Toward Validation of Textual Information Retrieval Techniques for Software Weaknesses

Jukka Ruohonen(✉) and Ville Leppänen

Department of Future Technologies, University of Turku, Turku, Finland
{juanruo,ville.leppanen}@utu.fi

Abstract. This paper presents a preliminary validation of common textual information retrieval techniques for mapping unstructured software vulnerability information to distinct software weaknesses. The validation is carried out with a dataset compiled from four software repositories tracked in the Snyk vulnerability database. According to the results, the information retrieval techniques used perform unsatisfactorily compared to regular expression searches. Although the results vary from a repository to another, the preliminary validation presented indicates that explicit referencing of vulnerability and weakness identifiers is preferable for concrete vulnerability tracking. Such referencing allows the use of keyword-based searches, which currently seem to yield more consistent results compared to information retrieval techniques. Further validation work is required for improving the precision of the techniques, however.

Keywords: Text mining · Software vulnerability · Snyk · LSA CVE · CWE · NVD

1 Introduction

Software weaknesses—as cataloged in the so-called Common Weakness Enumeration (CWE) framework—are abstractions for security-related mistakes made in software development. Such weaknesses may lead to concrete software vulnerabilities. The CWE framework covers numerous different weaknesses, ranging from software design flaws to inadequate input validation, insecure maintaining of time and state, and lack of encapsulation [34]. The richness of the framework has ensured its usefulness for both research and practice. To name a few of the application domains, the CWE framework has been used for security (compliance) assessments [8,9], risk analysis [1,6], quantitative trend analysis [22], data mining [13], static source code analysis [25], dissemination of fuzzing results [15], and last but not least, education and security awareness [16]. Also text mining applications have been common, although there are still gaps in the literature.

Many of the text mining applications have relied on the Open Web Application Security Project (OWASP), which limits the generalizability of the applications [21,27]. Another gap relates to the common focus on CWE-based ontologies. Although such ontologies are useful for understanding and clustering weaknesses [2,10,36], these have a limited appeal for vulnerability tracking. Here, the term vulnerability tracking refers to the concrete, largely manual software engineering work required for archiving and documenting software vulnerabilities. If this work does not explicitly cover weaknesses, an application of an ontology-based technique must first solve the problem of extracting the CWEs from the typically more or less unstructured textual vulnerability data. Limited attention has also been given for the validity of the text mining applications.

To contribute toward sealing some of these gaps, this paper tentatively examines the validity of common textual information retrieval techniques for extracting CWEs from vulnerability databases. The extraction itself has practical value because many vulnerability databases do not catalog weaknesses, partially due to the complexity of the CWE framework and the manual work required [35]. By superseding the manual assignment of security bug reports to CWEs [8], automatic extraction can also facilitate empirical security research. It is important to further emphasize that the task differs from conventional information retrieval systems that can return multiple documents for a single query. In contrast, this paper adopts a much stricter constraint: each unique vulnerability should map to a single unique CWE-identified weakness. As elaborated in the opening Sect. 2, this constraint can be also used for comparing common textual information retrieval techniques against simple regular expression searches. The comparative results are presented in Sect. 3 and discussed in Sect. 4.

2 Materials

The following will outline the data sources, the subset of data used for the validation, the pre-processing routines, and the weights used for computation.

2.1 Data Sources

The dataset is compiled from three distinct but related sources. The first source is the conventional National Vulnerability Database (NVD) maintained by the National Institute of Standards and Technology (NIST) in cooperation with the non-profit MITRE corporation. This database provides one-to-one mappings between abstract weaknesses identified with CWEs and concrete vulnerabilities identified with Common Vulnerabilities and Exposures (CVEs). These mappings are based on expert opinion; during the archival of vulnerabilities to the database, NVD's maintainers derive the weaknesses from the concrete vulnerabilities archived. At the time of retrieving the database's content [23], there were 102 unique CWEs in the database once rejected CVEs were excluded. (These invalid cases are marked with the string `REJECTED` in the summary field of a CVE.) These CWEs were used for assembling the estimation subset soon discussed.

