Learning to Predict Severity of Software Vulnerability Using Only Vulnerability Description

Zhuobing Han*, Xiaohong Li*, Zhenchang Xing†, Hongtao Liu*, and Zhiyong Feng‡
*Tianjin Key Laboratory of Advanced Networking (TANK), School of Computer Science and Technology,
Tianjin University, Tianjin, China, {zhuobinghan, xiaohongli, htliu}@tju.edu.cn
†Research School of Computer Science, Australian National University, Australia, zhenchang.xing@anu.edu.au
‡School of Computer Software, Tianjin University, Tianjin, China, zyfeng@tju.edu.cn

***Abstract*—Software vulnerabilities pose significant security risks to the host computing system. Faced with continuous disclosure of software vulnerabilities, system administrators must prioritize their efforts, triaging the most critical vulnerabilities to address first. Many vulnerability scoring systems have been proposed, but they all require expert knowledge to determine intricate vulnerability metrics. In this paper, we propose a deep learning approach to predict multi-class severity level of software vulnerability using only vulnerability description. Compared with intricate vulnerability metrics, vulnerability description is the "surface level" information about how a vulnerability works. To exploit vulnerability description for predicting vulnerability severity, discriminative features of vulnerability description have to be defined. This is a challenging task due to the diversity of software vulnerabilities and the richness of vulnerability descriptions. Instead of relying on manual feature engineering, our approach uses word embeddings and a one-layer shallow Convolutional Neural Network (CNN) to automatically capture discriminative word and sentence features of vulnerability descriptions for predicting vulnerability severity. We exploit large amounts of vulnerability data from the Common Vulnerabilities and Exposures (CVE) database to train and test our approach.**

***Index Terms*—vulnerability severity prediction; multi-class classification; deep learning; mining software repositories.**

I. Introduction

Software vulnerabilities are exploitable flaws in software systems that pose significant security risks to the host computing system. Software inevitably ships with many vulnerabilities. When vulnerabilities are discovered, software vendors develop patches and mitigations to addresses them. Ideally, software users should update the software with patches before an exploit is developed and targeted at them. However, patching can incur significant unacceptable down-time or unforeseen side effects for complex enterprise systems. Furthermore, organizations rarely have the resources to address every vulnerability that might impact their computing system.

Faced with continuous disclosure of software vulnerabilities, a key question that system administrators have to answer is *which vulnerability they should address first*. The severity of vulnerabilities can inform this decision about which vulnerability gets fixed first in a software system, and how fast it gets fixed. And the vendor is expected to release a timely patch and disclose the vulnerability information along with the patch, thus vulnerability severity classification influences the maintenance of the software and its next release [1]. Therefore, a variety of vulnerability scoring frameworks [2], [3], [4] have been developed to assess severity of software vulnerabilities. Among these frameworks, Common Vulnerability Scoring System (CVSS) [5] by the Forum of Incident Response and Security Teams (FIRST) emerges as the *de facto* standard used to measure the severity of software vulnerabilities. Once discovered, vulnerabilities are documented and disclosed to the public via vulnerability databases such as the Common Vulnerabilities and Exposures (CVE) database [6]. Figure 1 shows an excerpt of a vulnerability report (CVE id = CVE-2017-6798) in the CVE Details [7] database. Vulnerability reports describe (in short text) how each vulnerability works, and they also provide the expert rating of a vulnerability's severity according to the CVSS scoring framework.

