On the Use of Refactoring in Security Vulnerability Fixes: An Exploratory Study on Maven Libraries

Ayano Ikegami, Raula Gaikovina Kula
Nara Institute of Science and Technology
Japan
ikegami.ayano.hs9@is.naist.jp
raula-k@is.naist.jp

Ali Ouni
ETS Montreal, University of Quebec, QC
Canada
ali.ouni@etsmtl.ca

Bodin Chinthanet, Vittunyuta Maeprasart
Nara Institute of Science and Technology
Japan
maeprasart.vittunyuta.mn2@is.naist.jp
bodin.ch@is.naist.jp

Takashi Ishio, Kenichi Matsumoto
Nara Institute of Science and Technology
Japan
ishio@is.naist.jp
matumoto@is.naist.jp

ABSTRACT

Third-party library dependencies are commonplace in today's software development. With the growing threat of security vulnerabilities, applying security fixes in a timely manner is important to protect software systems. As such, the community developed a list of software and hardware weakness known as Common Weakness Enumeration (CWE) to assess vulnerabilities. Prior work has revealed that maintenance activities such as refactoring code potentially correlate with security-related aspects in the source code. In this work, we explore the relationship between refactoring and security by analyzing refactoring actions performed jointly with vulnerability fixes in practice. We conducted a case study to analyze 143 maven libraries in which 351 known vulnerabilities had been detected and fixed. Surprisingly, our exploratory results show that developers incorporate refactoring operations in their fixes, with 31.9% (112 out of 351) of the vulnerabilities paired with refactoring actions. We envision this short paper to open up potential new directions to motivate automated tool support, allowing developers to deliver fixes faster, while maintaining their code.

1 INTRODUCTION

Much like any software, third-party software libraries constantly evolve to meet the needs of clients, patch bugs or address other maintenance concerns [29]. Developers often use refactoring, as an effective practice to maintain their evolving software systems, by restructuring code while keeping its external behaviour [24]. Fowler recommends a set of refactoring actions to improve readability, reusability, and increase the speed to maintain code.

The threat of security vulnerabilities, especially in library dependencies, is a growing concern. Developers are wary of the risks of known security vulnerabilities that affect their dependencies [1]. Security advisories allow maintainers and users to be notified of any potential malicious exploits (identified with a unique CVE identifier [33]), which is then assessed by a Common Weakness Enumeration (CWE) [34]. CWE is a community-developed list of software and hardware weakness types, that serves as a baseline for weakness identification, mitigation, and prevention efforts. To reduce lags in the delivery of these fixes [14, 16], recent advancements in technologies such as bots (i.e., Dependabot [25]) are being proposed, so that software teams can quickly fix their software. The urgency to respond and fix vulnerabilities has been further amplified with the recent advisories of highly severe security flaws in the third-party libraries such as Log4Shell, affecting thousands of software application that use the Maven Log4J library.[1]

Prior work revealed that maintenance activities such as refactoring actions (aka operations) correlate with security-related aspects of code [13, 35]. For instance, Abid et al. [13] found correlations between refactoring operations and security-related aspects such as data-access security vulnerability metrics. Their study reveals that refactoring actions have the potential to impact the security of software systems. On the other hand, Mumtaz et al. [35] investigated security metrics (i.e., composition, coupling, extensibility, inheritance and design size) and found that refactoring improves the security of an application without compromising the overall quality in terms of code smells. The key motivation that this early research results is to understand the co-relationship between refactoring when fixing a vulnerability. We hypothesize that fixing a vulnerability may also include cleaning up code.

To fill this gap, in this study, we explore the relationship between refactoring and security vulnerability fixes in practice. In particular, we mine and extract refactoring operations that are jointly performed during a vulnerability fix at the same commit

---

[1]An example of the effect can be seen through this gihub repo https://github.com/NCSC-NL/log4shell

Figure 1: An Example of a Refactor-In-Fix [6], where the refactoring operation(s) and the vulnerability fix are located in the same commit.