The second source is the CWE database maintained by MITRE in cooperation with volunteers and governmental sponsors. In total, there were 730 documented weaknesses in the database at the time of retrieval [18]. These weaknesses have been used to construct different ontologies [10,27,36], including the famous "seven pernicious kingdoms" of security-related programming mistakes [34]. Reflecting such ontologies, MITRE provides also many predefined views to subsets of the weaknesses archived. In the information retrieval context the relevant view is the one pointing to the CWEs used by NVD. Due to recent changes made [19], however, the predefined NVD-specific view is not suitable. Therefore, data is used from the CWE database only for the 102 weaknesses that are present also in NVD indirectly via CVE mappings. Although the present context is weaknesses, the idea here is similar to the enforcement of "one-to-one vulnerabilities" between vulnerability databases [35]. In contrast to some previous studies [10], all textual information is used for constructing the corpora. This information contains also meta-data strings, but the results reported did not differ much from those obtained by including only fields specific to natural language. The pre-processing described later on also filters out much of the meta-data.

The third and final source is the so-called Snyk database used for tracking security issues in open source software packages particularly in the web development context [32]. In contrast to the primary package managers used in Linux distributions, Snyk targets the secondary package managers and their repositories that are specific to programming languages. Although the Snyk database has seldom been used for research purposes, the underlying repositories have been studied extensively (see [20,29,33], for instance). The following four repositories are included in the dataset assembled: *Maven* (Java), *pip* (Python), *npm* (JavaScript), and *RubyGems* (Ruby). In addition to the web development context, these repositories were selected due to sufficient amounts of vulnerabilities reported for the packages within the repositories. Therefore, the selection used allows to check whether the results are specific only to some repositories.

As is typical in vulnerability tracking [4,21,30], the Snyk database contains first-order and (online) second-order relations. As illustrated in Fig. 1, these relations can be either direct or indirect with respect to CWE and CVE identifiers. For instance, the following vulnerability report (pymongo/40183) in the Snyk database represents a second-order indirect relation because the CWE in question can be mapped from the CVE visible in the link pointing to NVD:

```
## Overview
[‘pymongo‘](https://pypi.python.org/pypi/pymongo) is a Python driver for
MongoDB.

‘bson/_cbsonmodule.c‘ in the mongo-python-driver (aka.  pymongo) before
2.5.2, as used in MongoDB, allows context-dependent attackers to cause a
denial of service (NULL pointer dereference and crash) via vectors related
to decoding of an "invalid DBRef."
```

```
## References
- [NVD](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-2132)
- [Github Commit](https://github.com/mongodb/mongo-python-driver/commit/
a060c15ef87e0f0e72974c7c0e57fe811bbd06a2)
```

The first-order relations refer to the primary textual content stored to the database. As can be seen from the above excerpt, Snyk's maintainers archive each vulnerability with a brief textual description, which can be potentially mapped to a CWE with different text mining techniques. When constructing the corpora, all textual information is used except comment fields (lines starting with a #) and links to further online material (lines starting with a dash). The additional online material constitute the second-order relations. Most vulnerabilities archived to Snyk are accompanied with one or more links pointing to security advisories, blogs, mailing lists, bug trackers, version control systems, hosting services such as GitHub, and other vulnerability databases, including NVD in particular. The content behind each of these links was downloaded, and for each successful download, all textual information was further added to the corpora sans the hypertext markup language elements. Given the terseness of the primary (first-order) textual information in the Snyk database, the additional (second-order) online material is beneficial for enlarging the corpora observed.

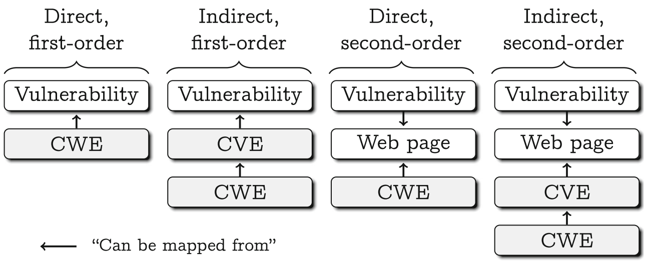

Fig. 1. An example of four abstract relations for software weaknesses

Thus, to summarize, the dataset contains textual information about software weaknesses from the CWE database, and first-order textual information about vulnerabilities from the Snyk database, as well as second-order textual information from online sources referenced in the Snyk database. The goal is to validate how well the vulnerabilities can be mapped to the CWEs. For this validation task, empirical estimation is carried out in a specific subset of the dataset.

