Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 184 (2021) 953–958

Procedia Computer Science

www.elsevier.com/locate/procedia

The 3rd International Symposium on Machine Learning and Big Data Analytics for Cybersecurity and Privacy (MLBDACP 2021), March 23 - 26, 2021, Warsaw, Poland

An Automated Post-Mortem Analysis of Vulnerability Relationships using Natural Language Word Embeddings

Benjamin S. Meyers[a,*], Andrew Meneely[a]

[a]*Department of Software Engineering, Rochester Institute of Technology, Rochester, NY 14623, USA*

Abstract

The daily activities of cybersecurity experts and software engineers—code reviews, issue tracking, vulnerability reporting—are constantly contributing to a massive wealth of security-specific natural language. In the case of vulnerabilities, understanding their causes, consequences, and mitigations is essential to learning from past mistakes and writing better, more secure code in the future. Many existing vulnerability assessment methodologies, like CVSS, rely on categorization and numerical metrics to glean insights into vulnerabilities, but these tools are unable to capture the subtle complexities and relationships between vulnerabilities because they do not examine the nuanced natural language artifacts left behind by developers. In this work, **we want to discover unexpected relationships between vulnerabilities with the goal of improving upon current practices for post-mortem analysis of vulnerabilities.** To that end, we trained word embedding models on two corpora of vulnerability descriptions from Common Vulnerabilities and Exposures (CVE) and the Vulnerability History Project (VHP), performed hierarchical agglomerative clustering on word embedding vectors representing the overall semantic meaning of vulnerability descriptions, and derived insights from vulnerability clusters based on their most common bigrams. We found that (1) vulnerabilities with similar consequences and based on similar weaknesses are often clustered together, (2) clustering word embeddings identified vulnerabilities that need more detailed descriptions, and (3) clusters rarely contained vulnerabilities from a single software project. Our methodology is automated and can be easily applied to other natural language corpora. We release all of the corpora, models, and code used in our work.

Keywords: Cybersecurity; Word Embedding; Vulnerabilities; Clustering; Exploratory Data Analysis; Natural Language Processing

1. Introduction

A key goal of any software development team is to produce the most robust and secure software possible, but security is an emergent property of software—software is only secure until a new vulnerability is discovered. Through

* Corresponding author.
E-mail address: bsm9339@rit.edu

1877-0509
10.1016/j.procs.2021.04.018

studying historical vulnerabilities and understanding their causes, consequences, and mitigations we can learn to engineer secure software. Prominent post-mortem analyses of vulnerabilities, by both developers and researchers, involves the categorization and the computation of metrics, such as severity and impact [4, 5], but the natural language descriptions of vulnerabilities are invaluable resources that reveal more nuanced information than just severity scores.

However, studying large corpora of security-focused natural language is a challenging task that can be simplified using natural language processing techniques, as demonstrated in previous work: Pletea *et al.* studied the sentiment of developers' natural language in GitHub commits/pull requests and discovered that security-related discussions tend to be more negative. Munaiah *et al.* found that code reviews tend to miss vulnerabilities when developers are too vague (low syntactic complexity) and make too many assertions (high propositional density) [9]. Meyers *et al.* explored pragmatic characteristics of natural language (*e.g.* politeness, formality, uncertainty) in bug report discussions and found that effective security conversations tend to be more polite and less formal [6]. Word Embeddings (WE) have been used to identify bug reports related to security [10] and to match security professionals with security tasks [8].

Our goal is to discover new insights into historical vulnerabilities by developing a method of automatically inferring the relationships between them. To that end, we train WE models on two corpora—189,734 vulnerability descriptions from Common Vulnerabilities and Exposures (CVE) and 721 vulnerability descriptions from the Vulnerability History Project (VHP). Then we group vulnerabilities together, based on the semantic meaning of their descriptions, using hierarchical agglomerative clustering. We derive insights about the relationships between vulnerabilities from the most common bigrams (sequences of two consecutive words) across vulnerability descriptions within clusters. Our corpora, models, and code are available on GitHub[1]. We address the following research question.

