VrT: A CWE-Based Vulnerability Report Tagger

Machine Learning Driven Cybersecurity Tool for Vulnerability Classification

Fahad A. Alshaya Sultan S. Alqahtani Yasser A. Alsamel
Department of Computer Science
Al Imam Mohammad Bin Saud Islamic University
Riyadh, Saudi Arabia
faaalshaya@sm.imamu.edu.sa, ssalqahtani@imamu.edu.sa, yasalsamel@sm.imamu.edu.sa

***Abstract*— vulnerability reports play an important role in the software maintenance domain. Disclosing vulnerabilities that attackers can exploit depends on the time of mitigating that software vulnerability. The information on vulnerability reports reported by several security scanning software tools facilitates vulnerability management, trends, and secure software development automation. Tagging of vulnerability reports with vulnerability type has thus far been performed manually. Therefore, human-induced errors and scalability issues suffered due to the shortage of security experts. This paper introduces a tool called Vulnerability Report Tagger (VrT), which leverages machine-learning strategies on vulnerability descriptions for automatically labeling NVD vulnerability reports. VrT automatically predicts the cybersecurity labels to assign to vulnerability text to encourage the use of labeling mechanisms in vulnerability reporting systems to facilitate the vulnerability management and prioritization process. Along with the presentation of the tool architecture and usage, we also evaluate our tool effectiveness in performing the vulnerabilities classification (i.e., tagging) process. Link to the tool: https://rb.gy/cz7hwa.**

Keywords: Vulnerabilities; software security; machine learning; fasttext algorithm.

I. INTRODUCTION

In the life cycle of software security, vulnerability monitoring is a crucial task, which implies several activities. First, software components should be kept up-to-date and any potential security issues in terms of performance and correctness should be removed. On the other hand, software developers must invest as little time and effort (e.g., in understanding the relevant security issue [1]) as possible to solve the potential threat in order to keep software maintenance costs low [2].

Vulnerabilities databases are essential for software security engineers to enable rigorous practical software security tasks. In vulnerability databases, security engineers report vulnerability reports or potential software threats, manage them, and keep track of their progress. However, as useful as they might be, many security engineers still end up with a rapidly growing workload and lose control of it [3]. Unlike from more traditional and structured vulnerability reporting systems, such as Mozilla Security[1]. The National Vulnerability Database (NVD), is one of the most popular open-source vulnerability databases, provides an integrated vulnerability reporting system [4]. In this system, vulnerability reporters are only required to provide a short textual abstract, containing a title and short description, to report a new vulnerability to the CVE-ID [2] (Common Vulnerabilities and Exposures Identification.) reserved for that vulnerability on NVD. While this simplified process of reporting vulnerabilities decreases the barrier to entry and attracts more external contributors, it also complicates the work of software security engineers for maintaining the software. This is because several hundreds of vulnerability reports of different natures (e.g., buffer-overflow, DoS, admin privileges) are usually submitted [3]. To cope with these problems, NVD also offers a labeling system (i.e., Common Weakness Enumeration - CWE), which can be used by security engineers and vulnerability reporters to mark and manage vulnerability reports. In particular, CWEs can provide immediate clues about the vulnerability report and are also useful for software project administrators, since they can serve as both a classification and filtering mechanism, thus facilitating the management of the project [5]. However, manually assigning CWEs to vulnerabilities is a labor-intensive and time-consuming task for security engineers [6]. Indeed, although using CWEs has a positive impact on the effectiveness of vulnerability processing, the CWE mechanism is rarely used automatically on NVD [4].

To predict and classify automatically the security labels (i.e, CWEs) of vulnerabilities reports (i.e., CVEs) we decided to use fasttext, classifier that was open sourced by Meta AI Research in 2016 [7]. In contrast to state-of-the-art of the machine learning models [6], [8], and [9], fasttext is a neural network-based approach and different than traditional methods such as bag-of-words in text classification tasks [7]. It considers not only individual words but also subword information, which makes it useful for handling out-of-vocabulary and rare words. For example, in the cybersecurity domain, terms such as "malware," "phishing," and "DDoS" may be considered rare words. However, fasttext's ability to handle these types of terms can be useful in identifying and classifying cyber threats. Therefore, we developed a tool called VrT - A CWE-Based Vulnerability Report Tagger. It leverages machine-learning strategies on vulnerability descriptions for automatically labeling NVD vulnerability reports with CWE. This is particularly important for security engineers who often spend a lot of time handling new vulnerabilities [10], as this approach automatically categorizes vulnerabilities immediately(i.e, *just-in-time*) after they are reported. We evaluate the vulnerability classification effectiveness of VrT by conducting an assessment of the classification performance of the machine learning model implemented in VrT using a dataset of 75,251 vulnerability reports extracted from the NVD database representing the top-25 CWEs (top-25 most dangerous software weakness

[1] https://www.mozilla.org/en-US/security/known-vulnerabilities/firefox/

[2] See https://cve.mitre.org/.

list which demonstrates the currently most common and impactful software weakness[5]).