2.2 Estimation Subset

A basic difficulty in classifying text documents is the typical absence of a ground truth against which text mining results can be compared. A typical way to tackle the issue is to compare text mining results against labels made by human experts [3]. The same applies in the vulnerability context [4,27]. However, the approach adopted takes a different path: the text mining results are compared against simple but relatively robust regular expression searches. For each vulnerability, the following two simple regular expressions were used to search for direct CWE mappings and indirect "CWE-from-CVE-in-NVD" mappings:

$$\texttt{(?:CWE)[-][0-9]\{1,\}} \quad \text{and} \quad \texttt{(?:CVE|CAN)[-][0-9]\{4\}-[0-9]\{4,\}}. \tag{1}$$

Only if a vulnerability matched only once from either one of the expressions, it was qualified to the estimation subset alongside the corresponding CWE, provided that the CWE was present among the 102 weaknesses in NVD. Thus, each vulnerability in the estimation subset can be mapped with regular expression searches to a single unique weakness. It can be further noted that techniques such as majority-voting are unsuitable. The reason is simple. Many of the second-order relations point to web pages that catalog all CVEs assigned for a given package. By per-vulnerability voting based on the frequency of CVEs (or CWEs) mentioned in such pages, the uniqueness condition would be lost. Such voting would presumably also lead to haphazard mappings. Given these remarks,

$$n_1 = 82 \text{ weaknesses and } n_2 = 585 \text{ vulnerabilities} \tag{2}$$

were qualified to the estimation subset. For comparing the regular expression searches against information retrieval techniques, a common metric can be used:

$$\text{Precision} = \frac{(\#\text{ same CWE})}{(\#\text{ same CWE}) + (\#\text{ different CWE})}, \tag{3}$$

where the numerator denotes the frequency of vulnerabilities that were assigned to the same CWEs by both the regular expression and information retrieval techniques. The second term in the denominator refers to the frequency of vulnerabilities assigned to different CWE identifiers by the two techniques. Due to the described operationalization of the estimation subset, additional metrics (such as recall and accuracy) are not meaningful—it cannot be deduced whether the remaining vulnerabilities in the Snyk database are relevant or irrelevant with respect to CWEs. That said, it would be possible to use more metrics by splitting the estimation subset further into training and test sets [22], but, as will be seen, the estimation subset and the precision metric are alone sufficient for summarizing the paper's empirical validation results.

An important further remark should be made. Although the two distinct terms in (3) connote with "true positives" and "false positives", these terms do not convey their usual meanings in the present context: it is impossible to say with certainty which one of the two techniques is (in)valid. On one hand, the whole vulnerability tracking infrastructure is based on the unique CVE identifiers that are easy to search based on keywords [13,31]. Therefore: if a vulnerability archived to Snyk can be mapped with a regular expression search to a unique CWE either directly or indirectly via a CVE identifier, there is a fairly good chance that the vulnerability truly reflects the given software weakness. This argument is reinforced by the sample: many of the vulnerabilities archived to Snyk are accompanied with CVE-specific links to NVD's website, which, in turn, contains also the CWEs assigned for the CVEs in question. On the other hand, some vulnerabilities archived to the Snyk database contain explicit notes that these particular vulnerabilities differ from those archived to NVD with some particular CVE identifiers. Such remarks cause some false positives for the regular expression searches but not for the information retrieval techniques.

2.3 Pre-processing

The pre-processing routine is fairly typical. To begin with, all textual data was *transformed* by removing hypertext markup language elements and then lower-casing all letters. After the transformation, the data was *tokenized* according to white space, punctuation characters, and other elements separating word boundaries. The resulting tokens were subsequently *trimmed* by excluding tokens containing non-alphabetic characters, tokens that are common stop words, as well as tokens with length less than three or more than twenty characters. The remaining tokens were finally *stemmed* with the Porter's [28] classical algorithm. The NLTK library [24] was used for the tokenization, stop words, and stemming.

The stemmed tokens were used for three types of "bag-of-words" matrices:

$$\mathbf{W}_{(k)}, \quad \text{where} \quad k \in \{1, 2, 3\}. \tag{4}$$

An element $w_{(k)ij}$ in a $\mathbf{W}_{(k)}$ denotes the *weight* given for the j:th tokenization-based k*-gram* (i.e., a unigram, a bigram, or a trigram) in the i:th *document* (i.e., either a given CWE or a given vulnerability). The number of rows remains constant in each matrix: $i = 1, \ldots, n_1 + n_2$, where n_1 and n_2 are defined in (2). Given that the use of bigrams and trigrams substantially enlarges the amount of data, the number columns varies: $j = 1, \ldots, m_k$, where m_k denotes the number of unique k-grams observed. It should be also noted that before assigning the actual weights, the underlying abstract data structures were pruned by excluding those k-grams that were present in a given corpus only twice or less.