RQ: What kinds of vulnerabilities are similar to each other? We expect vulnerabilities with the same consequences (*e.g.* denial of service, integrity violations) and vulnerabilities with the same weaknesses (*e.g.* buffer overflow, SQL injection) to be clustered together. We also expect vulnerabilities in the same project (*e.g.* Chromium, Apache Tomcat) to be clustered together.

2. Background

Word Embeddings: WE are vector-space representations of the semantic meaning of words within a corpus. WE are learned using neural networks, which compute fixed-length *word vectors* of real numbers representing the semantic meaning of each word in a corpus [7]. The numeric values in a word vector represent different dimensions of meaning associated with a word, such that words with similar meanings will have similar word vectors. For example, the words 'vulnerability' and 'flaw' will have similar word vectors because their meanings are similar and they are used in similar contexts, whereas the words 'hash' and 'browser' will have word vectors that are more dissimilar. At a document level, word vectors can be added together and the resultant *document vector* represents a numerical summary of the overall meaning of the document. Document vectors can be clustered to group semantically similar documents together. However, these document vectors are not easily interpreted by humans, so we use them for clustering and then examine common bigrams in clusters to derive insights, as described in the next section.

Common Vulnerabilities and Exposures (CVE): CVE[2] is a dictionary of publicly disclosed vulnerabilities (and their descriptions) in software projects. Many taxonomies—*e.g.* Common Attack Pattern Enumeration and Classification (CAPEC), Common Weakness Enumerations (CWE), National Vulnerability Database (NVD)—reference individual CVEs, making CVEs a powerful tool for shared reference and understanding when discussing vulnerabilities. We downloaded CVEs in CSV format from the CVE website.

3. Data Preprocessing, Modeling, and Clustering Methodology

Vulnerability History Project (VHP): VHP[3] is a museum of security mistakes curated by one of the authors and students in computing majors at the Rochester Institute of Technology (RIT). VHP collects CVEs from four

[1] https://github.com/meyersbs/MLBDACP21
[2] https://cve.mitre.org/
[3] http://vulnerabilityhistory.org/

Table 1. Postprocessed corpora statistics.

	# Documents	Total # Words	Total # Unique Words	Avg. Words/Document
CVE	189,734	5,334,268	122,896	28.11
VHP	721	51,861	5,434	71.92

Table 2. Word embedding model hyperparameters.

Algorithm	# Dimensions	Context Window	Negative Sampling Rate	Min. Word Frequency
Skip-Gram	150	5	10	5

open source projects: Chromium[4], HTTPD[5], Struts[6], and Tomcat[7]. Students and researchers at RIT examine the entire context of the vulnerabilities, including the CVE description, the fix, and the estimated origin commit, called the vulnerability-contributing commit. These entries provide more detailed descriptions than standard CVE descriptions. Descriptions, mistakes made, project associations, and discovery dates were exported using the VHP website's API.

Human Annotations: We manually annotated each vulnerability in the VHP corpus with a set of security topics (*e.g.* complex inputs, cryptography, integrity, spoofing) based on the vulnerability descriptions. In total, we manually identified 132 unique security topics. These annotations represent our preconceived expectations for clusters.

The remainder of this section outlines our methodology for training WE models and clustering vulnerability descriptions using document vectors derived from the WE models:

(1) Text Preprocessing: Before training WE models, we clean up the data by lowercasing and removing stop words (`nltk.corpus.stopwords`), punctuation, and special characters. We also use regular expressions to replace version numbers, numerical values, and hexcodes with special tokens ('VERVER', 'NUMNUM', and 'HEXHEX', respectively). Finally, we lemmatize the words in each corpus using the `WordNetLemmatizer` implemented in NLTK [1]. Table 1 shows statistics for each corpus after preprocessing.

