Classifying Web Exploits with Topic Modeling

Jukka Ruohonen
University of Turku, Finland
juanruo@utu.fi

***Abstract*—This short empirical paper investigates how well topic modeling and database meta-data characteristics can classify web and other proof-of-concept (PoC) exploits for publicly disclosed software vulnerabilities. By using a dataset comprised of over 36 thousand PoC exploits, near a 0.9 accuracy rate is obtained in the empirical experiment. Text mining and topic modeling are a significant boost factor behind this classification performance. In addition to these empirical results, the paper contributes to the research tradition of enhancing software vulnerability information with text mining, providing also a few scholarly observations about the potential for semi-automatic classification of exploits in the existing tracking infrastructures.**

***Index Terms*—Text mining, LDA, EDB, NVD, CVE, CVSS**

I. Introduction

This short empirical paper investigates whether and how well topic modeling and database meta-data characteristics can classify web and other proof-of-concept (PoC) exploits for publicly disclosed software vulnerabilities. To provide a tentative definition, PoC exploits are software implementations for concretely demonstrating the existence of a software vulnerability, which is a software defect with security implications. While underlining that PoC exploits may perform poorly for actual exploitation [1], this definition also aligns and connects the scholarly vulnerability research tradition to the field of software repository mining [2], [3]. However, the existing research in this tradition has mostly focused on software vulnerabilities. This gap in the literature may yield biases because not all vulnerabilities are exploitable as such; hence, not all vulnerabilities are relevant for repository mining.

The research on automatic solutions for defect labeling and related bug "triaging" aspects (e.g., [4], [5]) provide a further motivation. Exploits are disclosed and published on numerous different Internet platforms—including but not limited to security databases, mailing lists, blogs, and bug trackers. Although systematic archiving has recently gained traction, a human is still required to evaluate and classify exploits gathered from such diverse sources. This human-side has largely been also ignored in previous research. For instance, many proposals for unifying vulnerabilities, exploits, and related security information with different abstract frameworks (see [6], [7], [8], [9]) do not discuss how the unification should work in practice. In other words, proposing a yet another ontology or a further database schema is arguably unfruitful when there are well-known problems for managing even the current volume of vulnerabilities and exploits (cf. [10], [11]). In addition to benefits for manual exploit triaging, automation improvements are beneficial for large-scale harvesting of exploits with web crawling and related data collection techniques.

The noteworthy related works are the text mining applications for examining and enriching software vulnerability and related software security information [12], [13], [14], [15]. Given this background, the paper examines the natural language characteristics of the exploits archived to the open data Exploit Database (henceforth, EDB) [16], which is currently the likely most comprehensive database for archiving exploits. According to surveys, it is also preferred by many contemporary exploit developers [17]. For examining the database and the exploits archived, a small classification experiment is conducted for evaluating how well existing EDB meta-data characteristics and text mining can classify exploits. For brevity, exploits for web application vulnerabilities are classified in relation to other software types targeted by exploits. This empirical classification experiment is prepared in Section II. Results and discussion follow in Sections III and IV.

II. Materials and Methods

The raw datasets contains all exploits that were archived to EDB in early October 2016. While (mis)selection of a corpora is a typical reason for construct validity biases in topic modeling applications [18], the analysis in this paper exploits a "natural bias" in terms of the persistence of different web vulnerabilities [13], [19], and, hence, web exploits. That is to say, the intention is to classify web and other exploits based on meta-data and natural language characteristics. The meta-data characteristics refer to the database schema used in EDB, which is maintained semi-manually for categorization and other purposes, whereas the latter are derived by text mining techniques for which pre-processing is also required.

A. Pre-processing

The PoC exploits archived to EDB are in raw text format. In many cases, the exploits refer to self-contained programming code snippets that demonstrate the proof-of-concepts for exploitation. The programming languages used include C, Python, Perl, Ruby, and other scripting languages. Sparse code comments are also typically present. However, the issue of separating exploit code from natural language characteristics is not comparable to the well-known (e.g., [20]) problem of separating code from code comments. Instead, the issue resembles more the genre of software artifact retrieval from heterogeneous semi-structured sources [21], [22]. In other words, the exploits archived refer to free-form text entries that contain information beyond programming details. For instance, attribution credits, disclosure details, and remediation

instructions may be included in the archives, while the actual exploitation code may amount only to a few lines of code.