II. Method

Vulnerability Reports Tagger classifies vulnerability reports by performing the following steps: (i) processing of title and body of the vulnerability report; (ii) classification of vulnerability report; (iii) hierarchical representation of vulnerability report.

A. Preprocessing vulnerability reports

The vulnerability reports pre-processing steps depends on the method selected to perform the automated classification of report. The *fasttext* [7] linear classifier is trained with sentences represented as a bag of words and a bag of n-grams to partially embed information on word order. This because local word order information tends to improve the text classification performance of the linear model. Thus, each security report submitted to the vulnerability database platform is passed to a *fasttext* based classifier. We extract the title and body of the vulnerability report and concatenate them into a single textual paragraph. We then tokenize the resulting text and derive the bag of words representation from the tokenized text. In section IV-B we explained the detailed steps of our preprocessing step.

B. Machine Learning Algorithm

To classify vulnerability reports after pre-processed, we used *fasttext* classifier [7]. As we mentioned in Section I, *fasttext* uses a neural network architecture that can work with both word-level and character-level features of text to predict the probability of text (CVEs texts) belonging to a particular class (CWEs classes). One of the key advantages of *fasttexs* is its ability to work with out-of-vocabulary words, which are words that are not present in the training data. It does this by breaking words down into smaller subword units called n-grams, and using the n-grams as features in the model. This makes it useful in handling rare and domain-specific words, such as technical terms in the cybersecurity domain.

C. Hierarchical Representation of Vulnerability Reports

In each vulnerability report (i.e., an extracted bag of words representation), each word is represented by a vector of character n-gram. This process step represents the desired input of *fasttext*. To classify the vulnerability report through *fasttext*, the main objective is to minimize the following objective function over N possible labels (i.e., CWEs):

$$-\frac{1}{N}\sum_{n=1}^{N} y_n \log\left(f(BAx_n)\right)$$

Where x_n is a bag of features of the $n-th$ vulnerability reports, y_n the label, A represents the weight dictionary of the average text embeddings, B is the weight dictionary that converts the embedding to pre-softmax values for each class, and f is the hierarchical *softmax function* used to minimize computational complexity [7]. For the setting of *fasttext* we used the default values for all other parameters.

III. VrT System: An Example in Use

As reported in Section I, our goal is to support security engineers and analysts to facilitate the vulnerability monitoring and processing activities. We, therefore, integrated the pre-trained *fasttext* model (see subsection IV-B) into a web application, called VrT. This section describes its main functionalities and architecture.

A. Functionalities

To use the tool, users have to access the VrT website. Figure 1 shows the landing page of VrT, shortly describe the available vulnerability tagger and shows the recent accuracy the VrT reached to. The total accuracy section is based on the analysis of all vulnerabilities between 2012 and 2022 (see figure 1). While the first section (i.e., analysis) is shown as soon as the user loads the page, the second section (i.e., results) is initially empty since the user has to first submit a text report to use the automatic cybersecurity labeling of VrT. Once the report is submitted by the user, the corresponding CWE label is rendered and shown.

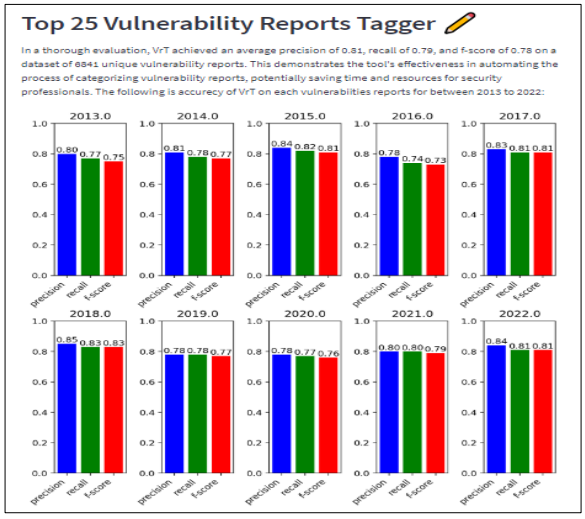

Figure 1: Landing page of VrT, and Vulnerably reports analysis between 2012 and 2022.