(i.e., Refactor-In-Fix). Henceforth, we define the following research questions to guide our study:

- **RQ1: *How prevalent is refactoring in security vulnerability fixing?***
  Motivation: Our motivation is to understand the extent to which refactoring operations are applied to security vulnerability fixes. Different to related work, we investigate refactoring-like actions applied when fixing reported security vulnerabilities (CWEs).
  Results: We find 31.9% of the security vulnerabilities incorporated refactoring operations in the fix (i.e., 112 vulnerabilities instances). Furthermore, the Extract Method is most prevalent refactoring type incorporated in a vulnerability fix.
- **RQ2: *What refactoring operations do developers apply in security vulnerability fixes?***
  Motivation: Based on RQ1 results, we would like to understand which common refactoring patterns that have a higher chance to be applied to refactor a vulnerability fix compared to being applied in other maintenance tasks.
  Results: We identified that the two refactoring types likely incorporated a vulnerability fix are the Extract Variable and the Extract Method.

Our results show that there is potential in dedicated refactoring-aware security vulnerability fix techniques, as we believe the adoption of refactoring in security fixes is crucial, especially since it helps developers deliver quicker patches that should complement existing bots like Dependabot [25], thus enabling a faster advisory and mitigation process.

2 DATA COLLECTION

In this section, we present our approach to pair a refactoring operation to a vulnerability fix, and then our dataset used in the study.

Pairing Refactoring Operations With a Vulnerability Fix (Refactor-In-Fix). A key challenge to respond our research questions, is to accurately identify which refactoring operations were applied to a vulnerability fix. To ensure relateness, we pair any refactoring operations that appear in the same commit as the vulnerability fix.

Figure 1 shows an example of a refactoring operation taken from the Jenkins project [8]. In this case, the Extract and Move Method is being applied in the same commit as the vulnerability fix [6]. The security vulnerability being fixed is related to *'Open redirect to scheme-relative URLs'* (CVE-2016-3726) [10]. Furthermore, the CWE is related to *URL Redirection to Untrusted Site* (CWE-601).

Table 1: A summary statistic of our data collection.

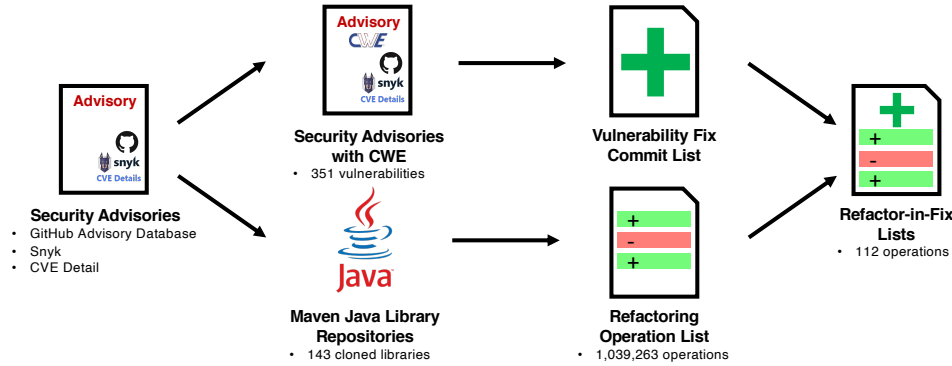| Maven Java Libraries	
Repository snapshot	Aug 28, 2020
# Collected vulnerabilities	1,809
# Collected vulnerabilities with CWE	351
- # Paired with Refactor-In-Fix	112
- # Not paired with Refactor-In-Fix	239
# Maven Java libraries identified	463
- # Cloned Maven Java libraries	143
# Refactoring Operations Extracted	1,039,263
- # Refactor-In-Fix operations	527

Figure 2: An overview of the data collection process.

Table 2: Top 10 most frequent vulnerabilities grouped by CWEs.

CWE ID	CWE Title	# Vulnerabilities Paired	# Not Paired	Total
CWE-200	Improper Input Validation	15 (34.1%)	29 (65.9%)	44
CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	4 (12.5%)	28 (87.5%)	32
CWE-502	Deserialization of Untrusted Data	5 (19.2%)	21 (80.8%)	26
CWE-611	Improper Restriction of XML External Entity Reference	16 (66.7%)	8 (33.3%)	24
CWE-284	Improper Access Control	1 (5.3%)	18 (94.7%)	19
CWE-94	Improper Control of Generation of Code ('Code Injection')	10 (55.6%)	8 (44.4%)	18
CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	1 (5.9%)	16 (94.1%)	17
CWE-352	Cross-Site Request Forgery (CSRF)	9 (60.0%)	6 (40.0%)	15
CWE-264	Permissions, Privileges, and Access Controls	4 (30.8%)	9 (69.2%)	13
CWE-400	Uncontrolled Resource Consumption	4 (36.4%)	7 (63.6%)	11
...	...	...	...	...
	Total	112 (31.9%)	239 (68.1%)	351

This is a severe weakness, with a high score (7.4 HIGH) ranking according to the National Vulnerability Database [11].