For these reasons, each archived entry is pre-processed without explicitly attempting to separate the non-code text excerpts from the programming language code. Implicitly, however, a partial separation is done during the six pre-processing steps subsequently enumerated.

1) Due to the aforementioned issue related to separating exploit code from non-code, it is not reasonable to consider tokenization techniques such as text splitting according to the so-called "CamelCase" or "under_score" notations [23], [24], [25], or according to the analogous "slash/notation" [26] and "dot.notation" [27] commonly used for malware labeling, for instance. Thus, each raw exploit entry is first tokenized simply by using the `word_tokenize` function in the *de facto* text ming library for Python [28]. These tokens are used for inputs to the subsequent pre-processing checks and routines.
2) Tokens less than four characters in length are subsequently excluded alongside with tokens with length longer than 20 characters. While the former exclusion criterion is commonly used in text mining (e.g., [25]), the latter is specific to the context, typically capturing and excluding tokens referring to large hexadecimal payloads used in exploits for buffer overflow vulnerabilities.
3) The next step involves classifying all tokens into words and non-words (henceforth, *words* and *terms*). Python bindings [29] for a common open source English dictionary [30] are used for this classification of the tokens.
4) The WordNet-based NLTK function `lemmatize` is then used for grouping similar words (but not terms) according to their dictionary forms. Due to the specific context of exploits, lemmatization is preferable to conventional stemming algorithms. For instance, exemplifying typical over-stemming issues [24], the word `vulnerability` is undesirably stemmed to `vulner`.
5) A few stopwords are removed from the collections of lemmatized words (and non-lemmatized terms as a double-check). NLTK's default stopword list is used.
6) After all exploits have been processed, those words and terms are excluded that have a frequency less than 20 across all exploits observed. While this again commonly used (e.g., [31]) frequency-based pre-processing criterion excludes a substantial amount of words and terms, the choice is partially justified by practical aspects related to computational memory requirements. Finally, seven exploits (alongside the corresponding words and terms) were excluded from the analysis because either no words or no terms were present after the exclusion.

These six steps result two frequency matrices for words and terms; each row denotes an exploit and each column an unigram (a word or a term), such that the (i, j):th entry contains the frequency of the j:th unigram for the i:th observed exploit among the $n = 36,184$ exploits observed. In total, $4,844$ words and $7,995$ terms were identified from these exploits with the sixfold pre-processing routine. Although the separation is only implicit, the two matrices can be reflected against the lower entropy that programming code typically exhibits compared to natural language [20]. The same likely applies also to general technical terms and security industry slang typically used in the sample. Therefore, separate topics are extracted from the two unigram matrices, and the topics computed are used as separate covariates for classification.

B. Topics

The topics are extracted from the two frequency matrices by using the latent Dirichlet allocation (LDA) method developed by Blei and associates [32]. As this method is a modern classic in computer science, which is accompanied by surveys and hands-on guides for software engineering [23] and related fields [33]—including detailed information about the use for software vulnerabilities [13]—a few practical remarks are more relevant than a brief summary of the LDA method itself.

In a nutshell, the method is based on Bayesian mixture modeling in which a topic is a multinomial probability distribution over words from a finite vocabulary [31]. As Dirichlet processes are used for deriving the prior distributions in LDA, the first practical concern for applied research relates to the two parameters governing the per-exploit topic distribution and per-topic "word-or-term" distribution. Although these parameters should arguably reflect genuine prior information, computational routines and rules-of-thumb are commonly used in practice [23], [33]. In this paper, likewise, the estimation routine in the R package used for computation [34] is adopted for determining the parameters (i.e., the package's default settings are used). The second issue for applied work relates to the fact that a single exploit is often characterized by multiple dominant topics, which entails a choice over a threshold scalar for cutting off irrelevant topics [35]. Given the classification purposes of this paper, each exploit is assigned to the most dominant topic with the highest membership rate. The third and final concern relates to the number of topics to extract. Because the topics assigned for each exploit are used for classification, which implies that interpretation and construct validity are lesser concerns [33], the LDA is computed, for the term and frequency matrices separately, by restricting the number of topics to $k = 5, 10, 20, 30, 40, 50$. For each of these restrictions, the corresponding dominant (word and term) topic assignments vectors are used to build separate classifiers.