2.4 Weights

Five different weights are used for the empirical validation. The *term* frequency (that is, in the present context, k-gram frequency) provides the starting point:

$$\text{TF} : w_{(k)ij} = f_{(k)ij}, \quad w_{(k)ij} \in \mathbf{W}_{(k)}, \tag{5}$$

where $f_{(k)ij}$ denotes the number of occurrences of the j:th k-gram in the document i. In addition, two simple TF-derivatives are empirically explored:

$$\text{TF-LOG} : w_{(k)ij} = \log(f_{(k)ij} + 1) \text{ and} \tag{6}$$

$$\text{TF-BOOLEAN} : w_{(k)ij} = \begin{cases} 0 \text{ if } f_{(k)ij} = 0, \\ 1 \text{ if } f_{(k)ij} > 0. \end{cases} \tag{7}$$

Although there is no particular prior reason to expect that the results would differ with respect to these three simple weighting schemes, the usual rationale for TF-LOG, for instance, is based on the reasoning that the importance of a term is unlikely to grow linearly as implied by the TF weights [26]. The same rationale works also behind the current *de facto* weighting scheme, the so-called term-frequency-inverse-document-frequency (TF-IDF). Despite of the scheme's complex name, the actual weighting is simple:

$$\begin{aligned} \text{TF-IDF}: \; & w_{(k)ij} = f_{(k)ij} \times \omega_{(k)j}, \quad \text{where} \\ & \omega_{(k)j} = \log([n_1 + n_2 + 1] \; / \; [\tilde{n}_{(k)j} + 1]) + 1 \end{aligned} \tag{8}$$

and $\tilde{n}_{(k)j}$ denotes the number of documents containing the j:th term (that is, k-gram). The IDF term, $\omega_{(k)j}$, used in (8) is one of the many smoothed variants of the standard IDF formula. With smoothing or no smoothing, the rationale behind the IDF term is to penalize the occurrence of common terms. Even though the TF-IDF weights perform well in many applied problems, the same rationale has been used to define also many other more or less analogous weights [14,26]. To empirically explore one of these IDF-based derivatives, the so-called (pivoted) document length normalization (DLM) is as a good choice as any. It is given by

$$\text{DLM-IDF}: w_{(k)ij} = \left[\frac{1 + \log(f_{(k)ij} + 1)}{(1 - \beta) + \beta(L_{(k)i} / \overline{L}_{(k)})} \right] \omega_{(k)j}, \tag{9}$$

where β is fixed to a scalar 0.2, $L_{(k)i}$ denotes the length of the i:th document (defined as the sum of all term frequencies), and $\overline{L}_{(k)}$ refers to the average document length in a whole corpus [12]. The five different weights enumerated are used for disseminating the subsequently discussed empirical validation results.

3 Results

The five different weights from (5) to (9) were assigned to each of the matrix types noted in (4). This assignment resulted in fifteen weight matrices that establish the basis for the empirical results. For computing the similarities between the CWEs and vulnerabilities observed, the standard cosine similarity is used for each of the matrices. If $\tilde{\mathbf{C}}$ denotes a document-by-document matrix of cosine similarities, a $n_2 \times n_1$ vulnerability-by-weakness matrix $\mathbf{C}$ can be obtained by deleting n_1 rows from $\tilde{\mathbf{C}}$ that refer to CWEs, and carrying out an analogous operation for the columns. Then, each vulnerability (row) is mapped to the corresponding maximum row value in $\mathbf{C}$. If ties are present (meaning that a vulnerability has the same maximum cosine similarity with two or more CWEs), the vulnerability is mapped to the CWE picked by the regular expression searches, provided that any of the maximum values map to this particular CWE identifier.