Interestingly, we can also see that the vulnerability fix includes the Add Method Annotation and the Extract and Move Method refactoring operations. More specifically, developers added the new annotations @Deprecated, @Restricted, @RestrictedSince to show deprecations of the method. We also see developers performed an extract and move of the .isAbsoluteUri and replace it with a isAbsoluteOrSchemeRelativeUri method to secure the code.

Mining Maven Libraries. Our approach is to mine software repositories to detect refactorings applied on vulnerability fixes. Since we wanted to explore this phenomenon in the wild, we selected Java libraries from Maven [9], one of the largest third-party library ecosystems. The Maven ecosystem of libraries is well-known and actively reports and fixes vulnerabilities, as shown by the number of vulnerabilities that appear in the GitHub Advisory Database [7]. Figure 2 shows the overview of our data collection and how we process them. We first needed to identify libraries that had reported and fixed a security vulnerability. Following prior works [16, 21, 23, 30, 36], we collected data from three reputable data sources (*i*) GitHub Advisory Database [7] (*ii*) Snyk [37] and (*iii*) CVE Details [2]. Similar to [14, 16], we then crawl the advisories to extract any linkages to a GitHub commit, and has a CWE associated

Table 3: Top 10 most frequent refactoring types paired to a vulnerability fix

Refactoring type	# Refactor-In-Fix	# Other
Extract Method	86	37, 688
Add Method Annotation	69	177, 153
Rename Method	58	52, 622
Extract Variable	51	19, 249
Rename Parameter	36	43, 927
Change Variable Type	33	87, 651
Rename Variable	27	43, 982
Change Attribute Type	27	42, 077
Rename Attribute	20	19, 717
Change Return Type	18	58, 939
...	...	...
Total	527	1, 039, 263

with the vulnerability. In the end, from an initial list of vulnerabilities that contains 1,809 advisories, we identified 463 vulnerable Java libraries. Since we were only able to clone 143 repositories, as they either had accessibility issues (permissions needed) or that the repository was not longer available, we ended up with 351 vulnerabilities.

To identify and extract the applied refactoring operations, we use the RefactoringMiner tool [38, 39], as it is the state-of-the-art for detecting refactoring operations in Java programs, and is widely used in the community [12, 19, 26]. From the 143 cloned repositories, we extract 1,039,263 refactoring operations. In the end, we paired refactoring operations with vulnerability fixes in the same commit (as Refactor-In-Fix). A summary statistic of the data collection is shown in Table 1.

3 EXPLORATORY STUDY RESULTS

We now present the results of our study.

3.1 Prevalence Analysis (RQ1)

To answer RQ1, we grouped our collected dataset by CWEs as shown in Table 2, and refactoring types in Table 3. We make two main observations.

First, we find that 31.9% of the security vulnerabilities incorporated refactoring operations in the fix. Additionally, as shown in Table 2, we find three security weaknesses (i.e., CWE-611 [4], CWE-94 [5], and CWE-352 [3]) that have more vulnerabilities paired that those vulnerabilities that are not paired. In detail, CWE-611 *allows an attacker to access arbitrary files on the system*, while CWE-94 includes *Code injection attacks can lead to loss of data integrity in nearly all cases as the control-plane data injected is always incidental to data recall or writing, as well as result in the execution of arbitrary code.* Finally, CWE-352 is a weakness *where an attacker could effectively perform any operations as the victim. If the victim is an administrator or privileged user, the consequences may include obtaining complete control over the web application - deleting or stealing data, uninstalling the product.* Second, as shown in Table 3, we find that the Extract Method is the most prevalent refactoring type applied in a vulnerability fix. Conversely, the Add Method Annotation is the most common refactoring type that is applied to other commits (not on the vulnerability fix) in the source code.

> **Summary**: To answer RQ1, we find that 31.9% of the security vulnerabilities had incorporated refactoring operations in the fix (i.e., 112 vulnerabilities instances). We also find that the *Extract Method* is most prevalent refactoring type incorporated in a vulnerability fix.