C. Classification

The classification experiment is computed by using an R implementation [36], in combination with the *caret* package [37], for the tree-based random forest classifier. Rather than summarizing this decision tree method, it is again more relevant to focus on the practical aspects, starting from the operationalization of the two binary-valued response metrics.

1) Responses: The binary-valued response metrics for the classification are constructed from two meta-data schemas provided in EDB. The first is based on the high-level categorization into denial-of-service (dos), local, remote, and web

exploits. As can be observed from Fig. 1, about 58 percent of all exploits observed are located in the web-category. For the first response metric, these web exploits are used as a reference category ("true"), while the remaining three categories are grouped into a single group of non-web exploits ("false").

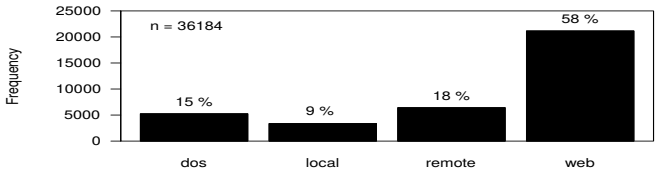

Fig. 1. Exploit Categories (EDB's meta-data)

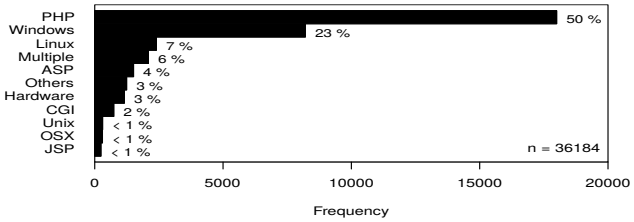

Fig. 2. Exploit Platforms (EDB's meta-data)

The second response metric is based on the target platforms to which the EDB community additionally groups the exploits archived. In total, as many as 51 distinct platforms have been used for this grouping, although the most common platforms show no big surprises. By regrouping less common platforms into a heterogeneous group of "Others", the ten most frequent platforms are illustrated in Fig. 2. Reflecting the large amount of web exploits in the earlier Fig. 1, web sites and web applications written in PHP have been the most frequent targets for the exploits archived, followed by software running on Windows and Linux, respectively. Analogous to the first response metric, the PHP platform is taken as a reference category against which the remaining platforms are compared.

2) Covariates: The covariate metrics enumerated in Table I are included for classifying web and PHP exploits. While the first two metrics are based on the dominant topic assignment vectors (see Section II-B), the remaining ones are all based on EDB's meta-data. These meta-data covariates can be categorized into four high-level abstract groups.

