Assisted Discovery of Software Vulnerabilities

Nuthan Munaiah
Rochester Institute of Technology
Department of Software Engineering
Rochester, NY, USA
nm6061@rit.edu

ABSTRACT

As more aspects of our daily lives rely on technology, the software that enables the technology must be secure. Developers rely on practices such as threat modeling, static and dynamic analyses, code review, and fuzz and penetration testing to engineer secure software. These practices, while effective at identifying vulnerabilities in software, are limited in their ability to describe the potential reasons for the existence of vulnerabilities. In order to overcome this limitation, researchers have proposed empirically validated metrics to identify factors that may have led to the introduction of vulnerabilities in the past. Developers must be made aware of these factors so that they can proactively consider the security implications of each line of code that they contribute. The goal of our research is *to assist developers in engineering secure software by providing a technique that generates scientific, interpretable, and actionable feedback on security as the software evolves.* In this paper, we provide an overview of our proposed approach to accomplish this research goal through a series of three research studies in which we (1) systematize the knowledge on vulnerability discovery metrics, (2) leverage the metrics to generate feedback on security, and (3) implement a framework for providing automatically generated feedback on security using code reviews as a medium.

ACM Reference Format:
Nuthan Munaiah. 2018. Assisted Discovery of Software Vulnerabilities. In *ICSE '18 Companion: 40th International Conference on Software Engineering, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages.
https://doi.org/10.1145/3183440.3183453

1 INTRODUCTION

In the process of engineering secure software, developers leverage a plethora of processes, techniques, and tools such as threat modeling, static and dynamic analyses, unit/integration/fuzz/penetration testing, and code reviews. These practices, while effective at identifying vulnerabilities in software, are limited in their ability to describe the potential reasons for the existence of vulnerabilities.

As researchers propose empirically validated metrics aimed at characterizing historical vulnerabilities, the factors that may have led to the introduction of these vulnerabilities emerge. Developers must be made aware of these factors to help them proactively consider security implications of the code that they contribute. In other words, we want developers to think like an attacker (i.e. inculcate an *attacker mindset*) to proactively discover vulnerabilities.

Over the last decade, several metrics [1, 10–12, 19, 20, 22, 24, 26–34] have been proposed to assist developers in discovering vulnerabilities. However, the adoption of these metrics in mainstream software engineering has been limited owing to concerns such as the high frequency of false positive from predictive models that use the metrics as features, the granularity at which the metrics operate, and lack of interpretable and actionable intelligence from the metrics [13]. These concerns may be mitigated by considering the empirical properties that the metrics embody beyond their predictive capabilities. We must consider the metrics not as mere features in a vulnerability prediction model but as agents of feedback on security. In other words, we must ask ourselves *what is the metric telling us?* and *what can we ask developers to do?* In essence, we must *humanize* the metrics such that they can communicate the reasoning for a source code entity to be regarded as vulnerable. In interpreting the feedback that a metric provides, developers gain awareness of the potential factors that lead to vulnerabilities thus aiding in inculcating an attacker mindset.

The goal of our research is *to assist developers in engineering secure software by providing a technique that generates scientific, interpretable, and actionable feedback on security as the software evolves.*

Our expected contributions are:

- Systematization of metrics that are known to be effective in assisting developers to discover vulnerabilities.
- Development of a novel approach to providing security feedback to developers.
- Reference implementation of a platform that leverages the vulnerability discovery metrics to provide developers with automatically generated feedback on security.

2 RELATED WORK

We regard two streams of research to be related to our goal: vulnerability discovery approaches and feedback in software engineering.

