A Manually-Curated Dataset of Fixes to Vulnerabilities of Open-Source Software

Serena E. Ponta, Henrik Plate, Antonino Sabetta, Michele Bezzi, Cédric Dangremont
SAP Security Research
Mougins, France
{<firstname>.<lastname>}@sap.com

***Abstract*—**

Advancing our understanding of software vulnerabilities, automating their identification, the analysis of their impact, and ultimately their mitigation is necessary to enable the development of software that is more secure.

While operating a vulnerability assessment tool, which we developed, and that is currently used by hundreds of development units at SAP, we manually collected and curated a dataset of vulnerabilities of open-source software, and the commits fixing them. The data were obtained both from the National Vulnerability Database (NVD), and from project-specific web resources, which we monitor on a continuous basis.

From that data, we extracted a dataset that maps 624 publicly disclosed vulnerabilities affecting 205 distinct open-source Java projects, used in SAP products or internal tools, onto the 1282 commits that fix them. Out of 624 vulnerabilities, 29 do not have a CVE (Common Vulnerability and Exposure) identifier at all, and 46, which do have such identifier assigned by a numbering authority, are not available in the NVD yet.

The dataset is released under an open-source license, together with supporting scripts that allow researchers to automatically retrieve the actual content of the commits from the corresponding repositories, and to augment the attributes available for each instance. Moreover, these scripts allow to complement the dataset with additional instances that are not security fixes (which is useful, for example, in machine learning applications).

Our dataset has been successfully used to train classifiers that could automatically identify security-relevant commits in code repositories. The release of this dataset and the supporting code as open-source will allow future research to be based on data of industrial relevance; it also represents a concrete step towards making the maintenance of this dataset a shared effort involving open-source communities, academia, and the industry.

I. Introduction

The availability of mature, high-quality open-source software (OSS) components represents an opportunity for the software industry to accelerate innovation and lower costs, but, at the same time, maintaining a secure OSS supply chain and an effective vulnerability management process is a significant challenge.

In recent years, several approaches to the management of OSS vulnerabilities have emerged, and the market of commercial tools has matured significantly. However, as shown in [7], an effective vulnerability management approach cannot rely purely on metadata only, since they are often inaccurate, incomplete, or missing. Rather, it has to be *code-centric*, that is, it must be based on the actual analysis of vulnerabilities, and their fixes, at code level.

While operating a vulnerability assessment tool that we developed to implement such code-centric approach, and that is currently used by hundreds of development units at SAP, we manually collected a substantial amount of code-level data about vulnerabilities of open-source software and their fixes. Starting from these data, and with the objective to study automated approaches to simplify the expensive and difficult problem of manually identifying the fixes for new vulnerability disclosures, we created a dataset, which we present here, that was successfully used to train automated commit classifiers [8].

Our dataset maps 624 publicly disclosed vulnerabilities affecting 205 distinct open-source Java projects used in SAP software (either products or internal tools) onto the 1282 commits that fix them. It was constructed and manually curated over a period of four years, monitoring the disclosure of security advisories, not only from the National Vulnerability Database (NVD), but also from numerous project-specific Web pages.

Compared to existing works, our dataset has several distinguishing characteristics.

- Gkortzis et al. [2] analyzed the advisories of the NVD, and mapped each vulnerability onto the open-source code repository of the affected products.
  Our dataset instead is entirely manually curated and it has a finer granularity, as it maps *vulnerabilities* onto the *commits that fix them*. Moreover, in addition to the NVD, we considered additional project-specific sources of advisories. By manually curating our dataset we could ensure the quality of each entry, detecting and addressing some of the inconsistencies that are known to affect the NVD [6].
- Our dataset includes open-source projects that are widely adopted and that have practical industrial relevance. We ensured that our dataset includes the population of OSS that are actually used in real enterprise software products or internal tools. We are not aware of any other freely available dataset that has these characteristics.
- Differently from [5], our dataset contains commits that implement *fixes* to vulnerabilities, whereas theirs contains commits that *introduce* vulnerabilities.

2574-3864/19/$31.00 ©2019 IEEE
DOI 10.1109/MSR.2019.00064

- Snyk[1] does advertise its vulnerability databases through its website. Such database is focused on vulnerability descriptions and a reference to the fix is not always included. Also, the access to the Snyk dataset is subject to restrictions and their data are not available for free download[2]. Our dataset instead is available under the Apache license and its release is part of an initiative to promote a collaborative effort involving industry, academia, and the open-source community, to maintain and extend the dataset in the long run.

In the next section, we present the process we followed to construct the dataset, whose content is described in Section III. Possible applications of the dataset are illustrated in Section IV. Section V presents some concluding remarks.