1) The first group includes three binary-valued metrics related to the evaluation work done by the EDB community for the exploits archived. Like all software, exploits require testing. *Verified*, therefore, scores one in case the exploit has been verified to work. *Application* is likewise a dummy variable for recording whether the archive contains also the vulnerable application available for download. Reflecting the occasional use of video demonstrations during software vulnerability disclosure [38], the third dummy variable records the cases for which a screenshot is available for download.
2) The second group contains three metrics explicitly or implicitly related to other databases. Akin to the "vulnerability claims" made by the orchestrators of the National Vulnerability Database (NVD) [3], per-exploit references made by the EDB community to the Open Source Vulnerability Database (OSVDB) are counted with the metric named *OSVDB references*. (It can be also remarked that OSVDB used to be the only comprehensive open database containing some information about exploits [39], but this database was shutdown in 2016 due to maintenance and other issues [11]. This does not affect the results reported, however.) For each exploit, the amount of references to Common Vulnerabilities and Exposures (CVEs) are likewise counted via the metric labeled *CVE references*. While these reference counters are implicit in the sense that no attempts are made to validate EDB's claims, the Common Vulnerability Scoring System (CVSS) is explicitly used to approximate the average severity of the vulnerabilities targeted by those exploits that have CVE references. *Mean CVSS* thus denotes the arithmetic mean of so-called base CVSS scores, which range from zero (negligible) to ten (catastrophic), and which are explicitly fetched from NVD.
3) The third group contains two calendar time metrics, *Month* and *Year*. Due to the well-known data quality issues affecting vulnerability (e.g., [10], [39]) and exploit archives, interpretation should be done with care— but, nevertheless, these two metrics convey the approximate year and month at which a given exploit was first published in the wild according to EDB's evaluation.
4) The final set of covariates expands to 30 dummy variables for the most productive exploit developers. Thus, the first dummy variable scores one for all exploits authored by the most productive developer, and so forth. When ranked by the number of exploits authored, the resulting list largely confirms existing empirical observations [10], [40], [41]. For instance, the Metasploit project and Luigi Auriemma are the top-two most productive authors, but the list contains also names such as rgod, r0t, and the Google Security Research group.

III. Results

Estimation is carried by fitting 2×7 models for classifying web and PHP exploits (see Section II-C1). In terms of practical estimation concerns, PHP exploits yield a precisely balanced set (see Fig. 2). A sufficiently balanced set is available also for web exploits (see Fig. 1). For comparing how much information is gained from the dominant topic assignments (see Section II-B), classification of the two response metrics is first done by excluding the topic assignment vectors, and then fitting separate models for the varying number of topics extracted via LDA. As the topic assignments vectors for words and terms do not notably correlate with each other (see Table II), both vectors are included in these separate models. A 5-fold cross-validation is used for training. To maintain a sensible scenario for predicting new data, the

TABLE I
COVARIATES

#	Mnemonic name	Description
1.	*Term topic*	One for the most dominant term-based topic characterizing the exploit; zero otherwise.
2.	*Word topic*	One for the most dominant word-based topic characterizing the exploit; zero otherwise.
3.	*Verified*	One if the EDB community has verified the exploit; zero otherwise.
4.	*Application*	One if the vulnerable application is available for download; zero otherwise.
5.	*Screenshot*	One if a screenshot is provided for a demonstration or other purposes; zero otherwise.
6.	*OSVDB references*	The number of OSVDB references or zero for no such references.
7.	*CVE references*	The number of CVE references or zero for the absence of CVE references.
8.	*Mean CVSS*	The mean of CVSS base scores for all CVE references (or zero for no references).
9.	*Publication year*	The year at which the exploit was first published according to EDB.
10.	*Publication month*	The month at which the exploit was first published according to EDB.
11. – 40.	*Top developers*	One if the author of the exploit is among the "top-30" developers; zero otherwise.

TABLE II
TOPIC ASSIGNMENT CORRELATIONS (WORDS AND TERMS)

	Topics (k)					
	5	10	20	30	40	50
Spearman rho	-0.16	0.23	0.08	0.22	0.06	-0.10

TABLE III
CLASSIFICATION PERFORMANCE

k	Covariates	Accuracy	
		Web [95 % CIs]	PHP [95 % CIs]
0	38	0.788 [0.765, 0.810]	0.742 [0.717, 0.766]
5	40	0.895 [0.877, 0.911]	0.843 [0.821, 0.862]
10	40	0.910 [0.893, 0.925]	0.861 [0.841, 0.880]
20	40	0.920 [0.904, 0.935]	0.888 [0.869, 0.905]
30	40	0.912 [0.894, 0.927]	0.881 [0.862, 0.898]
40	40	0.914 [0.897, 0.929]	0.863 [0.843, 0.882]
50	40	0.913 [0.896, 0.928]	0.878 [0.858, 0.895]

exploits published in 2016 are used as a test set; these amount to about 3.5 % of all exploits observed. (It can be remarked that comparable results are obtained with randomly picked test sets containing 10 % of the exploits.) Finally, accuracy (i.e., the number of true positives and true negatives to all exploits observed) is sufficient as a performance evaluation metric in this paper. The accuracy rates reported in Table III are presented also with 95 % confidence intervals (CIs).