CVSS was carefully designed based on expert knowledge. It rates a vulnerability's severity based on a carefully designed formula of several exploitability metrics (how difficult is the vulnerability to exploit) and impact metrics (how significant are the consequences of exploitation). Providing the required exploitability and impact metrics for computing CVSS score can be a daunting task, even for security experts. Compared with the intricate exploitability and impact metrics of a vulnerability, the textual description of the vulnerability is the "surface level" information that can be precisely described. We argue that exploitability and impact metrics, as well as severity score, are all rooted in how each vulnerability works. Therefore, the research problem we aim to tackle in this paper is that "*can we accurately predict severity of software vulnerability using only the "surface level" information (i.e., vulnerability description) of a vulnerability?*"

In particular, we want to exploit publicly available databases of past vulnerabilities (like CVE), and investigate whether we can train a robust machine learning model that can extract discriminative features of vulnerability descriptions and capture intrinsic correlations between these features and vulnerability severity. Before learning can take place, vulnerability descriptions need to be mapped from the original space of symbolic words into some feature space encoding various lexical, syntactic and semantic aspects of the descriptions. This is referred to as feature engineering in machine learning tasks. Feature engineering in general is an empirical process driven by the intuition, experience and domain expertise.

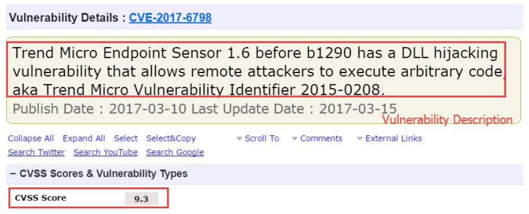
Feature engineering is a challenging task for our vulnerability severity prediction problem because of the diversity of software vulnerabilities and the conciseness of vulnerability descriptions. Software vulnerabilities are diverse in terms of the range of products from which vulnerabilities are discovered, the amount of vulnerability data for different products, and the mechanisms of how vulnerabilities work. Vulnerability descriptions are concise in that they are brief in form (average length is only 37.5 ± 15.4 words) but comprehensive in scope (vocabulary size is 64184). These data characteristics result in a very high-dimensional and sparse feature space if vulnerability descriptions are encoded in one-hot word and document representations (e.g., Term-Frequency/Inverse-Document-Frequency (TF/IDF) [8]). These characteristics also render topic based representations (e.g., Latent Semantic Analysis (LSA) [9], Latent Dirichlet Allocation (LDA) [10]) ineffective due to the difficulty in determining appropriate number of topics a priori.

In this paper, we formulate the problem of predicting severity level of software vulnerability using vulnerability description as a multi-class text classification problem. That is, given a vulnerability description, we want to classify the text into one of the severity levels, for example, critical, high, medium, low of the CVSS framework. We design a one-layer shallow CNN architecture for solving this multi-class text classification problem. The main building block of our architecture is distributional sentence model based on word embeddings and CNN. Our architecture is able to learn to extract and compose the most informative n-grams of vulnerability descriptions when mapping the meaning of individual words in a sentence to a continuous vector of the sentence. One of the greatest advantages of our approach is that it is trained in an end-to-end fashion, thus removing the need for manual feature engineering and greatly reducing the need for adapting to other vulnerability rating systems.

To evaluate our approach, we crawl about 83 thousands of vulnerabilities from CVE Details [7] websites. We design four baseline methods for comparison, using TF/IDF based features, only word-embeddings features, or features extracted by 2-layers CNN or a CNN with Long Short Term Memory (LSTM) layer [11]. Our experiments show that our CNN can be efficiently trained and is robust to input word embeddings learned from even general-text corpus. It can make accurate prediction of vulnerability severity level, with only 3.8% more-than-one-level prediction errors. Furthermore, trained with as minimal as 1000 vulnerabilities, our approach significantly outperforms the classification methods using TF/IDF features or only word embeddings. Compared with more complex deep learning architectures, our shallow CNN performs better on short-text vulnerability descriptions. Finally, trained with past vulnerabilities, our approach can reliably predict the severity of newly discovered vulnerabilities.

This work makes the following contributions:

- We identify a new type of vulnerability analysis, i.e., predicting severity level of software vulnerability using only vulnerability description. To the best of our knowledge, our work is the first attempt to solve this problem.