II. Dataset Construction

The motivation for collecting code-level vulnerability data originates in the work that SAP Security Research has performed since 2014 on the analysis of vulnerabilities in open-source software. A key result of that work is an approach to the detection, assessment and mitigation of open-source vulnerabilities [7]. The approach is implemented as a tool (internally called Vulas), which has been productively used at SAP to analyze hundreds of Java and Python software applications and that is now publicly available as free open-source software released under the Apache version 2.0 license[3].

The availability of a vulnerability knowledge base, that is comprehensive, accurate, and updated in a timely manner, is the prerequisite for such a tool to operate effectively. For this reason, we continuously monitor both the NVD and over 50 distinct project-specific websites for new vulnerability disclosures (e.g., the Pivotal vulnerability reports[4]). For each such disclosure, we manually review the available information and we search for the corresponding fix-commit(s) in the code repository of the affected open-source component. More in details, given a vulnerability disclosure:

1) First, we choose an identifier for the vulnerability. By default, we use the CVE identifier. If this is not available, we use a project-specific identifier (such as a bug-tracking identifier). When neither is available, we define an identifier using the project name and incremental numbers.
2) Second, we identify the source code repository of the project affected by the vulnerability. In most cases the advisory does not provide a link to the repository, and thus it needs to be determined manually. We estimate that over 70% of the NVD vulnerabilities in our dataset do not have any link to a source code repository.
3) Finally, we identify the commit(s) fixing the vulnerability. In most cases this requires expert knowledge to browse the repository and to determine the commit(s) that fix the vulnerability. In this step, we rely on the text of the advisory (and linked resources), on the commit messages, and on the code changes introduced by the commit. The difficulty in the identification of fix commits varies considerably across projects, depending on the amount and quality of the information that they disclose in commit messages, change logs, and other resources.

The result of this ongoing activity, started in 2014, is a database of vulnerability data that has a very high coverage of the projects that are relevant to our company.

In this paper we describe a snapshot of this database[5] taken on 21 January 2019 that contains 1282 commits, from 205 distinct open-source Java projects used in SAP software (either products or internal tools). These commits correspond to the fixes of 624 publicly known vulnerabilities.

III. Dataset Description

The dataset consists of a set of 4-tuples

$$(vulnerability\_id, repository\_url, commit\_id, class)$$

where $vulnerability\_id$ is the identifier of a vulnerability that is fixed in the commit with identifier $commit\_id$ performed in the source code repository at $repository\_url$. Each entry of the dataset represents a commit that contributes to fixing a vulnerability (so-called *fix commits*) and thus we assign them to the positive $class$ (*pos*). In certain applications, commits of the negative class (denoted by *neg*) are also needed (see Section IV). Together with the set of positive instances, we also provide supporting scripts to manipulate it and to obtain negative instances automatically. Both the data and the scripts are released under the Apache 2.0 license.

The dataset covers 205 distinct projects, and includes 1282 unique commits corresponding to the fixes to 624 vulnerabilities. Differently from other existing datasets (e.g., [2]), our dataset includes vulnerabilities that are not available at the National Vulnerability Database, and that we obtained from project-specific advisories. In particular the dataset includes 29 vulnerabilities without CVE identifier[6], and 46 vulnerabilities which have been given a CVE identifier by a CVE numbering authority, but are not yet published on NVD.

[1] https://snyk.io/vuln/

[2] Snyk stopped offering dumps of their vulnerability database on GitHub as of February 2018.

[3] https://github.com/sap/vulnerability-assessment-tool

[4] https://pivotal.io/security

[5] The dataset is available for free download at the address: https://github.com/SAP/vulnerability-assessment-kb/tree/master/MSR2019. Some of the entries were discarded because the commits were invalid (e.g., some repositories had been dismissed or moved since we first introduced them in the dataset).

[6] CVE identifiers are the most-used naming convention used to enumerate vulnerabilities.

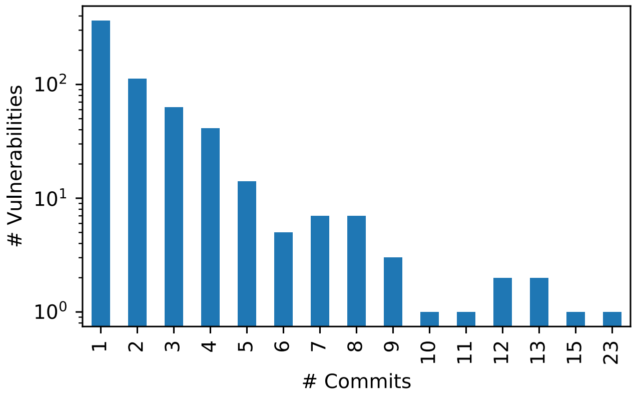

Figure 1: Number of vulnerabilities per number of commits performed for fixing them (note: the *y-axis* uses log-scale).