The so-called latent semantic analysis (LSA) was also briefly examined as an additional validation check. Although the usefulness for applied problems has been debated [7], LSA builds on the rationale that a rank-reduced similarity matrix based on linear combinations may better reveal the underlying latent

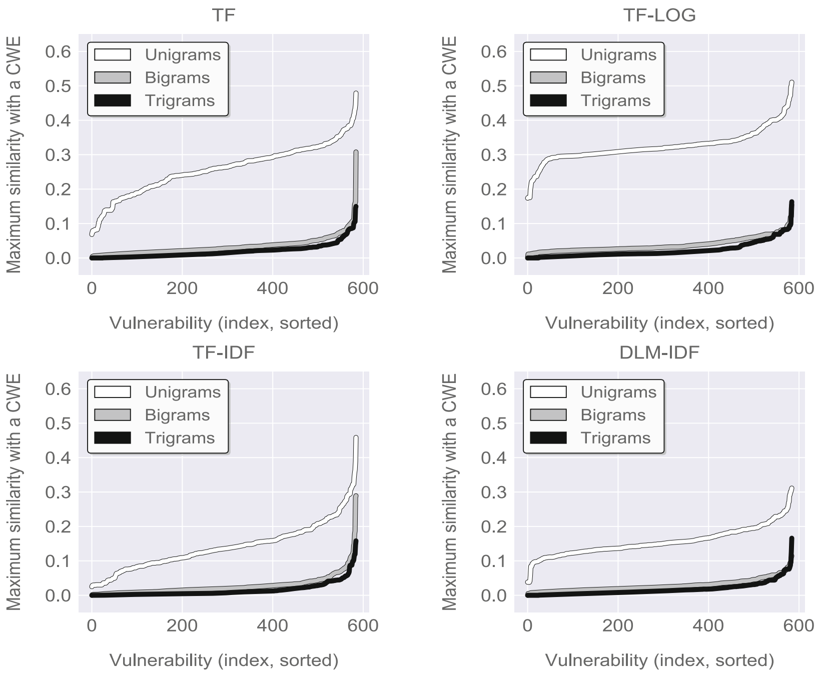

Fig. 2. Maximum cosine similarities according to four weights

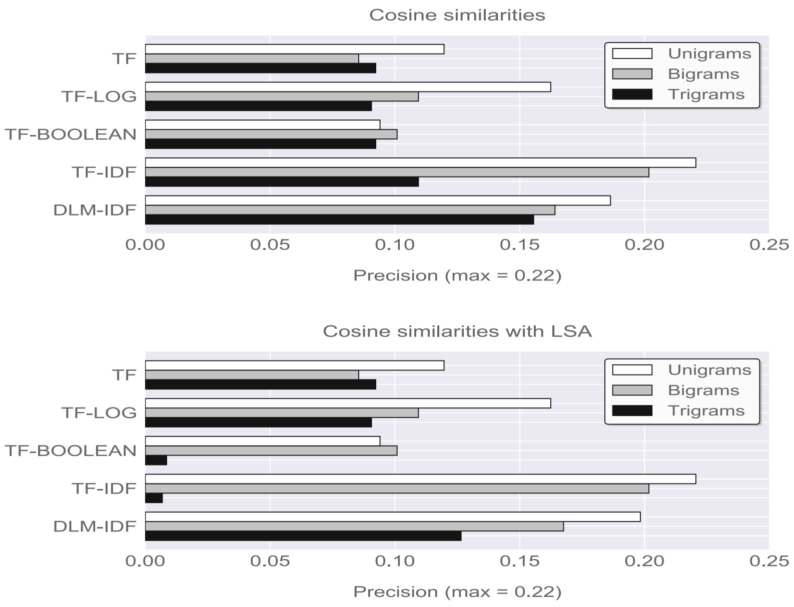

Fig. 3. Precision with five weights

structure of a corpus. Computation is simple but expensive: LSA is a straightforward application of the fundamental singular value decomposition, $\tilde{\mathbf{C}} = \mathbf{USV}^T$. Without delving into the linear algebra details, the actual reduction is also simple: a predefined number of the descending singular values in the diagonal matrix $\mathbf{S}$ is restricted to zero, after which $\tilde{\mathbf{C}}$ is reconstructed by using the manipulated diagonal matrix [3,11]. A simple heuristic was used for the manipulation: all singular values less than or equal to one were set to zero. For each of the fifteen matrices, roughly about a quarter of the singular values satisfied this heuristic.