Vulnerability Details : CVE-2017-6798

Trend Micro Endpoint Sensor 1.6 before b1290 has a DLL hijacking vulnerability that allows remote attackers to execute arbitrary code, aka Trend Micro Vulnerability Identifier 2015-0208.
Publish Date : 2017-03-10 Last Update Date : 2017-03-15

– CVSS Scores & Vulnerability Types

CVSS Score	9.3

Fig. 1. An excerpt of a CVE vulnerability report

- We formulate the prediction problem as a multi-class text classification problem and propose a deep learning approach for predicting a vulnerability's severity level based on only its textual description.
- We conduct large scale experiments with vulnerabilities from the CVE Details database. We investigate the efficiency of model training and the effectiveness and reliability of our approach.

II. Background

This section describes the vulnerability database and scoring framework that our approach relies on, i.e., Common Vulnerabilities and Exposures (CVE) and Common Vulnerability Scoring System (CVSS).

A. *Common Vulnerabilities and Exposures (CVE)*

Common Vulnerabilities and Exposures (CVE) [6] is regarded as a dictionary of CVE Identifiers for publicly known vulnerabilities, which includes three respects: 1) CVE Identifier number (e.g., "CVE-2017-6798" in Fig. 1), 2) standardized description of the security vulnerability or exposure, and 3) some pertinent references (e.g., CWE id).

Since CVE database only provides a reference-method of vulnerabilities, we turn to CVE Details [7] which is a web interface (*www.cvedetails.com*) to the integrated vulnerability data taken from CVE and National Vulnerability Database (NVD) [12]. Through CVE Details, we not only retrieve the detailed information of vulnerabilities (such as CVE identifier number, vulnerability description, expert-rated CVSS score), but also vulnerability statistics for vendors and products.

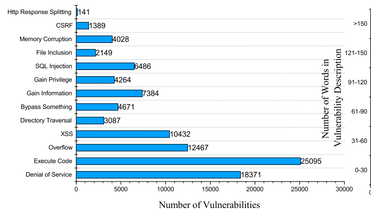
Below are some metrics of our crawled CVE Details data:

Total number of vulnerabilities. As of March 21, 2017, CVE contains 82974 vulnerabilities. These vulnerabilities are discovered in a wide range of 36436 products among the categories of applications, operating systems and hardware.

Vulnerabilities by products. The amount of vulnerability data vary greatly for different products. Among 36436 products, 35267 (96.8%) products contain less than 10 vulnerabilities, 1043 (2.9%) products have the number of vulnerabilities ranged from 11 to 100, and only 126 (0.3%) products have more than 100 vulnerabilities.

Vulnerabilities by type. There are 13 types of vulnerability, such as Denial of Service, Execute Code, Overflow, etc. The number of vulnerabilities by each type is showed in Fig. 2. We can see that various mechanisms have been frequently exploited for thousands of vulnerabilities.

Vulnerability description length and vocabulary size. The statistics of vulnerability description length is presented in



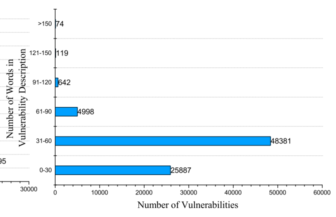


Fig. 2. Number of Vulnerabilities by type

Fig. 3. Distribution of vulnerability description length

Fig. 3. On average, the length of vulnerability descriptions is only 37.5 ± 15.4 words. The vocabulary size of the vulnerability descriptions is 64184. It is obvious that vulnerability description is brief in form but comprehensive in scope.

The above statistics of vulnerability data makes manual feature engineering a challenging problem because of the diversity of software vulnerabilities and the conciseness of vulnerability descriptions.

B. Common Vulnerability Scoring System (CVSS)

CVSS is a quantitative framework for modeling the characteristics and impacts of computer system vulnerabilities. CVSS scores range from 0 to 10, with 0.1 increments and 10 being the most severe, which is available for almost all known vulnerabilities in NVD [12]. According to Atlassian security advisories [13], vulnerabilities are classified into four severity levels based on their CVSS scores.

