Identifying Security Issues in Software Development: Are Keywords Enough?

Patrick Morrison
Department of Computer Science
North Carolina State University
Raleigh, North Carolina
pjmorris@ncsu.edu

Tosin Daniel Oyetoyan
Department of Software Engineering, Safety & Security, SINTEF Digital
Trondheim, Norway
tosin.oyetoyan@sintef.no

Laurie Williams
Department of Computer Science
North Carolina State University
Raleigh, North Carolina
williams@csc.ncsu.edu

KEYWORDS

Security, vocabulary, classification model, CVE, Prediction

1 EXTENDED ABSTRACT

Identifying security issues before attackers do has become a critical concern for software development teams and software users. While methods for finding programming errors (e.g. fuzzers [1], static code analysis [3] and vulnerability prediction models like Scandariato et al. [10]) are valuable, identifying security issues related to the lack of secure design principles and to poor development processes could help ensure that programming errors are avoided before they are committed to source code.

Typical approaches (e.g. [4, 6–8]) to identifying security-related messages in software development project repositories use text mining based on pre-selected sets of standard security-related keywords, for instance; *authentication*, *ssl*, *encryption*, *availability*, or *password*. We hypothesize that these standard keywords may not capture the entire spectrum of security-related issues in a project, and that additional project-specific and/or domain-specific vocabulary may be needed to develop an accurate picture of a project's security.

For instance, Arnold et al.[1], in a review of bug-fix patches on Linux kernel version 2.6.24, identified a commit (commit message: *"Fix − >vm_file accounting, mmap_region() may do do_munmap()"* [2]) with serious security consequences that was mis-classified as a non-security bug. While no typical security keyword is mentioned, memory mapping ('mmap') in the domain of kernel development has significance from a security perspective, parallel to buffer overflows in languages like C/C++. Whether memory or currency is at stake, identifying changes to assets that the software manages is potentially security-related.

The goal of this research is to support researchers and practitioners in identifying security issues in software development project artifacts by defining and evaluating a systematic scheme for identifying project-specific security vocabularies that can be used for keyword-based classification.

We derive three research questions from our goal:

- **RQ1**: How does the vocabulary of security issues vary between software development projects?
- **RQ2**: How well do project-specific security vocabularies identify messages related to publicly reported vulnerabilities?
- **RQ3**: How well do existing security keywords identify project-specific security-related messages and messages related to publicly reported vulnerabilities?

To address these research questions, we collected developer email, bug tracking, commit message, and CVE record project artifacts from three open source projects : Dolibarr, Apache Camel, and Apache Derby. We manually classified 5400 messages from the three project's commit messages, bug trackers, and emails, and linked the messages to each project's public vulnerability records, Adapting techniques from Bachmann and Bernstein [2], Schermann et al. [11], and Guzzi [5], we analyzed each project's security vocabulary and the vocabulary's relationship to the project's vulnerabilities. We trained two classifiers ($Model.A$ and $Model.B$) on samples of the project data, and used the classifiers to predict security-related messages in the manually-classified project oracles.

Our contributions include:

- A systematic scheme for linking CVE records to related messages in software development project artifacts
- An empirical evaluation of project-specific security vocabulary similarities and differences between project artifacts and between projects

To summarize our findings on **RQ1**, we present tables of our qualitative and quantitative results. We tabulated counts of words found in security-related messages. Traditional security keywords (e.g. password, encryption) are present, particularly in the explicit column, but each project also contains terms describing entities unique to the project, for example 'endpoint' (Camel), 'blob' (short for 'Binary Large Object'), 'clob' ('Character Large Object'), 'deadlock' (Derby), and 'invoice', 'order' for Dolibarr. The presence of these terms in security-related issues suggests that they are assets worthy of careful attention during the development life cycle.

Table 1 lists the statistics for security-related messages from the three projects, broken down by security class and security property. Explicit security-related messages (messages referencing security properties) are in the minority in each project. Implicit messages represent the majority of security-related messages in each project.