(2) Training WE Models: We trained two WE models, CVE_WE and VHP_WE, on vulnerability descriptions from the CVE and VHP corpora, respectively. WE are learned using the implementation of `word2vec` (as described in Mikolov *et al.* [7]) in the Python library `gensim` [12]. Following Mumtaz *et al.* [8], our WE models are skip-gram models that learn 150-dimensional word vectors with a sliding context window of five and a negative sampling rate of ten. We opt for a minimum word frequency of five since our corpora are smaller than those in Mumtaz *et al.*. Table 2 summarizes the hyperparameters we used.

(3) Clustering: We performed clustering on document vectors (computed from the WE models) for each vulnerability in the VHP corpus. Document vector values were normalized to values between $[-1, 1]$ using the `MinMaxScaler` from `scikit-learn` [11]. We used hierarchical agglomerative clustering (implemented in `scikit-learn`) since we do not know how many clusters there are in the VHP corpus and we expect clusters to also be related to each other. In agglomerative clustering, each data point is initially considered its own cluster, then clusters are merged using a linkage criterion and a distance metric. We use Ward's linkage method, which minimizes the variance between clusters being merged [13]. The implementation of Ward's method in `scikit-learn` only supports Euclidean distance.

(4) Choosing the Number of Clusters (K): We performed agglomerative clustering starting at $K = 3$ and chose the maximum value of K such that all clusters had at least two vulnerability descriptions in them, since singleton clusters do not immediately provide any insight into the relationships between vulnerabilities.

(5) Interpreting Clusters: Document vectors are not very interpretable to humans, so we instead show the most frequent bigrams within a cluster. We also examine the most frequent words that appear in at least half of the vulnerability descriptions within a cluster, as well as our own manual annotations of security topics.

[4] https://zenodo.org/badge/latestdoi/67513158
[5] https://zenodo.org/badge/latestdoi/67513281
[6] https://zenodo.org/badge/latestdoi/167394888
[7] https://zenodo.org/badge/latestdoi/166920755

4. Results

Using the approach outlined in Section 3, we determined the number of clusters (K) for the CVE_WE and VHP_WE document vectors to be $K = 14$ and $K = 11$, respectively. Table 3 shows the number of vulnerabilities assigned to each cluster and the ten most frequent bigrams within each cluster.

CVE_WE: We see that the majority (ten) of clusters contain vulnerabilities related to denial of service (DoS) attacks, but these clusters are still diverse: Cluster 1 contains vulnerabilities involving memory access (use-after-free) and arbitrary code execution, while cluster 2 involves race conditions in addition to use-after-free mistakes and arbitrary code execution; Cluster 11 contains vulnerabilities involving buffer and integer overflows, while cluster 9 contains vulnerabilities where insufficient testing and coding/design mistakes led to DoS. The clusters not related to DoS attacks are more interesting: Vulnerabilities in cluster 3 deal with origin policy violations in web browsers and confidentiality violations, while cluster 10's vulnerabilities seem to involve web applications/extensions, and vulnerabilities in cluster 13 deal with URLs.

VHP_WE: Again, we see that most (eight) of the clusters contain vulnerabilities related to DoS attacks, but our clustering approach identifies DoS-related vulnerabilities with nuanced differences: Vulnerabilities in clusters 0 and 6 both involve buffer overflows, but vulnerabilities in cluster 0 deal with javascript code and the Document Object Model (DOM), while those in cluster 6 involve remote attackers and use-after-free coding mistakes; Cluster 10's vulnerabilities involve input validation and insufficient unit tests, while cluster 3's vulnerabilities involve use-after-free coding mistakes. Looking at clusters of vulnerabilities not related to DoS attacks, we see that vulnerabilities in clusters 7 and 8 both involve origin policy violations, but those in cluster 8 tend to be design mistakes.

Interestingly, regardless of the WE model used to compute document vectors, our clustering approach identifies a cluster of vulnerabilities (CVE_WE: 12, VHP_WE: 5) that are embargoed or contain little information (an average of six and nine words per vulnerability description, respectively), showing that clustering using WE vectors could potentially be used to identify vulnerabilities with descriptions that need more detail. This should be explored further, since minimal vulnerability descriptions do little to aid in post-mortem analysis.