Additionally, the predicted label is associated total NVD CVEs for predicted CWEs, average CVSS score, and how it is ranked along with the top 25 CWEs rank. For instance, figure 2 shows a screenshot of the results page for the vulnerability report id CVE-2021-1782.

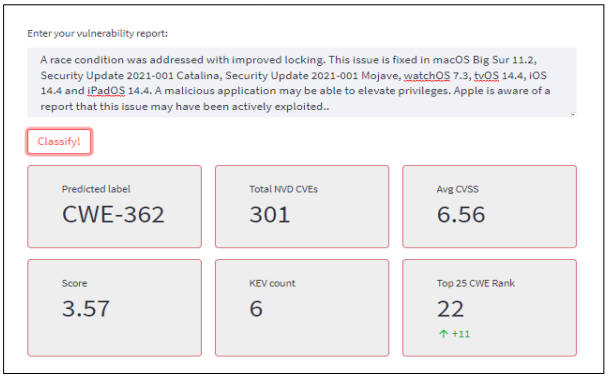

Figure 2: CVE-2021-1782 automatically tagged with CWE-362.

This vulnerability report is about the security content of IOS 14.4 and iPadOS 14.4[3], which the vulnerability discuss race condition. However, as the report submitted on NVD contain the text "*A race condition was addressed with improved locking. This issue is fixed in macOS Big Sur 11.2, Security Update 2021-001 Catalina, Security Update 2021-*

[3] https://support.apple.com/en-us/HT212146

001 Mojave, watchOS 7.3, tvOS 14.4, iOS 14.4 and iPadOS 14.4. A malicious application may be able to elevate privileges. Apple is aware of a report that this issue may have been actively exploited.". Therefore, VrT shows the users input report that it can automatically labeled with CWE-362 - Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition').

B. *Tool Architure*

Figure 3 depicts the main components of the VrT architecture, built using the Streamlit framework (version 1.16.0), and Python 3.7. It is composed of a server-side, which includes the dataset and the web services and a client-side, which includes the Streamlit app widgets. The server-side handles the data preprocessing, running the pre-defined machine learning model, and providing the results to the client-side.

The client-side, using Streamlit, provides an interactive user interface for submitting security reports and receiving results in real-time. The communication between the client and server happens through a RESTful API.

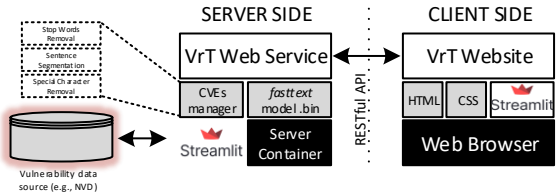

Figure 3: Tool Architecture.

The VrT tool uses the NVD data as the main dataset to perform CWE prediction. The NVD data is a scalable and offline copy that is offered through data feeds[4]. It is available in both data feeds (JSON Vulnerability Feeds and CPE Match Feed) and APIs. However, the JSON Vulnerability Feeds is the one that is used as the source for our experiment. The dataset is loaded into our server and preprocessed to increase the efficiency of the user's requests later on. Our process of extracting security vulnerabilities from the NVD [4] repository is illustrated in the left side of Figure 3.

The CVE descriptions are downloaded in JSON format, as it mentioned above, so some unnecessary text, such as tags, attributes, and declarations, is included. In order to transform these CVE reports into a more informative form (i.e., fasttext format), we extract three sections (title, summary, and CWE-ID) and combine them into a single text file for each CVE report (see Figure 4). Next, we pass the parsed CVEs corpus into the CVE manager component (as it explained in Figure 3). This is a basic text preprocessing approach that uses the `parse_sentence` and `initialize_words` functions imported from the utility library package for text tokenization and word list initialization. The segmented sentences for each CVE report are broken into tokens (preserving the sentence starting and ending tokens) in order to remove unnecessary special characters. As a result of this processing step, the dataset will be as depicted in Figure 4. Each vulnerability dataset for a particular year is formed in one text file; each text file line contains a list of labels, followed by the corresponding vulnerability report (i.e., document). All the labels start with the 'label' prefix, which is how the *fasttext* tool understands what a label is and what a word is.

```
__label__CWE-94 Microsoft Visio Viewer 2010 Gold and SP1 does not properly handle memory
__label__CWE-20 The log_cookie function in mod_log_config.c in the mod_log_config module
__label__CWE-400 MaraDNS before 1.3.07.12 and 1.4.x before 1.4.08 computes hash values fo
__label__CWE-119 Heap-based buffer overflow in the process_tx_desc function in the e1000
```

Figure 4: CVEs reports after transferred into fasttext format.