According to the results, performance is rather good for both response metrics, although higher accuracy rates are obtained for web exploits. Performance also increases from the inclusion of the dominant topic vectors. Given the optimum number of topics around $k \simeq 20$, the performance increases are about 0.132 and 0.146 for web and PHP exploits, respectively.

IV. DISCUSSION

This short empirical paper examined how well database meta-data and topic modeling can classify web-related and other exploits for known software vulnerabilities. By using a dataset of over 36 thousand exploits, LDA with varying number of topics, and a random forest classifier, the overall accuracy rate was observed to be in the range $[0.89, 0.92]$, which is a fairly decent range in empirical software engineering applications. The few remaining points can be presented in terms of limitations, which also provide more focused questions for further empirical research.

While topic modeling was shown to provide a neat dimensional reduction technique for classification, (a) it is unclear whether better performance might have been obtained by using the raw frequency matrices (or the inverse variants). In the same vein, (b) the classification experiment was limited to web and PHP exploits, although practical real-world applications would likely require multi-class classification that takes all categories (cf. Fig. 1) and platforms (cf. Fig. 2) into account.

Moreover, (c) different thresholds [23] could be adopted for the dominant topic assignments. Perhaps a more important concern relates to the pre-processing routines, which—like arguably in most text mining applications—are always exposed to validity concerns. In particular, (d) the used separation between English words and non-words is only a coarse approximation insofar as actual exploitation programming code is considered. In this respect, exploits posit a particularly challenging case for mining and extracting software engineering artifacts from heterogeneous sources [21], [22], [25]. Such extraction would have also practical value. Finally, (e) the near 0.9 accuracy rate can be still argued to be modest because most of the predictive power still comes from the meta-data characteristics, which require manual, human-made classification and related evaluation work. Taken together, these observations require further research on automatic classification of vulnerabilities and exploits gathered from heterogeneous Internet sources.

ACKNOWLEDGMENTS

The author acknowledges Tekes—the Finnish Funding Agency for Innovation, DIMECC Oy, and the Cyber Trust research program for their support.

REFERENCES

[1] H. Holm and T. Sommestad, "So Long, and Thanks for Only Using Readily Available Scripts," *Information & Computer Security*, vol. 25, no. 1, pp. 47–61, 2017.

[2] R. P. L. Buse and T. Zimmermann, "Information Needs for Software Development Analytics," in *Proceedings of the 34th International Conference on Software Engineering (ICSE 2012)*. Zurich: IEEE, 2012, pp. 987–996.

[3] V. H. Nguyen, S. Dashevskyi, and F. Massacci, "An Automatic Method for Assessing the Versions Affected by a Vulnerability," *Empirical Software Engineering*, vol. 21, no. 6, pp. 2268–2297, 2015.

[4] J. Anvik and G. C. Murphy, "Reducing the Effort of Bug Report Triange: Recommenders for Development-Oriented Decisions," *ACM Transactions on Software Engineering and Methodology*, vol. 20, no. 3, pp. 10:1–10:35, 2011.

[5] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy, "Characterizing and Predicting Which Bugs Get Reopened," in *Proceedings of the 34th International Conference on Software Engineering (ICSE 2012)*. Piscataway: IEEE, 2012, pp. 1074–1083.

[6] F. Cheng, S. Roschke, R. Schuppenies, and C. Meinel, "Remodeling Vulnerability Information," in *Proceedings of the International Conference on Information Security and Cryptology (Inscrypt 2009*, F. Bao, M. Yung, D. Lin, and J. Jing, Eds. Beijing: Springer, 2009, pp. 324–336.

[7] E. dos Santos Moreira, L. A. F. Martimiano, A. J. dos Santos Brandão, and M. C. Bernardes, "Ontologies for Information Security Management and Governance," *Information Management & Computer Security*, vol. 16, no. 2, pp. 105–165, 2008.