2.1 Vulnerability Discovery Approaches

Over the years, numerous approaches have been proposed to assist in the discovery of vulnerabilities. The approaches span the spectrum of proactive assessment of software security from using testing, static, dynamic or hybrid analysis, (automatic) patching [25] to leveraging metrics to characterize and predict vulnerabilities [4]. Among the vulnerability discovery approaches, the approach that relies on metrics is the only one that affords intelligence beyond the mere discovery of vulnerabilities. In addition to assisting

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05…$15.00
https://doi.org/10.1145/3183440.3183453

in discovering vulnerabilities, metrics can help developers understand the potential reasons for the existence of vulnerabilities. For instance, consider the metric *churn*, which is a measure of the frequency with which source code is modified. Researchers [34] have shown that vulnerabilities tend to exist in source code that has high churn. Therefore, *churn*, in addition to being useful in discovering vulnerabilities, can be used to inform developers to be cautious when working with source code that has been frequently modified.

The literature on vulnerability discovery metrics is largely varied with researchers evaluating metrics defined on product or process using a plethora of metric validation criteria in the context of subjects of study with varying attributes (size, history, domain, language, and open/closed source). We found nine authors [1, 10–12, 19, 20, 22, 24, 26–34] who have proposed and/or evaluated over one hundred vulnerability discovery metrics.

In almost all vulnerability discovery metric studies, the effectiveness of the metrics was ultimately evaluated in terms of the performance of a machine learning model that used the metrics as features. In doing so, researchers tend to condense the utility of the metrics to mere precision and recall of a machine learning model that uses the metric to predict vulnerabilities. We argue that the empirical properties (such as interpretability and actionability) that the metric embodies is as important, if not more, as its effectiveness in a vulnerability prediction model. These empirical properties can be used to provide developers with actionable intelligence about security as they engineer software.

2.2 Feedback in Software Engineering

Developers receive automatically generated feedback from a variety of tools such as integrated development environments that aggregate errors and warnings from compiling, building, testing, and/or statically analyzing source code. In addition to tools, practices such as code review create an opportunity to receive feedback from other developers as well. There are many studies in prior literature in which feedback during software engineering has been evaluated to characterize effective feedback. We can leverage the recommendations from these prior studies to provide feedback on security in a way that it useful to developers.

We can categorize prior studies on feedback into two categories: exploration [3, 5, 7] and experience [8, 23]. In both categories, researchers made recommendations for tool designers, specifically, tools that provide automatically generated feedback to developers. In aggregate, developers preferred feedback when it is easy to understand, informative, and actionable and provided in context so as to not disrupt their thought process. As for tools that provided the feedback, developers preferred tools to (1) be scalable, (2) integrate with existing workflow, (3) support collaboration, (4) allow customization, (5) provide a way for developers to affect the feedback provided, and (6) produce fewer false positives.

3 PRIOR WORK

The research goal proposed in this paper is inspired by our prior work in the areas of software metrics [14, 16], vulnerability discovery [15, 18], and requirements engineering [17]. We have showed that bugs and vulnerabilities are empirically dissimilar groups and that experience reviewing bugs may not be sufficient to uncover vulnerabilities [14]. We have evaluated the validity of a commonly used measure of vulnerability severity—the Common Vulnerability Scoring System (CVSS) base score—to show that CVSS base score underestimates the reward amount provided for responsible disclosure of vulnerabilities [16]. We have proposed two vulnerability discovery metrics based on the attack surface representation of software and showed that these metrics are associated with historical vulnerabilities in FFmpeg and Wireshark [15]. We have characterized the linguistic cues in code reviews that missed a vulnerability in the past [18]. We have also proposed a novel approach to identify security requirements in a domain independent way [17].

4 RESEARCH APPROACH

The approach to accomplish our research goal will be guided by three research studies. A pictorial overview of the workflow that we will implement as part of our research is shown in Figure 1.

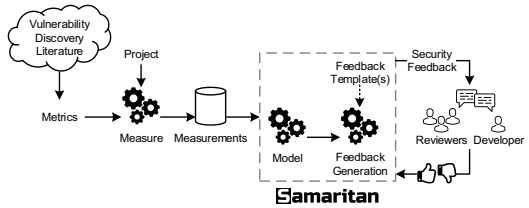

Figure 1: Pictorial overview of the workflow that we will implement to accomplish our research goal

4.1 Study 1: Systematization of Vulnerability Discovery Knowledge

Almost two decades ago, Fenton and Neil [2] recognized that the purpose of software metrics was to support the process of decision making in software development and that existing metrics must be used to build tools to support such decision making. The metrics to support proactive discovery of vulnerabilities exist but are largely scattered across several publications. An essential step in using vulnerability discovery metrics to generate actionable feedback on security is to systematically review existing literature to identify (1) the metrics that have been proposed and/or evaluated to discover vulnerabilities, (2) the degree of validity of the metrics, and (3) the challenges in collecting the metrics (if any).