Critical (CVSS 9.0 - 10.0) Vulnerabilities that score in the "critical" level are usually exploited straightforwardly since the attacker does not need any special authentication credentials or knowledge about individual victims. Exploitation of critical vulnerabilities could allow code execution without user interaction, for example, self-propagating malware (e.g. network worms). Exploiting the vulnerabilities is likely to result in root-level compromise of servers. For this level of vulnerabilities, system administrators or customers are advised to patch or upgrade the vulnerable system immediately.

High (CVSS 7.0 - 8.9) Vulnerabilities that score in the "high" level are usually difficult to exploit. Exploitation of such Vulnerabilities may result in elevated privileges, significant downtime, or compromise of the confidentiality, integrity, or availability of user data and processing resources. For such vulnerabilities, applying important updates at the earliest opportunity is recommended.

Medium (CVSS 4.0 - 6.9) Vulnerabilities that score in the "medium" level typically require the attacker to reside on the same local network as the victim or manipulate individual victims via social engineering tactics. Impact of such vulnerabilities is usually mitigated by factors such as user privileges, authentication requirements. For this kind of vulnerabilities, customers should consider applying the security update.

Low (0.1 - 3.9) Vulnerabilities in the "low" level have very little impact on an enterprise's business. Local or physical system access is always required for exploiting such vulnerabilities, and customers may evaluate whether to apply the security update to the affected systems.

III. The Approach

This section formulates our research problem, presents an overview of our approach and describes the technical details.

A. Problem Formulation

The goal of this work is to design an effective approach that allows computers to "understand" natural language text in order to classify the severity level of software vulnerability according to its description. Specially, we formulate our task as a multi-class text classification problem. Given the textual description of a vulnerability, our goal is to predict the severity level of this vulnerability among the four CVSS levels, i.e., "Critical", "High", "Medium", and "Low".

B. Approach Overview

Fig. 4 presents the overview of our approach. First, we crawl vulnerability descriptions and expert-rated CVSS scores for all vulnerabilities published in the CVE Details website. We map the expert rated CVSS scores to the corresponding severity levels as described in Section II-B. Our dataset contains entries in the form of $<$*vulnerability description, expert-rated CVSS severity level*$>$ for each crawled vulnerability.

As we formulate the severity prediction problem as a text classification problem, we first need to represent words in vulnerability description as vectors. In this work, we use word embeddings [14] to represent words in a vector space, because many studies [15], [16], [17] have shown that word embeddings can capture rich syntactic and semantic features of words in a low-dimensional vector. We train word embeddings with a corpus of crawled vulnerability descriptions using continuous skip-gram model [18] (the Python implementation in Gensim [19]) The output of word embedding training is a dictionary of word vectors for each word in the vocabulary of crawled vulnerability descriptions.

Word embedding captures word-level features, while we also need to capture sentence level features to predict a vulnerability's severity level. Because vulnerability descriptions usually involve 2-4 concise and informative sentences, we design a one-layer shallow CNN [20] to capture sentence-level features in vulnerability descriptions. Given a vulnerability description, vectors of words in the description sentences are first looked up in the word embeddings dictionary and then are concatenated into a vulnerability description vector. The vulnerability description vector is fed into the CNN to predict the security level of the vulnerability. To train the CNN for extracting discriminative word and sentence feature vectors from vulnerability descriptions, we enter a large number of the crawled vulnerability descriptions and the corresponding expert rated CVSS severity level to the CNN. The training is guided by hinge loss of the severity level predicted by a SVM classifier against the experted rated severity level.