In Table 2, we present the results of the classifiers built using the various project and literature security vocabularies to predict security-related messages in the oracle and CVE datasets. We have

[1] http://lcamtuf.coredump.cx/afl/
[2] https://git.amelchem.com/amel/linux/commit/8a459e44ad837018ea5c34a9efe8eb4ad27ded26

ICSE'18 Companion, Gothenburg, Sweden
 978-1-4503-5663-3/18/05…$15.00
DOI: 10.1145/3183440.3195040

Table 1: Counts of security-related messages per project, ratios of security properties to counts of security-related messages

		Statistics (%)											
Systems	Security-related Count	Security-related %	Explicit	Implicit	CWE	Integrity	Confidentiality	Availability	Authentication	Authorization	Auditing	API	Quality
Dolibarr	267	22.2	26.2	73.8	35.6	49.8	5.2	6.4	7.1	9.4	1.9	5.9	23.6
Apache Derby	778	64.8	24.7	75.3	29.3	68.4	4.8	16.1	3.1	10.0	1.2	4.2	10.2
Apache Camel	936	78	5.4	94.6	19.6	43.2	3.6	11.7	1.3	1.3	2.4	29.6	25.4

Table 2: Project Oracle Dataset Classifer Performance

	Model.A			Model.B		
	Recall	Precision	F-score	Recall	Precision	F-score
Derby.model → Derby.dataset	**0.917**	**0.960**	**0.938**	0.833	0.719	0.772
Camel.model → Derby.dataset	0.983	0.699	0.781	0.797	0.697	0.744
Dolibarr.model → Derby.dataset	0.238	0.894	0.375	**0.880**	0.707	**0.784**
Ray.vocab → Derby.dataset	0.148	0.938	0.257	0.171	0.795	0.282
Pletea.vocab → Derby.dataset	0.089	0.986	0.164	0.093	**0.894**	0.168
Derby.model → Camel.dataset	0.346	0.949	0.505	0.584	**0.894**	0.707
Camel.model → Camel.dataset	**0.961**	**0.961**	**0.961**	**0.905**	0.808	**0.854**
Dolibarr.model → Camel.dataset	0.085	0.879	0.154	0.704	0.846	0.768
Ray.vocab → Camel.dataset	0.081	0.909	0.148	0.082	0.762	0.148
Pletea.vocab → Camel.dataset	0.084	0.963	0.154	0.102	0.593	0.173
Derby.model → Dolibarr.dataset	0.355	0.408	0.376	0.503	0.308	0.382
Camel.model → Dolibarr.dataset	0.742	0.291	0.419	0.690	0.302	0.421
Dolibarr.model → Dolibarr.dataset	**0.807**	**0.947**	**0.871**	**0.935**	0.291	**0.444**
Ray.vocab → Dolibarr.dataset	0.157	0.907	0.267	0.158	**0.876**	0.267
Pletea.vocab → Dolibarr.dataset	0.287	0.840	0.428	0.251	0.729	0.373

Table 3: CVE Dataset Classifer Performance

	Model.A			Model.B		
	Recall	Precision	F-score	Recall	Precision	F-score
Derby.model → Derby.CVE.dataset	**0.786**	0.337	0.472	**0.897**	0.305	0.455
Ray.vocab → Derby.CVE.dataset	0.386	0.590	0.467	0.391	0.533	0.451
Pletea.vocab → Derby.CVE.dataset	0.361	**0.719**	**0.481**	0.433	**0.736**	**0.545**
Camel.model → Camel.CVE.dataset	**0.682**	0.031	0.058	**0.863**	0.034	0.065
Ray.vocab → Camel.CVE.dataset	0.226	0.110	0.147	0.226	**0.102**	**0.140**
Pletea.vocab → Camel.CVE.dataset	0.287	**0.116**	**0.171**	0.264	0.081	0.123
Dolibarr.model → Dolibarr.CVE.dataset	**0.272**	0.079	0.123	**0.625**	0.027	0.052
Ray.vocab → Dolibarr.CVE.dataset	0.108	**0.152**	**0.127**	0.108	**0.128**	**0.117**
Pletea.vocab → Dolibarr.CVE.dataset	0.096	0.023	0.037	0.118	0.027	0.044

marked in **bold** the highest result for each performance measure for each dataset. Both Models A and B have a high performance across the projects when predicting for the oracle dataset of the project for which they were built. Further, the project-specific models have higher performance than the literature-based models (Ray.vocab [9] and Pletea.vocab [7]) on the project oracle datasets. Model performance is not sustained and is inconsistent when applied to other project's datasets.