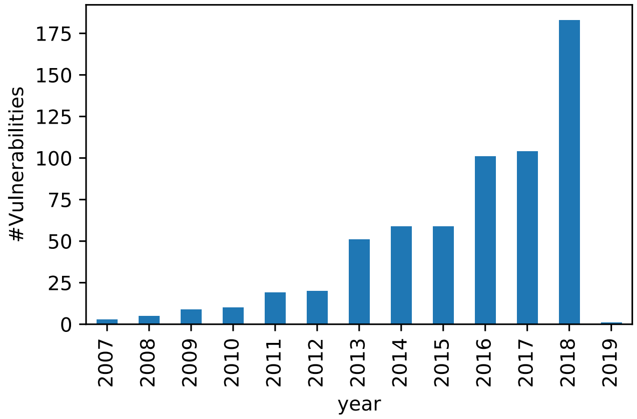

Figure 2: Number of vulnerabilities per year.

The dataset can be considered to cover a representative sample of projects of practical industrial relevance: these projects were identified based on an analysis of the data collected at SAP while operating our open-source vulnerability assessment tool (internally known as Vulas) for a period of about four years, during which the tool was used for hundreds of thousands of security scans. Most of the open-source projects included in the dataset are hosted on GitHub.com (or a mirror of their official repository is available there).

The dataset is released as comma-separated values (CSV) file, which makes it readily usable as input to scripts that can fetch automatically additional data from each repository. We do provide example code (in the form of a Jupyter notebook and a few supporting Python scripts) that simplify the manipulation, the extension, and the analysis of the dataset.

As shown in Figure 1, most of the vulnerabilities (364) are fixed in a single commit whereas in 7 cases the number of commits done to fix the vulnerability goes over ten, up to twenty-three for CVE-2015-5348.

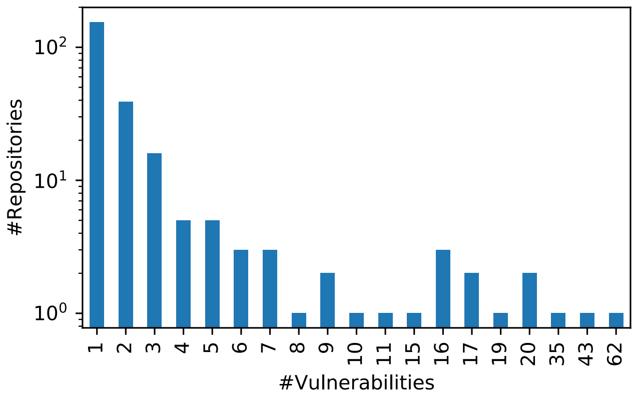

Figure 3: Number of repositories per number of vulnerabilities (note: the *y-axis* uses log-scale).

The existence of multiple commits per vulnerability may indicate either that the fix is performed in steps (in which case, the entire set of commits determines the fix), or that the fix is back-ported to other versions (branches). In the latter case, the fix could be replicated as-is, or it may be adpated to make it compatible with older code. The top-20 vulnerabilities by number of fix-commits is shown in Table I-a.

Figure 2 shows the distribution of the vulnerabilities in the dataset over the years. The bias towards recent years is due to several reasons: (1) the number of vulnerabilities published increases every year; (2) often the information available for old vulnerabilities is very limited (e.g., dead reference links, moved repositories no longer available); (3) as the dataset is manually-curated, the effort spent to identify fix-commits needs to be allocated carefully, and we give higher priority to recent vulnerabilities.

Figure 3 shows the distribution of vulnerabilities across different repositories. In particular it highlights that the majority of repositories (178 over 205) have only one vulnerability. The top-20 projects by number of vulnerabilities are listed in Table I-b

IV. Applications

The dataset presented in this paper can be used to study vulnerabilities and their fixes in open-source software of industrial relevance (e.g., [4], [8]).

Using the scripts we distribute with the dataset, researchers can easily clone all the repositories referred to in the dataset and follow our code samples to extract any commit feature available through Git.