Research Questions

- What metrics have been proposed to understand vulnerabilities?
- What is the degree to which the metrics have been validated?
- What are the challenges in collecting the metrics?
- Are the empirical properties of the metrics consistent across projects?

Evaluation Approach

We will follow the protocol prescribed by Kitchenham and Brereton [6] in conducting the systematic literature review of papers that have proposed and/or empirically evaluated vulnerability discovery metrics. The relevant studies identified by the application of the protocol will be subject to further analysis to identify the metrics that have been proposed to discovery vulnerabilities. In assessing

the degree of validity, we will use the summary of metric validation criteria from prior work by Meneely *et al.* [9] as a checklist. If a metric appears in multiple studies, we will aggregate the validation criteria to express the degree of validity of the metric as a whole.

We will implement the method associated with a metric and use it to collect the metric from several subjects of study to identify any implementation challenges. We will begin by identifying a set of subjects of study that will be targeted. The subjects of study will be identified using predetermined criteria such as open source, actively maintained and well documented project that leverages sound software engineering practices. In implementing the methods to collect the metrics from multiple subjects of study that may differ in application domain, programming language, size, architecture, etc., we can describe, based on actual experience, the limitations (if any) of the metrics. The metrics collected from the various subjects of study will then be compared to determine the consistency (or lack thereof) among the empirical properties (such as distributions, descriptive statistics, and correlation among the metrics) of the metrics. Consistency in the empirical properties of a metric across subjects of study will strengthen the support for its generalizability.

In the software engineering domain, continuous-valued metrics tend to follow a variety of distributions [21]. Therefore, we cannot assume that the vulnerability discovery metrics follow a normal distribution. We will rely on nonparametric statistical tests when analyzing the vulnerability discovery metrics. We will use Mann-Whitney-Wilcoxon (MWW) test to assess if two samples of metric values are from the same distribution, Cliff's δ to assess the extent to which two samples of metric values are differently distributed, and Spearman's ρ to assess if two metrics are correlated.

4.2 Study 2: Feedback in Software Engineering

Traditionally, the utility of vulnerability discovery metrics has been as features in a prediction model. We argue that the role of the metrics must be broadened to leverage the intelligence that the metrics afford to provide developers with natural language feedback on security. The objective in this study it to understand the types (i.e. what), instances (i.e. when), and ways (i.e. how) in which developers receive feedback as they engineer software. The insights gained in this study will enable us to develop techniques to generate automatic feedback on security that is effective in assisting developers discover vulnerabilities. The comment from a participant that using "real words" would be a start toward improving the understandability of warnings from static analyzers [5] is an example of the kind of insight we hope to gain from this study.

Research Questions

- What are the ways in which developers receive feedback?
- What are the elements of effective feedback?

Evaluation Approach

In this research study, developers will be interviewed using a semi-structured process to understand the role of automated feedback in their workflow. As a first step toward identifying potential interview participants, we will thoroughly review the documentation pertaining to the subjects of study identified in the previous research study to understand the software development process employed by each subject. We will then identify one or more subjects of study from which a group of developers may be sampled to be potential participants in a semi-structured interview designed to elicit information about experience with receiving feedback. Some of the key questions that will be included in the interview are • What are the different ways in which you receive feedback? • What do you think are the essential characteristics of high quality feedback? • What are your opinions on a system that provides automated feedback on your tasks? • What are your opinions on a system that provides feedback on your tasks but in a way that other developers reviewing the task can also see the feedback? The interview responses will inform the design and implementation of our approach to providing automated feedback on security.