Given these computational remarks, the results indicate only modest similarities between the CWEs and the vulnerabilities archived to Snyk. To illustrate this observation, Fig. 2 displays the maximum cosine similarities with four weights. Unigrams perform much better than bigrams or trigrams regardless of the weights used. The likely explanation relates to the numbers in Table 1, which shows the number of columns in $\mathbf{W}_{(k)}$ and the average document lengths used for the DLM-IDF weights in (9). That is: even after the pruning of the weight matrices, many of the bigrams and trigrams appear only in few documents. The TF and TF-LOG weights also attain higher maximum similarities than the two IDF-based weights shown in the two plots on the bottom row in Fig. 2. On average, however, the clear majority of the cosine similarities are well below 0.5 for all weights used. These low similarity values translate into poor precision. As illustrated in Fig. 3, the maximum precision is as low as 0.22. The TF-IDF weights perform the best in this regard. LSA does not improve the precision.

Table 1. Descriptive statistics

	Unigrams	Bigrams	Trigrams
Unique k-grams (m_k)	8435	32166	31745
Average document length ($\overline{L}_{(k)}$)	1095	935	839
• Average CWE length	424	357	252
• Average vulnerability length	1175	1016	921

Table 2. Average per-repository precision (TF-IDF)

	Maven	*pip*	*npm*	*RubyGems*
Unigrams	0.17	0.34	0.55	0.25
Bigrams	0.16	0.31	0.09	0.62
Trigrams	0.10	0.12	<0.01	0.50

All in all, the validity of common textual information retrieval techniques seems questionable for software weaknesses. That said, the results diverge considerably between the four repositories (see Table 2). While the results are consistently poor particularly for *Maven* and to a lesser extent *pip*, an average precision of 0.55 is obtained for *npm*. Interestingly, the use of bigrams raises the average precision to 0.62 for *RubyGems*. Thus, it can be also concluded that the poor precision cannot be generalized to all repositories or programming languages.

4 Discussion

This paper presented a preliminary validation of common textual information retrieval techniques for mapping vulnerabilities to weaknesses. When compared to basic regular expression searches, the techniques seem to perform poorly according to the results based on four repositories tracked in the Snyk vulnerability database. For searching specific vulnerabilities or weaknesses from software repositories, simple keyword searches based on CVE and CWE identifiers seem more robust. These commonly used [4,13,31] domain-specific searches could be augmented by the information retrieval techniques [5], however. In other words: it might be possible to prefer the regular expression searches as a primary retrieval technique and use the information retrieval techniques as a secondary method for retrieving additional content not captured by the keyword-based searches. It is also important to stress that the results vary across repositories. This observation hints that the choice over particular security-related corpora has likely a strong effect upon the vulnerability-CWE mappings.

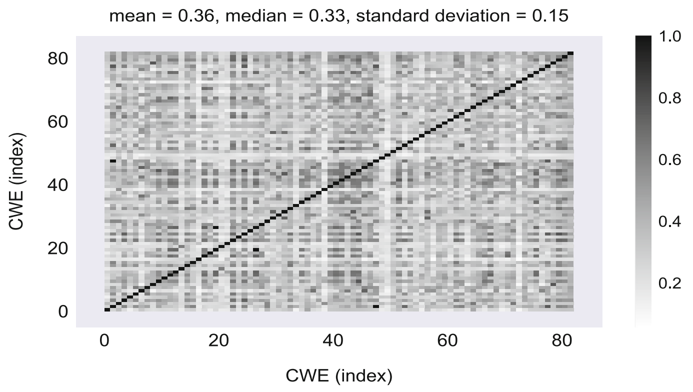

Fig. 4. Similarities between weaknesses (TF-IDF, unigrams)

Three points can be noted about limitations and directions for further research. First, as CWE is a hierarchical classification system, it might be argued that larger ontology-based weakness groups should be used. However, all of the individual CWEs observed in this paper have been tracked in NVD, which undermines the rationale for using such CWE groups in the present context. The construct validity is also undermined by the illustration in Fig. 4, which shows that the CWEs observed are not very similar with respect to each other.

Second, many computational checks could be done to further validate the vulnerability-CWE mappings. For instance, the cosine similarity used could be verified against other commonly used similarity metrics. (It can be noted that the so-called Jaccard similarity performs even worse, however.) Further examples include the calibration of the LSA's reduction step, the use of a more aggressive stemming algorithm, and the examination of part-of-speech tagging.