3.2 Refactoring operations associated with vulnerability fixes (RQ2)

To answer RQ2, we use a statistical measure to understand the likelihood that a refactoring operation is performed in a vulnerability fix. Similar to Iammarino et al. [27], we use the odds ratio [22] as a "measure of association", to quantify this relationship. Using the results from RQ1, we calculate the top 10 most frequent refactoring types (527 refactoring operations paired with vulnerabilities and 1,039,263 refactorings from other commits in the code, as shown in Table 3). For our statistical analysis, we test the null hypothesis that: H_1: *The odds of using a certain refactoring operation to refactor a vulnerability fix is strong,* where the *odds ratio* refers to the strength of the relationship, and whether it is likely to be applied to a vulnerability fix (ratio is > 1) or applied as a regular refactoring (ratio is < 1). To statistically validate our hypothesis, we calculated the

Table 4: Proportion of the refactoring operations being applied to a vulnerability fix vs. others: Chi-square test p-values and odds ratio. ($*$: p-value < 0.05)

Refactoring Type	Odds Ratio
Extract Variable	5.678*
Extract Method	5.157*
Rename Method	2.319*
Rename Attribute	2.040*
Rename Parameter	1.662*
Change Attribute Type	1.280
Rename Variable	1.222
Add Method Annotation	0.733*
Change Variable Type	0.725
Change Return Type	0.588*

odds ratio using a 95% confidence interval and use the chi-squared test (*p-value* < 0.05). We use the online odds ratio calculator.[2]

Table 4 reports our findings, from which we make two main observations. First, the two most likely refactoring types to be applied to a vulnerability fix are the Extract Variable (odds ratio of 5.678) and the Extract Method (odds ratio of 5.157). This result is consistent with our finding in RQ1 (cf. Table 3). On the other hand, we find three refactoring types that are more likely to be applied to other commits. The three refactoring types are Add Method Annotation with an odds ratio of 0.733, Change Variable Type with an odds ratio of 0.725, Change Return Type with an odds ratio of 0.588. We find that except for three cases (i.e., Change Attribute Type and Rename Variable, Change Variable Type), all odds ratios are statistically significant.

> **Summary**: Results show that the two most likely refactoring types to be applied to a vulnerability fix is Extract Variable (odds ratio of 5.678) and Extract Method (odds ratio of 5.157). On the other hand, we identified three refactoring types (i.e., Add Method Annotation, Change Variable Type, Change Return Type) that are more likely to be applied as *regular* refactoring operations.

4 THREATS TO VALIDITY

As a short paper, the key threat is the completeness and vigor of the sample size, and a manual validation of refactoring miner shortcomings (i.e., accuracy). We do acknowledge that our current pairing is a very conservative approach, as developers may applied refactoring operations in subsequent commits after the vulnerability fix. However, we argue that our result is substantial and calls attention for researchers to further explore this research avenue.

Another key limitation, we we plan to address in the full study, is to handle vulnerabilities fixed across multiple commits. To mitigate this, we will include these vulnerabilities in the future study.

---

[2]https://www.medcalc.org/calc/odds_ratio.php

5 IMPLICATIONS AND FUTURE PLANS

Our exploratory study delivers various important takeaways and implications to existing practices, and future plans as our roadmap.

Takeaways. With recent works exploring the inducing effects of refactoring actions on software maintenance tasks such as bug fixes, Pull Request acceptance and security-aspects [13, 15, 17, 20, 31, 32], we take a reverse approach to explore further how refactoring is used to help in vulnerability fixes. Our key takeaway message as highlighted in RQ1, is that developers do incorporate refactoring (i.e., 31.9%) operations when applying a vulnerability fix in practice. Furthermore, as shown in RQ2, we distinguish with statistical significance, the odds that a certain refactoring type is paired with a vulnerability fix. This potentially spark research into more dedicated refactoring-aware security vulnerability fix techniques.

Disruption to Current Practices. We believe the adoption of refactoring in security fixes is crucial, especially since it helps developers deliver quicker patches, while maintaining the quality of their code through refactoring. Complementing existing bots like Dependabot [25], automated tooling may help developers understand how vulnerabilities could be fixed in the future, thus improving the vulnerability advisory process. As shown by recent studies, there are lags and issues that exist with fixing vulnerabilities [14, 16, 18, 28, 40].

Future Plans. Motivated by our quantitative results, we plan to execute a more comprehensive study on Java applications, as well as Maven libraries and other library ecosystems. Furthermore, we would like to extend our pairing approach and use other heuristics such as the commit messages and the location at the lines-of-code granularity to achieve more accurate results. We plan our extended study to also include a qualitative component, where we can manually validate how the refactoring types are implemented, and how automation could be achieved. Finally, we plan to conduct a pre and post-assessment experiment, using existing code quality metrics and code smells to validate whether or not vulnerability fixes paired with refactoring operations do impact the code quality.