4.3 Study 3: Code Reviews for Automated Feedback on Security

The validity of metrics is of utmost importance to researchers who propose them. However, developers' perception of the usefulness of a metric is an important factor in determining its effectiveness in mainstream software engineering. The objective in this study is to demonstrate, through a reference implementation, that vulnerability discovery metrics from prior literature can be used to automatically generate actionable feedback on security. The reference implementation, called Samaritan, will be integrated with a code review system to provide automatically generated feedback on security in the context of a change. We argue that having an attacker mindset is an essential skill in engineering secure software and that the conversation the automatically generated feedback may spur contributes to the development of the skill.

Research Questions

- Can existing vulnerability discovery metrics be used to provide useful feedback on security?
- Did the automated feedback prompt developers to have a conversation about security?
- What reasons do developers cite when saying feedback was not helpful?

Evaluation Approach

In this research study, the data used to address the research questions will be collected using Samaritan. The automatically generated feedback on security will be provided to the developer in the context of a change being reviewed. The feedback will be accompanied by a set of links allowing the developer to provide comments. The comments will be used to improve the feedback we provide. In addition to the links, we will also monitor the code review conversation after the automatically generated feedback on security has been provided to ascertain if the feedback prompted the developers to have a conversation about security.

5 TIMELINE

The research goal and approach described in this paper is condensed from my Ph.D. Proposal. I successfully defended the proposal on November 20, 2017.

In terms of the research studies, we propose to conduct Study 1 and Study 2 in parallel. We plan to submit papers describing our findings from Studies 1 and 2 to the 12th International Symposium on Empirical Software Engineering and Measurement (ESEM) and 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE) conferences, respectively. We will begin Study 3

as soon as we have insights from Studies 1 and 2 and plan to submit a paper describing the results from Study 3 to the Conference on Human Factors in Computing Systems (CHI) 2019.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Andrew Meneely, for his invaluable feedback, which has been instrumental in motivating me to pursue the research goal proposed in this paper.

REFERENCES

[1] Istehad Chowdhury and Mohammad Zulkernine. 2011. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. *Journal of Systems Architecture* 57, 3 (2011), 294 – 313. DOI:http://dx.doi.org/https://doi.org/10.1016/j.sysarc.2010.06.003 Special Issue on Security and Dependability Assurance of Software Architectures.

[2] Norman E. Fenton and Martin Neil. 2000. Software Metrics: Roadmap. In *Proceedings of the Conference on The Future of Software Engineering (ICSE '00)*. ACM, New York, NY, USA, 357–370. DOI:http://dx.doi.org/10.1145/336512.336588

[3] Sylvie L Foss and Gail C Murphy. 2015. Do Developers Respond to Code Stability Warnings?. In *Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering (CASCON '15)*. IBM Corp., Riverton, NJ, USA, 162–170. http://dl.acm.org/citation.cfm?id=2886444.2886469

[4] Seyed Mohammad Ghaffarian and Hamid Reza Shahriari. 2017. Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey. *ACM Comput. Surv.* 50, 4 (aug 2017), 56:1—-56:36. DOI:http://dx.doi.org/10.1145/3092566

[5] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why Don't Software Developers Use Static Analysis Tools to Find Bugs?. In *2013 35th International Conference on Software Engineering (ICSE)*. 672–681. DOI:http://dx.doi.org/10.1109/ICSE.2013.6606613

[6] Barbara Kitchenham and Pearl Brereton. 2013. A systematic review of systematic review process research in software engineering. *Information and Software Technology* 55, 12 (2013), 2049–2075. DOI:http://dx.doi.org/10.1016/j.infsof.2013.07.010

[7] L Layman, L Williams, and R S Amant. 2007. Toward Reducing Fault Fix Time: Understanding Developer Behavior for the Design of Automated Fault Detection Tools. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. 176–185. DOI:http://dx.doi.org/10.1109/ESEM.2007.11