C. Representing Vulnerability Descriptions by Word Embeddings

The first problem in our task formulation is how to represent vulnerability descriptions as input to the text classification model. Since words in vulnerability descriptions are discrete

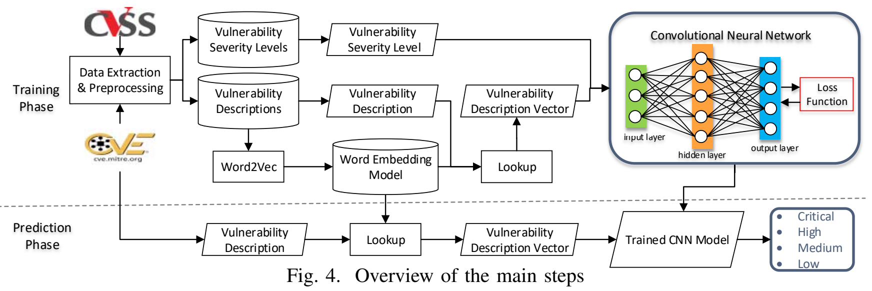

CVSS
Training Phase
Data Extraction & Preprocessing
cve.mitre.org
Vulnerability Severity Levels
Vulnerability Descriptions
Vulnerability Severity Level
Vulnerability Description
Vulnerability Description Vector
Word2Vec
Word Embedding Model
Lookup
Convolutional Neural Network
input layer
hidden layer
output layer
Loss Function
Prediction Phase
Vulnerability Description
Lookup
Vulnerability Description Vector
Trained CNN Model
Critical
High
Medium
Low

Fig. 4. Overview of the main steps

symbols which cannot be fed directly to a model, we need to represent words into a vector space. Considering the large vocabulary size of vulnerability descriptions, we decide not to use one-hot word vector [21], because the dimensionality of one-hot vector is too large to use the vector as features for words in a supervised model [22]. Inspired by the recent advance of word embeddings in Natural Language Processing (NLP) tasks [14], [23], [24], we decide to adopt word embeddings to represent words in vulnerability descriptions.

In this paper, we use continuous skip-gram model, one of the popular word embedding models proposed by Mikolov et al. [14], [18]. The basic idea of continuous skip-gram model is to learn dense low-dimensional word vectors that are good at predicting the surrounding words with a center word. The objective function of the model is to maximize the sum of log probabilities of the surrounding context words w_{i+j} conditioned on the center word w_i, i.e., $\sum_{i=1}^{n} \sum_{-k \leq j \leq k, j \neq 0} \log p\left(w_{i+j} | w_i\right)$ where n denotes the length of the word sequence and $2k+1$ is the size of context window.

Learning word embeddings needs only large amounts of unlabeled text. As our task deals with vulnerability specific text, we prepare a large corpus of crawled vulnerability descriptions for learning vulnerability specific word embeddings.The output of the model is a dictionary of words. Each word is associated with a vector representation. A vulnerability description is then represented as a description vector by looking up the dictionary of word embeddings and concatenating the word embeddings of words comprising the vulnerability description. The resulting description vector is used as the input of the CNN. The trained word embeddings may not cover all words in vulnerability descriptions. We follow the common practice to deal with such out-of-vocabulary words [20], [25], i.e., randomly initialize the corresponding word vectors.

D. Predicting Vulnerability Severity Level by CNN

To predict the severity level of a vulnerability based on its description, discriminative word and sentence features of vulnerability descriptions should be designed. However, the diversity of software vulnerabilities makes manual feature engineering a challenging problem. In this work, we design a shallow CNN architecture which takes as input a vulnerability vector and automatically extract the most informative word and sentence features in the vulnerability description for predicting a vulnerability's severity level. The architecture is presented in Fig. 5. Our CNN model contains an input (embedding) layer, a convolution and max-pooling layer, a fully connected layer and an output SVM layer.

In the input layer, a vulnerability description is tokenized and converted to a sequence of word vectors, i.e., vulnerability description vector. In general, let $x_i \in \mathbb{R}^k$ denotes the k-dimensional word vector corresponding to the ith word in the description. A vulnerability description of length n is represented as $x_{1:n} = x_1 \oplus x_2 \oplus \ldots \oplus x_n$ where $\oplus$ is the concatenation operator.