ACKNOWLEDGMENTS

REFERENCES

[1] 2020. The State of the Octoverse | The State of the Octoverse explores a year of change with new deep dives into developer productivity, security, and how we build communities on GitHub. https://octoverse.github.com/#securing-software. (Accessed on 13/10/2021).

[2] 2021. CVE security vulnerability database. Security vulnerabilities, exploits, references and more. https://www.cvedetails.com/. (Accessed on 02/24/2021).

[3] 2021. CWE - CWE-352: Cross-Site Request Forgery (CSRF) (4.5). https://cwe.mitre.org/data/definitions/352.html. (Accessed on 15/10/2021).

[4] 2021. CWE - CWE-611: Improper Restriction of XML External Entity Reference (4.5). https://cwe.mitre.org/data/definitions/611.html. (Accessed on 15/10/2021).

[5] 2021. CWE - CWE-94: Improper Control of Generation of Code ('Code Injection') (4.5). https://cwe.mitre.org/data/definitions/94.html. (Accessed on 15/10/2021).

[6] 2021. [FIX SECURITY-276] Don't allow open redirect using scheme-rel. URL · jenkinsci/jenkins@2ed0c04. https://github.com/jenkinsci/jenkins/commit/2ed0c046dfbb2003a17df27c53777e72c6eaff25. (Accessed on 13/10/2021).

[7] 2021. GitHub Advisory Database. https://github.com/advisories/. (Accessed on 24/02/2021).

[8] 2021. jenkinsci/jenkins: Jenkins automation server. https://github.com/jenkinsci/jenkins. (Accessed on 13/10/2021).

[9] 2021. Maven Central Repository Search. https://search.maven.org/. (Accessed on 10/16/2021).

[10] 2021. NVD - CVE-2016-3726. https://nvd.nist.gov/vuln/detail/CVE-2016-3726. (Accessed on 13/10/2021).

[11] 2021. NVD - National Vulnerability Database. https://nvd.nist.gov/. (Accessed on 10/10/2021).

[12] Eman Abdullah AlOmar, Anthony Peruma, Christian D Newman, Mohamed Wiem Mkaouer, and Ali Ouni. 2020. On the Relationship Between Devel-oper Experience and Refactoring: An Exploratory Study and Preliminary Results. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, Vol. 20. ACM, 342–349.

[13] Chaima Abid, Marouane Kessentini, Vahid Alizadeh, Mouna Dhouadi, and Rick Kazman. 2020. How Does Refactoring Impact Security When Improving Quality? A Security-Aware Refactoring Approach. *IEEE Transactions on Software Engineering* (2020), 1–1. https://doi.org/10.1109/TSE.2020.3005995

[14] Mahmoud Alfadel, Diego Costa, and Emad Shihab. 2021. Empirical Analysis of Security Vulnerabilities in Python Packages. In *International Conference on Software Analysis, Evolution and Reengineering (SANER)*.

[15] Gabriele Bavota, Bernardino De Carluccio, Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, and Orazio Strollo. 2012. When does a refactoring induce bugs? An empirical study. In *Proceedings - 2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation, SCAM 2012*. 104–113.

[16] Bodin Chinthanet, 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 Software Engineering (ESME)* (2021).

[17] Flávia Coelho, Nikolaos Tsantalis, Tiago Massoni, and Everton Alves. 2021. An Empirical Study on Refactoring-Inducing Pull Requests. 1–12. https://doi.org/10.1145/3475716.3475785

[18] Alexandre Decan, Tom Mens, and Eleni Constantinou. 2018. On the Evolution of Technical Lag in the npm Package Dependency Network. In *Proceedings of the 34th International Conference on Software Maintenance and Evolution (ICSME)*. 404–414.

[19] Massimiliano Di Penta, Gabriele Bavota, and Fiorella Zampetti. 2020. On the Relationship between Refactoring Actions and Bugs: A Differentiated Replication. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020)*. Association for Computing Machinery, 556–567.