[8] G. Settanni, F. Skopik, Y. Shovgenya, R. Fiedler, M. Carolan, D. Conroy, K. Boettinger, M. Gall, G. Brost, C. Ponchel, M. Haustein, H. Kaufmann, K. Theuerkauf, and P. Olli, "A Collaborative Cyber Incident Management System for European Interconnected Critical Infrastructures," *Journal of Information Security and Applications*, no. Published online in 2 June 2016, 2016.

[9] H. Tian, L. Huang, Z. Zhou, and H. Zhang, "Common Vulnerability Markup Language," in *Proceedings of the First International Conference on Applied Cryptography and Network Security (ACNS 2003), Lecture Notes in Computer Science (Volume 2846)*, J. Zhou, M. Yung, and Y. Han, Eds. Kunming: Springer, 2003, pp. 228–240.

[10] S. Christey and B. Martin, "Buying Into the Bias: Why Vulnerability Statistics Suck," in *Presentation at Black Hat 2013*, Las Vegas, 2013, available online in January 2017: https://media.blackhat.com/us-13/US-13-Martin-Buying-Into-The-Bias-Why-Vulnerability-Statistics-\\Suck-Slides.pdf.

[11] Open Source Vulnerability Database, "OSVDB: FIN," 2016, available online in February 2017: https://blog.osvdb.org/2016/04/05/osvdb-fin/.

[12] Z. Chen, Y. Zhang, and Z. Chen, "A Categorization Framework for Common Computer Vulnerabilities and Exposures," *The Computer Journal*, vol. 53, no. 5, pp. 551–580, 2010.

[13] S. Neuhaus and T. Zimmermann, "Security Trend Analysis with CVE Topic Models," in *Proceedings of the IEEE 21st International Symposium on Software Reliability Engineering (ISSRE 2010)*. San Jose: IEEE, 2010, pp. 111–120.

[14] N. Scarabeo, B. C. Fung, and R. H. Khokhar, "Mining Known Attack Patterns from Security-Related Events," *PeerJ Computer Science*, vol. e25, no. 1, pp. 1–21, 2014.

[15] D. Toloudis, G. Spanos, and L. Angelis, "Associating the Severity of Vulnerabilities with Their Description," in *Proceedings of the CAiSE 2016 International Workshops, Lecture Notes in Business Information Processing (Volume 249)*, J. Krogstie, H. Mouratidis, and J. S. D. Toloudis, Eds. Ljubljana: Springer, 2016, pp. 231–242.

[16] Exploit Database, "Offensive Security Exploit Database," 2017, available online in February 2017: https://www.exploit-db.com/.

[17] M. Fang and M. Hafiz, "Discovering Buffer Overflow Vulnerabilities in the Wild: An Empirical Study," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2014)*. Torino: ACM, 2014, pp. 1–10.

[18] D. Marciniak, "Computational Text Analysis: Thoughts on the Contingencies of an Evolving Method," *Big Data & Society*, pp. 1–5, 2016.

[19] H. Homaei and H. R. Shahriari, "Seven Years of Software Vulnerabilities: The Ebb and Flow," *Security & Privacy*, vol. 15, no. 1, pp. 58–65, 2017.

[20] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu, "On the Naturalness of Software," in *Proceedings of the 34th International Conference on Software Engineering (ICSE 2012)*. Zurich: IEEE, 2012, pp. 837–847.

[21] L. Ponzanelli, A. Mocci, and M. Lanza, "Summarizing Complex Development Artifacts by Mining Heterogeneous Data," in *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR 2015)*. Piscataway: IEEE, 2015, pp. 401–405.

[22] L. Wu, L. Du, B. Liu, G. Xu, Y. Ge, Y. Fu, J. Li, Y. Zhou, and H. Xiong, "Heterogeneous Metric Learning with Content-Based Regularization for Software Artifact Retrieval," in *Proceedings of the IEEE International Conference on Data Mining (ICDM 2014)*. Shenzhen: IEEE, 2014, pp. 610–619.