Third, it seems reasonable to try to either enlarge or enrich the datasets used for validation. The textual information stored to Snyk, CWE, OWASP, and related databases is terse in terms of natural language and prose, but there are technical terms that should be specifically weighted. If the small excerpt shown in Subsect. 2.1 is taken as an example, the trigrams `denial of service` and `NULL pointer dereference` should attain higher weights than any of the other k-grams. Such domain-specific weights entail the construction of a reference corpus. Given the generally poor precision reported and the variance across repositories, it may also be that neither CWE nor OWASP are ideal for a construction of a reference corpus. Instead, language-specific guides for secure programming [17] may be more suitable, among other potential sources related to software weaknesses and vulnerabilities. As an alternative to enrichment, big data analysis is also plausible. Due to the central role of CVE identifiers (cf. Fig. 1), web crawling could be used to gather a truly massive dataset for text mining. Recent work [4] shows also some promise for web crawling approaches. But the larger the datasets, the coarser the mappings, and the bigger the validity concerns.

References

1. Alsaleh, M.N., Al-Shaer, E., Husari, G.: ROI-driven cyber risk mitigation using host compliance and network configuration. J. Netw. Syst. Manag. **25**(4), 759–783 (2017)
2. Bojanova, I., Black, P.E., Yesha, Y., Wu, Y.: The bugs framework (BF): a structured approach to express bugs. In: Proceedings of the IEEE International Conference on Software Quality, Reliability and Security (QRS 2016), Vienna, pp. 175–182. IEEE (2016)
3. dos Santos, J.C.A., Favero, E.L.: Practical use of a latent semantic analysis (LSA) model for automatic evaluation of written answers. J. Braz. Comput. Soc. **21**(1), 1–21 (2015)
4. Du, D.: Refining traceability links between vulnerability and software component in a vulnerability knowledge graph. In: Mikkonen, T., Klamma, R., Hernández, J. (eds.) ICWE 2018. LNCS, vol. 10845, pp. 33–49. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91662-0_3
5. Fautsch, C., Savoy, J.: Adapting the TF IDF vector-space model to domain specific information retrieval. In: Proceedings of the 2010 ACM Symposium on Applied Computing (SAC 2010), Sierre, pp. 1708–1712. ACM (2010)
6. Franqueira, V.N.L., Tun, T.T., Yu, Y., Wieringa, R., Nuseibeh, B.: Risk and argument: a risk-based argumentation method for practical security. In: Proceedings of the IEEE 19th International Requirements Engineering Conference (RE 2011), Trento, pp. 239–248. IEEE (2011)