[8] Chris Lewis, Zhongpeng Lin, Caitlin Sadowski, Xiaoyan Zhu, Rong Ou, and E James Whitehead Jr. 2013. Does Bug Prediction Support Human Developers? Findings from a Google Case Study. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 372–381. http://dl.acm.org/citation.cfm?id=2486788.2486838

[9] Andrew Meneely, Ben Smith, and Laurie Williams. 2013. Validating Software Metrics: A Spectrum of Philosophies. *ACM Trans. Softw. Eng. Methodol.* 21, 4, Article 24 (Feb 2013), 28 pages. DOI:http://dx.doi.org/10.1145/2377656.2377661

[10] A. Meneely, H. Srinivasan, A. Musa, A. R. Tejeda, M. Mokary, and B. Spates. 2013. When a Patch Goes Bad: Exploring the Properties of Vulnerability-Contributing Commits. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*. 65–74. DOI:http://dx.doi.org/10.1109/ESEM.2013.19

[11] Andrew Meneely and Laurie Williams. 2009. Secure Open Source Collaboration: An Empirical Study of Linus' Law. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS '09)*. ACM, New York, NY, USA, 453–462. DOI:http://dx.doi.org/10.1145/1653662.1653717

[12] Andrew Meneely and Laurie Williams. 2010. Strengthening the Empirical Analysis of the Relationship Between Linus' Law and Software Security. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '10)*. ACM, New York, NY, USA, Article 9, 10 pages. DOI:http://dx.doi.org/10.1145/1852786.1852798

[13] Patrick Morrison, Kim Herzig, Brendan Murphy, and Laurie Williams. 2015. Challenges with Applying Vulnerability Prediction Models. In *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security (HotSoS '15)*. ACM, New York, NY, USA, Article 4, 9 pages. DOI:http://dx.doi.org/10.1145/2746194.2746198

[14] Nuthan Munaiah, Felivel Camilo, Wesley Wigham, Andrew Meneely, and Meiyappan Nagappan. 2017. Do bugs foreshadow vulnerabilities? An in-depth study of the chromium project. *Empirical Software Engineering* 22, 3 (01 Jun 2017), 1305–1347. DOI:http://dx.doi.org/10.1007/s10664-016-9447-3

[15] Nuthan Munaiah and Andrew Meneely. 2016. Beyond the Attack Surface: Assessing Security Risk with Random Walks on Call Graphs. In *Proceedings of the 2016 ACM Workshop on Software PROtection (SPRO '16)*. ACM, New York, NY, USA, 3–14. DOI:http://dx.doi.org/10.1145/2995306.2995311

[16] Nuthan Munaiah and Andrew Meneely. 2016. Vulnerability Severity Scoring and Bounties: Why the Disconnect?. In *Proceedings of the 2Nd International Workshop on Software Analytics (SWAN 2016)*. ACM, New York, NY, USA, 8–14. DOI:http://dx.doi.org/10.1145/2989238.2989239

[17] N. Munaiah, A. Meneely, and P. K. Murukannaiah. 2017. A Domain-Independent Model for Identifying Security Requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 506–511. DOI:http://dx.doi.org/10.1109/RE.2017.79

[18] Nuthan Munaiah, Benjamin S. Meyers, Cecilia O. Alm, Andrew Meneely, Pradeep K. Murukannaiah, Emily Prud'hommeaux, Josephine Wolff, and Yang Yu. 2017. *Natural Language Insights from Code Reviews that Missed a Vulnerability*. Springer International Publishing, Cham, 70–86. DOI:http://dx.doi.org/10.1007/978-3-319-62105-0_5

[19] Stephan Neuhaus and Thomas Zimmermann. 2009. The Beauty and the Beast: Vulnerabilities in Red Hat's Packages. In *Proceedings of the 2009 USENIX Annual Technical Conference (USENIX ATC) (USENIX ATC '09)*. https://www.usenix.org/legacy/event/usenix09/tech/full

[20] Stephan Neuhaus, Thomas Zimmermann, Christian Holler, and Andreas Zeller. 2007. Predicting Vulnerable Software Components. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*. ACM, New York, NY, USA, 529–540. DOI:http://dx.doi.org/10.1145/1315245.1315311

[21] C. Ravindranath Pandian and Murali Kumar S K. 2015. *Simple Statistical Methods for Software Engineering: Data and Patterns*. CRC Press.