[23] T.-H. Chen, S. W. Thomas, and A. E. Hassan, "A Survey on the Use of Topic Models When Mining Software Repositories," *Empirical Software Engineering*, vol. 21, no. 5, pp. 1843–1919, 2016.

[24] S. Khatiwada, M. Kelly, and A. Mahmoud, "STAC: A Tool for Static Textual Analysis of Code," in *Proceedings of the IEEE 24th International Conference on Program Comprehension (ICPC 2016)*. Austin: IEEE, 2016, pp. 1–3.

[25] T. Pawelka and E. Juergens, "Is This Code Written in English? A Study of the Natural Language of Comments and Identifiers in Practice," in *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME 2015)*. Bremen: IEEE, 2015, pp. 401–410.

[26] F. Maggi, A. Bellini, G. Salvaneschi, and S. Zanero, "Finding Non-Trivial Malware Naming Inconsistencies," in *Proceedings of the 7th International Conference (ICISS 2011), Lecture Notes in Computer Science (Volume 7093)*, S. Jajodia and C. Mazumdar, Eds. Kolkata: Springer, 2011, pp. 144–159.

[27] R. Perdisci and M. U, "VAMO: Towards a Fully Automated Malware Clustering Validity Analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC 2012)*. Orlando: ACM, 2012, pp. 329–338.

[28] The Natural Language Toolkit (NLTK), "NLTK 3.0 Documentation," 2017, available online in January 2017: http://www.nltk.org.

[29] R. Kelly, "PyEnchant: A Spellchecking Library for Python (1.6.8)," 2017, available online in January 2017: http://pythonhosted.org/pyenchant/.

[30] D. Lachowicz, "Enchant (1.6.0)," 2017, available online in January 2017: http://www.abisource.com/projects/enchant/.

[31] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei, "Hierarchical Dirichlet Processes," *Journal of the American Statistical Association*, vol. 101, no. 476, pp. 1566–1581, 2006.

[32] D. M. Blei, "Probabilistic Topic Models," *Communications of the ACM*, vol. 55, no. 4, pp. 77–84, 2012.

[33] S. Debortoli, O. Müller, I. Junglas, and J. vom Brocke, "Text Mining for Information Systems Researchers: An Annotated Topic Modeling Tutorial," *Communications of the Association for Information Systems*, vol. 39, p. Article 7, 2016.

[34] B. Grün and K. Hornik, "topicmodel: An R Package for Fitting Topic Models," *Journal of Statistical Software*, vol. 40, no. 13, pp. 1–30, 2016.

[35] A. Barua, S. W. Thomas, and A. E. Hassan, "What Are Developers Talking About? An Analysis of Topics and Trends in Stack Overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.

[36] A. Liaw and M. Wiener, "Classification and Regression by randomForest," *R News*, vol. 2, no. 3, pp. 18–22, 2002.

[37] M. Kuhn, "Building Predictive Models in R Using the `caret` Package," *Journal of Statistical Software*, vol. 28, no. 5, pp. 1–26, 2008.

[38] Google, Inc., "Avoid Videos... But If You Can't, Here Are Some Tips :-)," 2016, Google Bughunter University, available online in November 2016: https://sites.google.com/site/bughunteruniversity/improve/how-to-record-an-effective-proof-of-concept-video.

[39] F. Massacci and V. H. Nguyen, "Which Is the Right Source for Vulnerability Studies? An Empirical Analysis on Mozilla Firefox," in *Proceedings of the 6th International Workshop on Security Measurements and Metrics (MetriSec 2010)*. Bolzano: ACM, 2010, pp. 4:1–4:8.

[40] P. Johnson, D. Gorton, R. Langerström, and M. Ekstedt, "Time Between Vulnerability Disclosures: A Measure of Software Product Vulnerability," *Computers & Security*, vol. 62, pp. 278–295, 2016.

[41] J. Ruohonen, S. Hyrynsalmi, and V. Leppänen, "Trading Exploits Online: A Preliminary Case Study," in *Proceedings of the IEEE Tenth International Conference on Research Challenges in Information Science (RCIS 2016)*. Grenoble: IEEE, 2016, pp. 1–12.