To summarize our findings on **RQ2**, Table 3 presents performance results for the project vocabulary models on the CVE datasets for each project. We have marked in **bold** the highest result for each performance measure for each dataset. Results for Model.A shows a high recall for Derby and Camel and a worse than average recall for Dolibarr. However, in Model.B, the recall is above 60% for Dolibarr and over 85% for both Derby and Camel. We reason the low precision is due to our approach of labeling only CVE-related messages as security-related and the rest of the messages are labeled to be not security-related. The Dolibarr results are further complicated by the low proportion of security-related messages compared with the other two projects (as reported in 1).

To summarize our findings on **RQ3**, Table 2 and Table 3 present the classifier performance results for two sets of keywords, Ray.vocab, and Pletea.vocab, drawn from the literature. In each case, the project vocabulary model had the highest recall, precision and F-Score on the project's oracle dataset. With regards to the CVE-dataset, the project vocabulary model has the highest recall. However, the overall performance, as measured by F-Score, varied by dataset, with the Ray and Pletea keywords scoring higher than the project vocabulary model. The low precision for the classifier built on the project's vocabularies follows the explanation provided under RQ2.

Our results suggest that domain vocabulary model show recalls that outperform standard security terms across our datasets. Our conjecture, supported in our data, is that augmenting standard security keywords with a project's security vocabulary yields a more accurate security picture. In future work, we aim to refine vocabulary selection to improve classifier performance, and to define tools implementing the approach in this paper to aid practitioners and researchers in identifying software project security issues.

ACKNOWLEDGEMENTS

The work in this paper was partly supported by the Research Council of Norway through the project SoS-Agile: Science of Security in Agile Software Development (247678/O70).

REFERENCES

[1] Jeff Arnold, Tim Abbott, Waseem Daher, Gregory Price, Nelson Elhage, Geoffrey Thomas, and Anders Kaseorg. 2009. Security Impact Ratings Considered Harmful.. In *HotOS*.

[2] A. Bachmann and A. Bernstein. 2009. Data Retrieval, Processing and Linking for Software Process Data Analysis. (2009).

[3] Brian Chess and Gary McGraw. 2004. Static analysis for security. *IEEE Security & Privacy* 2, 6 (2004), 76–79.

[4] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. 2006. The Detection and Classification of Non-Functional Requirements with Application to Early Aspects. In *Requirements Engineering, 14th IEEE International Conference* (2006-09). Minneapolis, Minnesota, 39–48. https://doi.org/10.1109/RE.2006.65

[5] Anja Guzzi, Alberto Bacchelli, Michele Lanza, Martin Pinzger, and Arie van Deursen. 2013. Communication in Open Source Software Development Mailing Lists. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*. IEEE Press, Piscataway, NJ, USA, 277–286. http://dl.acm.org/citation.cfm?id=2487085.2487139

[6] Abram Hindle, Neil A. Ernst, Michael W. Godfrey, and John Mylopoulos. 2013. Automated Topic Naming. *Empirical Softw. Engg.* 18, 6 (Dec. 2013), 1125–1155. https://doi.org/10.1007/s10664-012-9209-9

[7] Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. 2014. Security and emotion: sentiment analysis of security discussions on GitHub. In *Proceedings of the 11th working conference on mining software repositories*. ACM, 348–351.

[8] Baishakhi Ray, Vincent Hellendoorn, Saheel Godhane, Zhaopeng Tu, Alberto Bacchelli, and Premkumar Devanbu. 2016. On the "Naturalness" of Buggy Code. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*. ACM, New York, NY, USA, 428–439. https://doi.org/10.1145/2884781.2884848

[9] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. 2014. A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 155–165.

[10] Riccardo Scandariato, James Walden, Aram Hovsepyan, and Wouter Joosen. 2014. Predicting vulnerable software components via text mining. *IEEE Transactions on Software Engineering* 40, 10 (2014), 993–1006.

[11] Gerald Schermann, Martin Brandtner, Sebastiano Panichella, Philipp Leitner, and Harald Gall. 2015. Discovering Loners and Phantoms in Commit and Issue Data. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension (ICPC '15)*. IEEE Press, Piscataway, NJ, USA, 4–14. http://dl.acm.org/citation.cfm?id=2820282.2820287