[22] Henning Perl, Sergej Dechand, Matthew Smith, Daniel Arp, Fabian Yamaguchi, Konrad Rieck, Sascha Fahl, and Yasemin Acar. 2015. VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. ACM, New York, NY, USA, 426–437. DOI:http://dx.doi.org/10.1145/2810103.2813604

[23] C Sadowski, J v. Gogh, C Jaspan, E Söderberg, and C Winter. 2015. Tricorder: Building a Program Analysis Ecosystem. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 598–608. DOI:http://dx.doi.org/10.1109/ICSE.2015.76

[24] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen. 2014. Predicting Vulnerable Software Components via Text Mining. *IEEE Transactions on Software Engineering* 40, 10 (Oct 2014), 993–1006. DOI:http://dx.doi.org/10.1109/TSE.2014.2340398

[25] Hossain Shahriar and Mohammad Zulkernine. 2012. Mitigating Program Security Vulnerabilities: Approaches and Challenges. *ACM Comput. Surv.* 44, 3, Article 11 (Jun 2012), 46 pages. DOI:http://dx.doi.org/10.1145/2187671.2187673

[26] Yonghee Shin. 2011. *Investigating Complexity Metrics As Indicators of Software Vulnerability*. Ph.D. Dissertation. North Carolina State University. Advisor(s) Williams, Laurie and Vouk, Mladen. AAI3442705.

[27] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne. 2011. Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities. *IEEE Transactions on Software Engineering* 37, 6 (Nov 2011), 772–787. DOI:http://dx.doi.org/10.1109/TSE.2010.81

[28] Yonghee Shin and Laurie Williams. 2008. An Empirical Model to Predict Security Vulnerabilities Using Code Complexity Metrics. In *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '08)*. ACM, New York, NY, USA, 315–317. DOI:http://dx.doi.org/10.1145/1414004.1414065

[29] Yonghee Shin and Laurie Williams. 2008. Is Complexity Really the Enemy of Software Security?. In *Proceedings of the 4th ACM Workshop on Quality of Protection (QoP '08)*. ACM, New York, NY, USA, 47–50. DOI:http://dx.doi.org/10.1145/1456362.1456372

[30] Yonghee Shin and Laurie Williams. 2011. An Initial Study on the Use of Execution Complexity Metrics As Indicators of Software Vulnerabilities. In *Proceedings of the 7th International Workshop on Software Engineering for Secure Systems (SESS '11)*. ACM, New York, NY, USA, 1–7. DOI:http://dx.doi.org/10.1145/1988630.1988632

[31] Yonghee Shin and Laurie Williams. 2013. Can traditional fault prediction models be used for vulnerability prediction? *Empirical Software Engineering* 18, 1 (01 Feb 2013), 25–59. DOI:http://dx.doi.org/10.1007/s10664-011-9190-8

[32] J. Walden, J. Stuckman, and R. Scandariato. 2014. Predicting Vulnerable Components: Software Metrics vs Text Mining. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*. 23–33. DOI:http://dx.doi.org/10.1109/ISSRE.2014.32

[33] Awad Younis, Yashwant Malaiya, Charles Anderson, and Indrajit Ray. 2016. To Fear or Not to Fear That is the Question: Code Characteristics of a Vulnerable Function with an Existing Exploit. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy (CODASPY '16)*. ACM, New York, NY, USA, 97–104. DOI:http://dx.doi.org/10.1145/2857705.2857750

[34] T Zimmermann, N Nagappan, and L Williams. 2010. Searching for a Needle in a Haystack: Predicting Security Vulnerabilities for Windows Vista. In *2010 Third International Conference on Software Testing, Verification and Validation*. 421–428. DOI:http://dx.doi.org/10.1109/ICST.2010.32