As an example, for each fix-commit, we automatically determine the oldest tag from which that commit is reachable and compute the time elapsed between the two: this can be used to study the time elapsing from the time when a vulnerability fix is committed in a repository until a new

Vulnerability Id.	Commits
CVE-2015-5348	23
CVE-2012-0022	15
CVE-2018-8009	13
CVE-2016-6801	13
CVE-2016-8749	12
CVE-2018-8027	12
CVE-2014-0119	11
CVE-2012-2098	10
CVE-2013-1768	9
CVE-2017-15719	9
CVE-2016-5641	9
CVE-2016-3674	8
CVE-2016-5018	8
CVE-2014-3558	8
CVE-2010-0432	8
CVE-2016-6796	8
CVE-2017-12617	8
CVE-2017-5664	8
CVE-2017-15695	7
CVE-2015-1833	7
CVE-2018-1335	7

(a)

Repository	Vulnerabilities
https://github.com/apache/tomcat	69
https://github.com/apache/struts	44
https://github.com/apache/tomcat70	35
https://github.com/jenkinsci/jenkins	34
https://github.com/spring-projects/spring-framework	24
https://github.com/cloudfoundry/uaa	22
https://github.com/apache/tomcat85	20
https://github.com/apache/tomcat80	20
https://github.com/apache/activemq	18
https://github.com/apache/cxf	17
https://github.com/apache/tomcat55	16
https://github.com/bcgit/bc-java	15
https://github.com/apache/camel	13
https://github.com/eclipse/jetty.project	13
https://github.com/apache/lucene-solr	11
https://github.com/apache/hadoop	9
https://github.com/spring-projects/spring-security	8
https://github.com/FasterXML/jackson-databind	8
https://github.com/apache/tika	7
https://github.com/apache/ofbiz	6
https://github.com/apache/nifi	6

(b)

Days	# Commits
1	181
2	166
3	256
4	259
5	308
6	282
7	257
8	136
9	144
10	172
11	170
12	129
13	128
14	512
15	520
15 to 50	592
50 to 100	257
100 to 500	290
500 to 1000	72
1000 to 2000	19
No tag	*167*

(c)

Table I: Dataset description: (a) Number of fix-commits per vulnerability (top-20); (b) Number of vulnerabilities per repository (top-20); (c) Number of commits per days elapsed between the the fix-commit and the following tag (release).

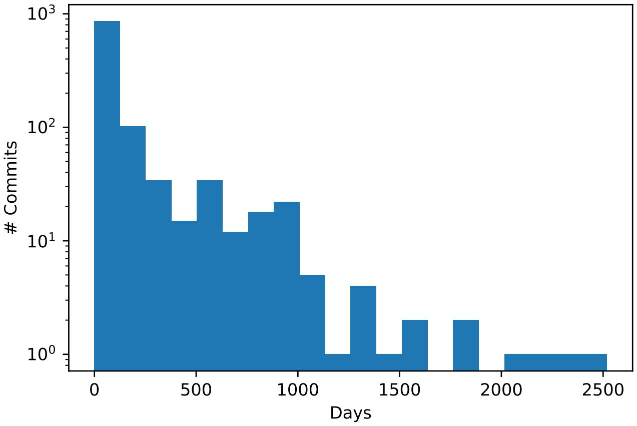

Figure 4: Number of days from fix commit to release (note: the *y-axis* uses log-scale).

non-vulnerable release (containing such fix) is created (and tagged).

Because attackers could easily monitor issues and commits in the repositories of security-relevant open source projects, it is critical to keep this delay as short as possible. During that time-frame, the fix (and thus the vulnerability that it fixes) is public, but the clients relying on the affected open-source project cannot update to a non-vulnerable release, because it does not exist yet.

Figure 4 shows the distribution of such delays for the commits in our dataset. The figure shows that most fix commits (817) are released in less than 100 days (of which 181 are released the same day), but delays can be much higher in a considerable number of cases. Detailed information about how many days occur from fix-commits to their releases is available in Table I-c.

It is interesting to observe that as many as 167 commits are not reachable from any tag (see last line in Table I-c). Since most open-source projects follow the established practice of creating a tag for each release, this figure might suggest that a non-negligible number of fix-commits is available in the code repository of the project but is not yet part of any release.

Another example of a possible application of the dataset is presented in [8]. Motivated by the need to automate the maintenance of the very vulnerability database from which our dataset is extracted, Sabetta and Bezzi [8] presented a novel approach to the automated classification of commits that are *security-relevant* (i.e., that are likely to fix a vulnerability). They used (an older, and smaller version of) the dataset presented here to train two independent classifiers, considering, respectively, the patch introduced by a commit (*Patch Classifier*) and the *log messages* (*Message Classifier*), without relying on information from vulnerability advisories.

Inspired by the *naturalness* hypothesis [1], [3], the *Patch Classifier* treats source code changes as documents written in natural language (*code-as-text*), and it classifies them using established Natural Language Processing (NLP) methods. A similar approach is also used for the *Message Classifier*. The results of the two classifiers, which are tuned for high precision, are then combined with a simple voting mechanism that flags a commit as security-relevant as soon as at least one of the two models does.