In the convolution and max-pooling layer, we have *filters* of S different window sizes (h) in order to extract features of phrases of different length from the input sentence. In Fig. 5, we depict *filters* of three ($S = 3$) window sizes ($h = 1, 3, 5$) for 1-gram, 3-gram and 5-gram, respectively. For each filter window size, we use N (e.g., 128) filters to learn complementary features from the same word windows. Since each word is represented as a k-dimensional vector, the width of filters is set to the dimensionality of the word vectors in NLP tasks. Then a convolution operation with a *filter* $w \in \mathbb{R}^{hk}$ is applied to a window of h words to generate a feature c_i. The convolution operation is applied repeatedly to each possible window of h words (zero padding where necessary) in the vulnerability description (i.e., $1 \leq i \leq n-h+1$). After convolution, the nonlinear activation function ReLu [26] is applied to generate a feature map.

Specifically, a feature c_i is produced by a window of h words $x_{i:i+h-1}$ and the filter w as $c_i = f\left(w \cdot x_{i:i+h-1} + b\right)$ where $\cdot$ is the dot product between the filter vector and the word sequence vector, $b \in \mathbb{R}$ is a h-dimensional bias vector and f is the activation function $ReLu(z) = max(0, z)$.

The dimensionality of each feature map is a function of the length of vulnerability description and the window size (h) of the filter. Thus, a pooling function should be applied to each feature map to induce a fixed-length vector. In this work, a 1-max-pooling operation [20], [23] is performed over each feature map and the maximum value is selected as the most informative feature corresponding to a particular filter. For N filters, the length of vector induced after 1-max-pooling is N. Thus, the convolution and max-pooling layer produces in total $S \times N$ feature vectors from $S \times N$ filters, and these feature vectors are concatenated to form a feature vector in a fully connected layer. The fully-connected feature vector would capture features of the most informative n-grams of different length in the vulnerability description.

The output SVM layer then receives as input the fully-connected feature vector and outputs the probability distribution over the four severity levels. In the training phase, hinge loss between the output probability distribution over the four severity levels and the expert rated severity level is used to guide the training.In the prediction phase, the severity label with the highest probability is considered as the severity level of the input vulnerability.

E. Training the CNN to Learning Word and Sentence Features

The proposed CNN is trained with sufficient samples of four severity levels of vulnerability descriptions, so that the CNN can capture discriminative word and sentence features for the severity prediction task.

Given the feature vector of a vulnerability description, before passing it to the SVM classifier, we add a dropout layer [27] during training to regularize the CNN by stochastically disable a fraction of its neurons. The basic idea is to avoid neurons from co-adapting and force them to learn individually useful features [28]. The fraction of neurons we keep is set to 0.5 during training, and to 1.0 (disable dropout layer) during prediction. Then we pass the feature vector to the SVM classifier which uses the hinge loss as the loss function. The training objective is to minimize the loss function in order to measure the error made by our CNN. The hinge loss for the ith training vulnerability description is calculated as $L_i = \sum_{j \neq y_i} max\,(0, s_j - s_{y_i} + 1)$, where s_j represents the score for the jth severity level, and y_i specifies the true severity level for the training sample.

Next, we apply the Adam update algorithm [29] to optimize our CNNs loss function. In order to prevent our model from overfitting, we use the $l2$ norm regularization which penalizes large weight parameters such as the the convolution filters w and the bias terms b. Finally, the SVM classifier gives the normalized class scores as output by which the model can be used to predict the severity level of software vulnerability given the vulnerability description.

IV. Experiment

We conduct several experiments to study the performance of the proposed CNN architecture and make comparison with four baseline methods. We also report the hyperparameters optimization of the word embedding and CNN models and the training cost of our approach. All experiments run with TensorFlow on an NVIDIA Tesla M40 GPU.