[20] Massimiliano Di Penta, Gabriele Bavota, and Fiorella Zampetti. 2020. On the Relationship between Refactoring Actions and Bugs: A Differentiated Replication. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020)*. Association for Computing Machinery, 556–567. https://doi.org/10.1145/3368089.3409695

[21] Xiaoning Du, Bihuan Chen, Yuekang Li, Jianmin Guo, Yaqin Zhou, Yang Liu, and Yu Jiang. 2019. LEOPARD: Identifying Vulnerable Code for Vulnerability Assessment Through Program Metrics. In *International Conference on Software Engineering (ICSE)*.

[22] A. W. Edwards. 1963. The Measure of Association in a 2 × 2 Table. *Journal of the Royal Statistical Society* 1 (1963).

[23] Jiahao Fan, Yi Li, Shaohua Wang, and Tien N. Nguyen. 2020. A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries. In *International Conference on Mining Software Repositories Conference (MSR)*.

[24] Martin Fowler. 1999. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.

[25] GitHub. 2020. Keep all your packages up to date with Dependabot - The GitHub Blog. https://github.blog/2020-06-01-keep-all-your-packages-up-to-date-with-dependabot/. (Accessed on 10/09/2020).

[26] Oumayma Hamdi, Ali Ouni, Mel Ó Cinnéide, and Mohamed Wiem Mkaouer. 2021. A longitudinal study of the impact of refactoring in android applications. *Information and Software Technology* 140 (2021), 106699.

[27] Martina Iammarino, Fiorella Zampetti, Lerina Aversano, and Massimiliano Di Penta. 2019. Self-Admitted Technical Debt Removal and Refactoring Actions: Co-Occurrence or More?. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 186–190. https://doi.org/10.1109/ICSME.2019.00029

[28] Riivo Kikas, Georgios Gousios, Marlon Dumas, and Dietmar Pfahl. 2017. Structure and Evolution of Package Dependency Networks. In *Proceedings of the 14th International Conference on Mining Software Repositories (MSR)*. 102–112.

[29] Raula Gaikovina Kula, Ali Ouni, Daniel M. German, and Katsuro Inoue. 2018. An empirical study on the impact of refactoring activities on evolving client-used APIs. *Information and Software Technology* 93 (2018), 186–199. https://doi.org/10.1016/j.infsof.2017.09.007

[30] Frank Li and Vern Paxson. 2017. A Large-Scale Empirical Study of Security Patches. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[31] Wanwangying Ma, Lin Chen, Yuming Zhou, and Baowen Xu. 2016. Do We Have a Chance to Fix Bugs When Refactoring Code Smells?. In *2016 International Conference on Software Analysis, Testing and Evolution (SATE)*. 24–29. https://doi.org/10.1109/SATE.2016.11

[32] Katsuhisa Maruyama. 2007. SECURE REFACTORING - Improving the Security Level of Existing Code. In *International Conference on Software and Data Technologies (ICSOFT)*. 222–229.

[33] Mitre Corporation. 2018. CVE - Common Vulnerabilities and Exposures (CVE). https://cve.mitre.org/. (Accessed on 20/04/2020).

[34] Mitre Corporation. 2018. CWE - Common Weakness Enumeration. https://cwe.mitre.org/. (Accessed on 20/04/2020).

[35] Haris Mumtaz, Mohammad Alshayeb, Sajjad Mahmood, and Mahmood Niazi. 2018. An empirical study to improve software security through the application of code refactoring. *Information and Software Technology (IST)* 96 (2018), 112–125.

[36] Henning Perl, Sergej Dechand, Matthew Smith, Daniel Arp, Fabian Yamaguchi, Konrad Rieck, Sascha Fahl, and Yasemin Acar. 2015. VCCFinder. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[37] Snyk. 2015. Vulnerability DB. https://snyk.io/vuln. (Accessed on 04/20/2020).

[38] Nikolaos Tsantalis, Ameya Ketkar, and Danny Dig. 2020. RefactoringMiner 2.0. *IEEE Transactions on Software Engineering* (2020), 21 pages. https://doi.org/10.1109/TSE.2020.3007722

[39] Nikolaos Tsantalis, Matin Mansouri, Laleh M. Eshkevari, Davood Mazinanian, and Danny Dig. 2018. Accurate and Efficient Refactoring Detection in Commit History. In *International Conference on Software Engineering (ICSE)*. 483–494.

[40] Ahmed Zerouali, Eleni Constantinou, Tom Mens, Gregorio Robles, and Jesus Gonzalez-Barahona. 2018. An Empirical Analysis of Technical Lag in npm Package Dependencies. In *Proceedings of the 17th International Conference on Software Reuse (ICSR)*. 95–110.