We expected to find clusters that only contained vulnerabilities from a single project, but that is only true for two clusters (CVE_WE: 2, 12). 14 clusters (CVE_WE: 1, 4, 5, 6, 9, 10, 11; VHP_WE: 0, 1, 2, 3, 8, 9, 10) contain vulnerabilities from at least three projects, with 11 of those containing vulnerabilities from all four projects (CVE_WE: 1, 4, 6, 9, 10; VHP_WE: 1, 2, 3, 8, 9, 10). Contrary to expectations, vulnerabilities were not clustered together by project, suggesting that the natural language used to describe vulnerabilities transcends project boundaries.

RQ: What kinds of vulnerabilities are similar to each other? As expected, we observe clusters of vulnerabilities with similar consequences (*e.g.* denial of service, information disclosure) and based upon the same weaknesses (*e.g.* buffer overflow, use-after-free), but only in limited circumstances; we expected to see more varied clusters (*e.g.* integrity violations, spoofing attacks, cross-site scripting, user interfaces), but that was not the case. During manual annotation, we noticed small groups of vulnerabilities involving technologies (*e.g.* archives, clipboards, pseudo-random number generators), specific types of attacks (*e.g.* sniffing, man-in-the-middle, phishing), programming concepts (*e.g.* recursion, type confusion, encapsulation), and security concepts (*e.g.* defense in depth, security by obscurity, access controls)—none of the clusters capture these relationships, but they are still valid relationships that should be considered during post-mortem analysis of vulnerabilities. Perhaps there are sub-clusters contained within the ones automatically discovered that do capture these small relationships—we hope to explore this in future work.

5. Summary & Future Work

In this work, we explored the use of Word Embedding (WE) models and hierarchical agglomerative clustering to discover unexpected relationships between vulnerabilities. Many of the resulting clusters contained vulnerabilities with the same consequences (*e.g.* denial of service) and the same weaknesses (*e.g.* buffer overflow), but a lot of the clusters we expected (*e.g.* man-in-the-middle attacks) to see were not found, suggesting the need to examine the more granular (smaller) clusters in the hierarchy in future work. We found that contrary to our expectations, clusters did not often contain vulnerabilities from only a single project. We also observed that clustering grouped vulnerabilities with minimal descriptions and embargoed vulnerabilities together, which suggests that clustering WE can be used to identify vulnerability descriptions that need to be more detailed.

Table 3. Clusters identified via hierarchical agglomerative clustering.