In short, with Streamlit, once a visitor (e.g., security engineers) submits a security report through the app's user interface, the tool does not recalculate the data required for label prediction from scratch, but instead sends an `XMLHttpRequest` object to the server. On the server-side, we use Streamlit's back-end script to process every HTTP request using our designed CVEs manager component, which preprocesses the user's input and injects the clear text after preprocessing into our ML model (i.e., `fasttext-model.bin`) component. The results of the ML module on the server are then returned back to the Streamlit's front-end and displayed to the user in the form of an interactive widget, showing the predicted label (CWE-ID) and all its associated information such as total NVD CVEs for predicted CWEs, average CVSS score, and how it is ranked along with the top 25 CWEs rank (as shown in Figure 3). The user can also submit new reports and receive updated predictions in real-time using the Streamlit's widgets.

IV. Tool Performance

The measurements used to in our experiment is the vulnerability reports classification accuracy (F-measure). This measure is used for evaluating the classification of security labels extracted from the CVE report. However, in our case, many CVEs have been assigned a classification in the form of Common Weakness Enumeration (CWE) values, which enables us to compare them with the predicted labels suggested by running *fasttext*.

A. *Performance Evaluation*

In general, classifier quality is determined through precision and recall. A high precision value indicates that the classifications are generally correct, while a high recall value indicates that a large number of vulnerability reports will be classified correctly. We use an F-measure (Fig. 5) to evaluate each security label as well as the overall performance of extracting labels from the CVE corpus.

	In fact is *cwe-id*	**Is not *cwe-id***
Classified as *cwe-id*	True Positive (TP)	False Positive (FP)
Classified as not *cwe-id*	False Negative (FN)	Ture Negative (TN)

$$Precision = \frac{\#TP}{\#TP + \#FP} \times 100\% \qquad Recall = \frac{\#TP}{\#TP + \#FN} \times 100\%$$

$$\boldsymbol{F-sore} = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Figure 5: Classifier quality assessment.

B. *Results*

In this sub-section, we evaluate the performance achieved by the *fasttext* model integrated into Vulnerability Report Tagger – VrT. To conduct our evaluation, we collected dataset containing 130,049 and filter the top-25 CWEs (top-25 most dangerous software weakness list which demonstrates the currently most common and impactful software weakness[5]). However, the dataset used in this experiment statistics shown in table 1, 75,243 unique vulnerabilities reports extracted from NVD, and represents 445,466 vulnerable products versions. All CVE reports extracted from 2012 until 2022. We performed a 10-fold cross-validation [11] using the dataset containing the

aforementioned pre-labeled (top-25 CWEs) vulnerability reports.

Table 1: database statistics

Repository	# unique vulnerabilities	# vulnerable products versions	Period
NVD [4]	75,243	445,466	2012 - 2022

More precisely, we used 10-fold cross-validation for each vulnerability dataset promulgated in a particular year. First, we divided the dataset for each vulnerabilities' dataset (CVEs reports) into ten folds. We used eight folds (i.e., 80% of the data) to train the *fasttext* and the remaining two folds to evaluate the classifier's performance. We ran this process ten times for each fold (i.e., 1x10-folds). Finally, to assess the performance of the *fasttext* classifier in detecting CWEs, we computed the well-known evaluation metrics, precision and recall, and their combination F1-score, as explained in subsection V-A, ten times for each fold. Then, to come up with one value for the ten runs, we computed the average of the evaluation metrics for 10-folds ten times (i.e., 1x10-fold) for every vulnerabilities dataset in our testing dataset.

Table 2: VrT performance after a 10-fold cross-validation.

	Precision	**Recall**	**F1-score**
Avg.	**0.81**	**0.79**	**0.78**
Median	**0.81**	**0.80**	**0.79**

As shown in Table 2, the results represent the precision, recall and f-score values of the *fasttext* classifier for the eleven studied vulnerabilities reports (in total 75,243) in our test datasets. First, the precision values obtained by the *fasttext* classifier range between 0.78 and 0.85 with an average of 0.81 (median = 0.80). This suggests that the classifier is making a high number of true positive predictions out of all positive predictions, which is desirable. Second, the recall values range between 0.77 and 0.83 with an average value of 0.79 (median = 0.79). And this suggests that the *fasttext* model of our tool is detecting a high number of true positive instances out of all actual positive instances. Also, the f1-score achieves values between 0.75 and 0.8, with an average of 0.78 and a median of 0.78. Therefore, the classifier is performing well overall, with a balance between precision and recall. The results suggest that the VrT tool classifier is performing well in classifying vulnerabilities reports and the generated model is able to identify the vulnerabilities reports with high precision and recall values. Although the results presented in Table 2 suggest that the VrT tool has good performance, in future work we plan to compare its performance to other state-of-the-art tools in the field of vulnerabilities report classification in order to provide a more comprehensive evaluation of its effectiveness.