As we pointed out in Section III, it is common practice to commit the same change in multiple branches of the same repository, so that the fix is made available to users of different supported versions of the component at hand. As a consequence of this practice, our dataset contains duplicates (commits with different identifier but identical content) which need to be removed before training a classifier. The de-duplication results in 862 unique *positive* instances.

Generally speaking, datasets used to train automated classifiers must include commits corresponding both to security fixes (*positive* class) and to other changes not related to security (*negative* class). To support these applications, we provide a script that, starting from the set of positive instances at our disposal, augments the dataset with negative instances (non-security commits). The algorithm works as follows: for each positive instance p from repository R, we take k random commits $n_1, \ldots, n_k$ from R and, under the assumption that security-relevant commits are rare compared to other types of commits, we treated these as "negative" examples. To avoid including obvious outliers (extremely large, empty, or otherwise invalid commits), we performed a manual review of these commits, supported by *ad-hoc* scripts and pattern matching (similarly to [9]). These patterns are used to speed-up the manual review by searching for patterns in the commit messages that would indicate with high probability that the commit is security-related.

The model of [8] seems to represent an improvement over a similar state-of-the-art approach [9] which relies on log messages processed through a more complex architecture trained with a different (and substantially larger) dataset. Unfortunately, the dataset used in [9] is not public, which makes it impossible to conduct a reliable comparison of the two approaches.

We hope that, by sharing our dataset, we will encourage further works to demonstrate the extent of their claimed improvements on the basis of a common benchmark that is *freely available*, *machine-readable*, and that covers open-source projects of *actual industrial relevance*.

V. Concluding remarks

We have presented a dataset of fixes to vulnerabilities in Java OSS projects of industrial relevance, resulting from our experience with developing and operating an open-source vulnerability management solution at SAP.

While the data we are releasing at this time cover only Java projects, in the meantime the tool has evolved to support more languages and consequently we are working to extend our dataset further.

To remain useful for the validation of enterprise applications in the long term, the dataset needs to be updated on a continuous basis to incorporate new vulnerabilities as soon as they are disclosed. While the current dataset has been constructed at SAP, we are working to make its future maintenance a community effort and to encourage the very developers who implement the security fixes for vulnerable open-source components to contribute to this dataset. To facilitate this community effort, we plan to release support tools and processes, thus complementing the release of our *vulnerability assessment tool*, which was open-sourced in October 2018. We are convinced that the tool and its vulnerability database represent a practical contribution to ensuring the security of the open-source software supply-chain.

References

[1] Miltiadis Allamanis, Earl T. Barr, Premkumar Devanbu, and Charles Sutton. A survey of machine learning for big code and naturalness. *arXiv preprint arXiv:1709.06182*, 2017.

[2] Antonios Gkortzis, Dimitris Mitropoulos, and Diomidis Spinellis. Vulinoss: A dataset of security vulnerabilities in open-source systems. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, pages 18–21, New York, NY, USA, 2018. ACM.

[3] Abram Hindle, Earl T. Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. On the naturalness of software. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 837–847, Piscataway, NJ, USA, 2012. IEEE Press.

[4] Ivan Pashchenko, Henrik Plate, Serena Elisa Ponta, Antonino Sabetta, and Fabio Massacci. Vulnerable open source dependencies: counting those that matter. In Markku Oivo, Daniel Méndez Fernández, and Audris Mockus, editors, *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2018, Oulu, Finland, October 11-12, 2018*, pages 42:1–42:10. ACM, 2018.

[5] Henning Perl, Sergej Dechand, Matthew Smith, Daniel Arp, Fabian Yamaguchi, Konrad Rieck, Sascha Fahl, and Yasemin Acar. 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, pages 426–437, New York, NY, USA, 2015. ACM.

[6] Henrik Plate, Serena Elisa Ponta, and Antonino Sabetta. Impact assessment for vulnerabilities in open-source software libraries. In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, pages 411–420. IEEE, 2015.

[7] Serena Elisa Ponta, Henrik Plate, and Antonino Sabetta. Beyond metadata: Code-centric and usage-based analysis of known vulnerabilities in open-source software. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sept 2018.

[8] Antonino Sabetta and Michele Bezzi. A practical approach to the automatic classification of security-relevant commits. In *34th IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sept 2018.

[9] Yaqin Zhou and Asankhaya Sharma. Automated identification of security issues from commit messages and bug reports. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 914–919, New York, NY, USA, 2017. ACM.