Model	Cluster	# Vulns.	10 Most Frequent Bigrams w/ Frequencies
CVE_WE	0	2	shadow dom (10), fastcgi server (3), past end (3), http header (3), web server (3), null character (3), dom document (2), document object (2), object model (2), denial service (2)
	1	53	denial service (26), use free (16), arbitrary code (15), coding mistake (14), design mistake (14), cause denial (13), google chrome (13), execute arbitrary (10), third party (9), VERVER VERVER (9)
	2	30	use free (23), denial service (19), google chrome (15), cause denial (13), coding mistake (12), remote attackers (10), service attack (9), free vulnerability (9), arbitrary code (9), race condition (9)
	3	34	origin policy (26), google chrome (12), web page (10), remote attackers (7), design mistake (7), url bar (7), web pages (7), access data (7), web browser (6), sensitive information (6)
	4	146	coding mistake (37), denial service (37), remote attackers (31), google chrome (29), cause denial (23), mistake made (13), cross site (13), design mistake (13), allows remote (13), attackers cause (10)
	5	59	denial service (41), use free (28), cause denial (25), coding mistake (21), free vulnerability (21), attackers cause (15), memory freed (13), google chrome (13), mistake made (11), service attack (11)
	6	77	coding mistake (18), design mistake (14), http request (11), denial service (10), google chrome (9), remote attackers (9), input validation (8), attacker could (8), apache struts (8), blank page (7)
	7	41	remote attackers (10), google chrome (7), denial service (7), cause denial (6), coding mistake (5), allows remote (5), spoofing omnibox (4), attackers cause (4), escape sequences (3), allowing remote (3)
	8	11	denial service (8), buffer overflow (6), shared memory (6), javascript code (5), attacker could (4), could cause (4), wrap around (4), invalid memory (4), xml file (4), heap buffer (4)
	9	83	denial service (19), unit tests (16), coding mistake (15), design mistake (12), make sure (8), lack testing (7), cve 2017 (7), mistake made (6), even though (6), specially crafted (6)
	10	14	web applications (9), object prototype (6), quirks mode (6), new tab (5), coding mistake (5), fullscreen widget (5), web page (5), web application (4), remote attackers (4), chromium extensions (4)
	11	45	denial service (39), buffer overflow (27), cause denial (18), integer overflow (13), remote attackers (12), coding mistake (11), attackers cause (10), heap based (9), google chrome (9), based buffer (8)
	12	123	use free (25), embargoed use (17), embargoed bounds (10), spoofing omnibox (9), free pdfium (8), url spoofing (8), type confusion (8), embargoed url (7), bounds write (6), embargoed heap (6)
	13	4	example com (5), vary header (5), bug report (4), origin policy (3), address bar (3), http example (3), harmony proxy (2), web page (2), mime sniffing (2), proposed solution (2)
VHP_WE	0	13	denial service (11), shadow dom (10), buffer overflow (8), double free (8), shared memory (6), javascript code (5), design mistake (5), attacker could (4), cause denial (4), could cause (4)
	1	74	denial service (17), attacker could (13), google chrome (13), arbitrary code (13), web applications (13), VERVER VERVER (12), origin policy (12), coding mistake (11), design mistake (11), cross site (10)
	2	117	denial service (31), remote attackers (30), coding mistake (27), cause denial (21), google chrome (20), allows remote (11), attackers cause (11), design mistake (9), mistake made (9), allowed remote (7)
	3	105	denial service (32), coding mistake (19), cause denial (16), remote attackers (13), google chrome (12), design mistake (12), simple coding (9), input validation (9), service attack (8), seems like (7)
	4	37	use free (28), denial service (26), remote attackers (17), cause denial (17), google chrome (17), arbitrary code (14), mistake made (13), free vulnerability (12), coding mistake (12), buffer overflow (12)
	5	144	use free (25), embargoed use (17), spoofing omnibox (13), url spoofing (10), embargoed bounds (10), free pdfium (8), type confusion (8), embargoed url (7), bounds write (6), embargoed heap (6)
	6	46	denial service (43), cause denial (25), coding mistake (19), use free (19), buffer overflow (18), attackers cause (16), remote attackers (14), google chrome (13), free vulnerability (12), service attack (11)
	7	12	origin policy (17), web page (11), access data (6), object prototype (6), bug report (5), evil com (5), cross origin (5), example com (5), vary header (5), remote attackers (4)
	8	67	google chrome (17), remote attackers (13), design mistake (13), coding mistake (12), mistake made (10), cross site (10), http request (9), origin policy (8), site scripting (8), chrome browser (7)
	9	47	denial service (28), coding mistake (19), use free (17), google chrome (13), cause denial (13), unit tests (12), design mistake (11), mistake made (10), seems like (9), buffer overflow (8)
	10	60	denial service (20), unit tests (18), coding mistake (16), make sure (9), design mistake (8), google chrome (7), input validation (6), code execution (6), service attacks (5), third party (5)

As this was an exploratory study using WE for clustering natural language from vulnerability descriptions, there is a wide range of potential future work. Going forward, we intend to examine how these clusters are related to each other in the hierarchy. We also hope to explore other clustering algorithms (*e.g.* DBSCAN, K-Means, MeanShift) and models trained on different sources of cybersecurity natural language (*e.g.* Common Weakness Enumerations). We also want to explore topical word embeddings [3] and Latent Dirichlet Allocation (LDA) topic modeling [2]. More importantly, we wish to explore the following research questions:

- **RQ: How are vulnerability clusters related to each other?** What can we learn about vulnerabilities and their relationships by comparing super- and sub-clusters identified by hierarchical clustering?
- **RQ: How are vulnerabilities related across the dimension of time?** Based on past development experience, we anticipate that similar types of vulnerabilities (*e.g.* XSS, integer overflow) are discovered and disclosed around the same time. What can we learn from vulnerabilities discovered at a steady rate independent of time?
- **RQ: Does training with more diverse corpora yield better clusters?** We believe that training WE models on security-focused natural language from other modalities—*e.g.* Common Weakness Enumerations, commit messages for vulnerability fixes, code review comments—will result in better clustering, since the word (and resulting document) vectors will have been learned using more natural language context.
- **RQ: Can clustering with word embeddings capture more subtle relationships between vulnerabilities?** Contrary to our expectations, clustering did not group vulnerabilities dealing with similar concepts (*e.g.* internationalization and fonts, SQL and databases) together. Can we find a different set of training parameters or a different clustering algorithm that results in vulnerabilities dealing with these concepts being clustered together?

Acknowledgements

This work has been sponsored in part by the National Science Foundation grant 1922169, and by a Department of Defense DARPA SBIR program. The authors wish to thank Dr. Emily Prud'hommeaux in the Department of Computer Science at Boston College for providing resources and guidance.

References

[1] Bird, S., Klein, E., Loper, E., 2009. Natural language processing with Python: analyzing text with the natural language toolkit. O'Reilly Media, Inc.
[2] Blei, D.M., Ng, A.Y., Jordan, M.I., 2003. Latent dirichlet allocation. Journal of machine Learning research 3, 993–1022.
[3] Liu, Y., Liu, Z., Chua, T.S., Sun, M., 2015. Topical word embeddings, in: Twenty-ninth AAAI conference on artificial intelligence, Citeseer.
[4] Martin, B., Christey, S.C., 2014. Common weakness scoring system (cwss). https://cwe.mitre.org/cwss/cwss_v1.0.1.html.
[5] Mell, P., Scarfone, K., Romanosky, S., 2006. Common vulnerability scoring system. IEEE Security & Privacy 4, 85–89.
[6] Meyers, B.S., Munaiah, N., Meneely, A., Prud'hommeaux, E., 2019. Pragmatic characteristics of security conversations: an exploratory linguistic analysis, in: 2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), IEEE. pp. 79–82.
[7] Mikolov, T., Chen, K., Corrado, G., Dean, J., 2013. Efficient estimation of word representations in vector space. arXiv:1301.3781 preprint .
[8] Mumtaz, S., Rodriguez, C., Benatallah, B., Al-Banna, M., Zamanirad, S., 2020. Learning word representation for the cyber security vulnerability domain, in: 2020 International Joint Conference on Neural Networks (IJCNN), IEEE. pp. 1–8.
[9] Munaiah, N., Meyers, B.S., Alm, C.O., Meneely, A., Murukannaiah, P.K., Prud'hommeaux, E., Wolff, J., Yu, Y., 2017. Natural language insights from code reviews that missed a vulnerability, in: International Symposium on Engineering Secure Software and Systems, Springer. pp. 70–86.
[10] Palacio, D.N., McCrystal, D., Moran, K., Bernal-Cárdenas, C., Poshyvanyk, D., Shenefiel, C., 2019. Learning to identify security-related issues using convolutional neural networks, in: 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE. pp. 140–144.
[11] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830.
[12] Řehůřek, R., Sojka, P., 2010. Software Framework for Topic Modelling with Large Corpora, in: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, ELRA, Valletta, Malta. pp. 45–50. http://is.muni.cz/publication/884893/en.
[13] Ward Jr, J.H., 1963. Hierarchical grouping to optimize an objective function. Journal of the American statistical association 58, 236–244.