engineering*

- *ASE '10*, p. 447, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1858996.1859089

[20] N. Munaiah, F. Camilo, W. Wigham, A. Meneely, and M. Nagappan, "Do bugs foreshadow vulnerabilities? An in-depth study of the chromium project," *Empirical Software Engineering*, vol. 22, no. 3, 2017.

[21] H. Sajnani, V. Saini, J. Ossher, and C. V. Lopes, "Is popularity a measure of quality? An analysis of maven components," *Proceedings - 30th International Conference on Software Maintenance and Evolution, ICSME 2014*, pp. 231–240, 2014.

[22] S. Bajracharya, J. Ossher, and C. Lopes, "Sourcerer: An internet-scale software repository," in *Proceedings - International Conference on Software Engineering*, 2009.

[23] M. Siavvas, K. Chatzidimitriou, and A. Symeonidis, "QATCH - An adaptive framework for software product quality assessment," *Expert Systems with Applications*, 2017.

[24] O. H. Alhazmi, Y. K. Malaiya, and I. Ray, "Measuring, analyzing and predicting security vulnerabilities in software systems," *Computers and Security*, vol. 26, no. 3, 2007.

[25] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, "Predicting vulnerable software components via text mining," *IEEE Transactions on Software Engineering*, vol. 40, no. 10, pp. 993–1006, 2014.

[26] Y. Pang, X. Xue, and H. Wang, "Predicting Vulnerable Software Components through Deep Neural Network," *Proceedings of the 2017 International Conference on Deep Learning Technologies - ICDLT '17*, pp. 6–10, 2017. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3094243.3094245

[27] G. McGraw, "Automated code review tools for security," *Computer*, vol. 41, no. 12, 2008.

[28] B. Chess and G. McGraw, "Static analysis for security," *Security & Privacy, IEEE*, 2004.

[29] M. Howard, *Writing secure code*. Redmond, Wash: Microsoft Press, 2003.

[30] M. Howard and S. Lipner, *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software*. Microsoft Press, 2006.

[31] M. Gegick, L. Williams, J. Osborne, and M. Vouk, "Prioritizing Software Security Fortification through Code-Level Metrics," *Proceedings of the 4th ACM workshop on Quality of Protection*, pp. 31–38, 2008.

[32] J. Yang, D. Ryu, and J. Baik, "Improving vulnerability prediction accuracy with Secure Coding Standard violation measures," *2016 International Conference on Big Data and Smart Computing, BigComp 2016*, pp. 115–122, 2016.

[33] D. Hovemeyer and W. Pugh, "Finding bugs is easy," *ACM SIGPLAN Notices*, 2004.

[34] K. Goseva-Popstojanova and A. Perhinschi, "On the capability of static code analysis to detect security vulnerabilities," *Information and Software Technology*, vol. 68, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2015.08.002

[35] N. Ayewah and W. Pugh, "The Google FindBugs fixit," in *Proceedings of the 19th international symposium on Software testing and analysis - ISSTA '10*, 2010, p. 241.

[36] C. Spearman, "The proof and measurement of association between two things. By C. Spearman, 1904." *The American journal of psychology*, vol. 100, no. 3-4, pp. 441–471, 1987.

[37] J. Cohen, *Statistical power analysis for the behavioral sciences*. Hillsdale, N.J: L. Erlbaum Associates, 1988.

[38] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," *Journal of Systems Architecture*, vol. 57, pp. 294–313, 2011.