V. Conclusion And Future Work

In this paper we presented a tool (Vulnerability Report Tagger -VrT) to help software security engineers and developers to automatically tagging cybersecurity text, which help in vulnerability management and prioritization process. In particular, we presented the tool's functionalities support and architecture. The core of VrT is presented by a machine learning model that analyzes the vulnerabilities reports (title and vulnerability description) in order to determine whether such cybersecurity text can be labeled as one of top-25 CWEs label or not. To evaluate the tool performance, we conducted an experimental evaluation of the model on a dataset compromising 75,243 CVEs report. The results showed that our tool allows to automatically assign CWEs labels with appreciable levels of precision and recall for all the 25 CWEs. The results confirm the practical usefulness of VrT for improving the vulnerabilities reports management practices on vulnerability databases and security scanning platforms. The tool is provided as a web service for analysis of any cybersecurity texts. The tool's accuracy has primarily been tested on the top-25 CWEs, so it may not be as effective in detecting and addressing vulnerabilities outside of those top 25, caution should be taken when using it for less common or emerging vulnerabilities. This limitation will be addressed as part of our future work to improve the tool's accuracy and effectiveness in detecting and addressing vulnerabilities beyond the top-25 CWEs. Additionally, our future plans for the tool include (i) comparing VrT accuracy and functionality with other existing solutions, as well as (ii) investigating its usefulness through the analysis of direct feedback from software security engineers.

References

[1] P. K. Chouhan, F. Yao, and S. Sezer, "Software as a service: Understanding security issues," in *2015 Science and Information Conference (SAI)*, Jul. 2015, pp. 162–170, doi: 10.1109/SAI.2015.7237140.

[2] R. Wang, Y. Zhang, G. Fortino, Q. Guan, J. Liu, and J. Song, "Software Escalation Prediction Based on Deep Learning in the Cognitive Internet of Vehicles," *IEEE Trans. Intell. Transp. Syst.*, pp. 1–11, 2022, doi: 10.1109/TITS.2022.3140903.

[3] A. Zerouali, T. Mens, A. Decan, and C. De Roover, "On the impact of security vulnerabilities in the npm and RubyGems dependency networks," *Empir. Softw. Eng.*, vol. 27, no. 5, p. 107, Sep. 2022, doi: 10.1007/s10664-022-10154-1.

[4] NIST, "National Vulnerability Database," 2007. http://web.nvd.nist.gov/view/vuln/search (accessed Dec. 25, 2022).

[5] MITRE, "2022 CWE Top 25 Most Dangerous Software Weaknesses," 2022. https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html.

[6] X. Sun, L. Li, L. Bo, X. Wu, Y. Wei, and B. Li, "Automatic software vulnerability classification by extracting vulnerability triggers," *J. Softw. Evol. Process*, Sep. 2022, doi: 10.1002/smr.2508.

[7] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of Tricks for Efficient Text Classification," in *Proceedings of the 15th Conference of the {E}uropean Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, Apr. 2017, pp. 427–431, [Online]. Available: https://aclanthology.org/E17-2068.

[8] E. Aghaei, W. Shadid, and E. Al-Shaer, "ThreatZoom: Hierarchical Neural Network for CVEs to CWEs Classification," 2020, pp. 23–41.

[9] V. Yosifova, A. Tasheva, and R. Trifonov, "Predicting Vulnerability Type in Common Vulnerabilities and Exposures (CVE) Database with Machine Learning Classifiers," in *2021 12th National Conference with International Participation (ELECTRONICA)*, May 2021, pp. 1–6, doi: 10.1109/ELECTRONICA52725.2021.9513723.

[10] F. Lomio, E. Iannone, A. De Lucia, F. Palomba, and V. Lenarduzzi, "Just-in-time software vulnerability detection: Are we there yet?," *J. Syst. Softw.*, vol. 188, p. 111283, Jun. 2022, doi: 10.1016/j.jss.2022.111283.

[11] D. Berrar, "Cross-Validation," in *Encyclopedia of Bioinformatics and Computational Biology*, Elsevier, 2019, pp. 542–545.