7. Gamallo, P., Bordag, S.: Is singular value decomposition useful for word similarity extraction? Lang. Resour. Eval. **45**(2), 95–119 (2011)
8. Goseva-Popstojanova, K., Tyo, J.: Experience report: security vulnerability profiles of mission critical software: empirical analysis of security related bug reports. In: Proceedings of the IEEE 28th International Symposium on Software Reliability Engineering (ISSRE 2017), Toulouse, pp. 152–163. IEEE (2017)
9. Hale, M.L., Gamble, R.F.: Semantic hierarchies for extracting, modeling, and connecting compliance requirements in information security control standards. Requir. Eng. 1–38 (2018). Published online in December 2017
10. Han, Z., Li, X., Liu, H., Xing, Z., Feng, Z.: DeepWeak: reasoning common software weaknesses via knowledge graph embedding. In: Proceedings of the IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER 2018), Campobasso, pp. 456–466. IEEE (2018)
11. Hussain, S.F., Suryani, A.: On retrieving intelligently plagiarized documents using semantic similarity. Eng. Appl. Artif. Intell. **45**, 246–258 (2015)
12. Ibrahim, O.A.S., Landa-Silva, D.: Term frequency with average term occurrences for textual information retrieval. Soft. Comput. **20**(8), 3045–3061 (2016)
13. Jimenez, M., Papadakis, M., Traon, Y.L.: An empirical analysis of vulnerabilities in OpenSSL and the Linux Kernel. In: Proceedings of the 23rd Asia-Pacific Software Engineering Conference (APSEC 2016), Hamilton, pp. 105–112. IEEE (2016)
14. Jin, R., Chai, J.Y., Si, L.: Learn to weight terms in information retrieval using category information. In: Proceedings of the 22nd International Conference on Machine Learning (ICML 2005), Bonn, pp. 353–360. ACM (2005)
15. Kang, J., Park, J.H.: A secure-coding and vulnerability check system based on smart-fuzzing and exploit. Neurocomputing **256**, 23–34 (2017)
16. Martin, R.A., Barnum, S.: Common weaknesses enumeration (CWE) status update. ACM SIGAda Ada Lett. Arch. **XXVII**(1), 88–91 (2008)
17. McManus, J.: SEI CERT Oracle Coding Standard for Java, Carnegie Mellon University, Software Engineering Institute (SEI) (2018). https://wiki.sei.cmu.edu/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java. Accessed May 2018
18. MITRE: Common Weaknesses Enumeration, CWE List Version 3.1, CWE Comprehensive View (2018). http://cwe.mitre.org/data/csv/2000.csv.zip. Accessed April 2018
19. MITRE: CWE VIEW: Weaknesses Originally Used by NVD from 2008 to 2016 (2018). http://cwe.mitre.org/data/definitions/635.html. Accessed January 2018
20. Mitropoulos, D., Karakoidas, V., Louridas, P., Gousios, G., Spinellis, D.: The bug catalog of the maven ecosystem. In: Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014), Hyderabad, pp. 372–375. ACM (2014)
21. Muñoz, F.R., Villalba, L.J.G.: An algorithm to find relationships between web vulnerabilities. J. Supercomput. **74**(3), 1061–1089 (2018)
22. Murtaza, S., Khreich, W., Hamou-Lhadj, A., Bener, A.B.: Mining trends and patterns of software vulnerabilities. J. Syst. Softw. **117**, 218–228 (2016)
23. NIST: NVD Data Feeds, National Institute of Standards and Technology (NIST) (2018). https://nvd.nist.gov/vuln/data-feeds. Accessed April 2018
24. The Natural Language Toolkit (NLTK): NLTK 3.2.5 Documentation (2017). http://www.nltk.org. Accessed April 2018
25. Oyetoyan, T.D., Milosheska, B., Grini, M., Soares Cruzes, D.: Myths and facts about static application security testing tools: an action research at Telenor digital. In: Garbajosa, J., Wang, X., Aguiar, A. (eds.) XP 2018. LNBIP, vol. 314, pp. 86–103. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91602-6_6

26. Paik, J.H.: A novel TF-IDF weighting scheme for effective ranking. In: Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2013), Dublin, pp. 343–352. ACM (2013)
27. Peclat, R.N., Ramos, G.N.: Semantic analysis for identifying security concerns in software procurement edicts. New Gener. Comput. **36**(1), 21–40 (2018)
28. Porter, M.F.: An algorithm for suffix stripping. Program **14**(3), 130–137 (1980)
29. Raemaekers, S., van Deursen, A., Visser, J.: Semantic versioning and impact of breaking changes in the maven repository. J. Syst. Softw. **129**, 140–158 (2017)
30. Ruohonen, J.: Classifying web exploits with topic modeling. In: Proceedings of the 28th International Workshop on Database and Expert Systems Applications (DEXA 2017), Lyon, pp. 93–97. IEEE (2017)
31. Ruohonen, J., Rauti, S., Hyrynsalmi, S., Leppänen, V.: Mining social networks of open source CVE coordination. In: Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement (IWSM Mensura 2017), Gothenburg, pp. 176–188. ACM (2017)
32. Snyk Ltd.: Snyk Vulnerability Database (2018). https://github.com/snyk/vulnerabilitydb. Accessed April 2018
33. Squire, M.: Data sets describing the circle of life in Ruby hosting, 2003–2016. Empir. Softw. Eng. **23**(2), 1123–1152 (2018)
34. Tsipenyuk, K., Chess, B., McGraw, G.: Seven Pernicious Kingdoms: a taxonomy of software security errors. IEEE Secur. Priv. **3**(6), 81–84 (2005)
35. Wen, T., Zhang, Y., Wu, Q., Yang, G.: ASVC: an automatic security vulnerability categorization framework based on novel features of vulnerability data. J. Commun. **10**(2), 107–116 (2015)
36. Wu, Y., Gandhi, R.A., Siy, H.: Using semantic templates to study vulnerabilities recorded in large software repositories. In: Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems (SESS 2010), Cape Town, pp. 22–28. ACM (2010)