A. Vulnerability Dataset

Our experimental dataset is based on the vulnerability data crawled from the CVE Details website [7]. Section II-A summarizes the basic information of this dataset. The statistics of each severity level of the crawled vulnerabilities is summarized in Table I. We can see that the distribution of vulnerabilities at different severity levels is imbalanced. The Low severity level has the least number of vulnerabilities, i.e., 6427. Furthermore, we discard 494 low-severity vulnerabilities with CVSS score 0 and with incomplete vulnerability descriptions. Therefore, we have 5933 low-severity vulnerabilities in our dataset. To prepare a balanced training and testing dataset, we adopt an undersampling strategy. Specially, we select all low-severity vulnerabilities and randomly select 5933 vulnerabilities for each of the other three severity levels, i.e., in total $5933 \times 4 = 23732$ vulnerabilities for our experiments. We use 80% of the vulnerabilities of each severity level as the training data, 10% as the validation data for optimizing the hyperparameters of the word embedding and CNN models, and 10% as the testing data for evaluating and comparing the performance of our approach and the baseline methods.

TABLE I
SUMMARY OF EACH SEVERITY LEVEL OF VULNERABILITIES

Severity Level	CVSS Score Range	Number	Percentage
Critical	9.0-10.0	12341	14.8%
High	7.0-8.9	20972	25.2%
Medium	4.0-6.9	43504	52.3%
Low	0.0-3.9	6427	7.7%

B. Baseline Building

We design four baseline text classification methods. All baseline methods use a SVM multiclass classifier with Radial Basis Function (RBF) kernel [30] (the Python implementation in scikit-learn) for prediction, but with different text features: 1) TF/IDF based features, 2) word embeddings based features, 3) features extracted by a 2-layer CNN, and 4) features extracted by a CNN with Long-Short-Term-Memory (LSTM) RNN. The Baseline3 and Baseline4 are supposed to capture longer dependencies of words in vulnerability descriptions using more complex deep learning architectures [31], [32]. We use the same training dataset to train the baseline classifiers as that for training our CNN. We adopt the same hyperparameters optimization process as described in Section IV-E to optimize the hyperparameters of the CNN and LSTM models in the Baseline3 and Baseline4.

1) TF/IDF based SVM (TF/IDF + SVM): The Baseline1 is designed to extract features by TF/IDF [33]. TF/IDF is a statistical measure to evaluate the importance of a word to a document in a corpus. As we have about 21 thousands vulnerability descriptions in the training dataset and more than 25 thousands words in the vocabulary (preprocessed by stop-word removal and tokenization), the TF/IDF matrix becomes extremely huge and sparse. Thus we adopt the scikit-learn HashingVectorizer API[1] for dimensionality reduction. Specifically, HashingVectorizer converts a collection of text files from a matrix holding token occurrence counts to a fixed dimensional feature space (300-dimension in this experiment) by using the hashing trick. And then, IDF weighting is calculated by the scikit-learn TfidfTransformer API. Finally, vulnerability descriptions are represented as 300-dimensional TF/IDF vectors for the SVM classifier training.

2) Word embeddings based SVM (Word Embedding + SVM): The Baseline2 is designed to extract feature by word embeddings. Each vulnerability description in our dataset can be represented by taking the mean of the word embeddings

[1]http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.HashingVectorizer.html

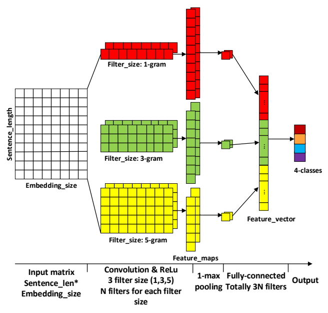

Filter_size: 1-gram
Sentence_length
Embedding_size
Filter_size: 3-gram
Filter_size: 5-gram
Feature_maps
Feature_vector
4-classes
Input matrix Sentence_len* Embedding_size
Convolution & ReLu 3 filter size (1,3,5) N filters for each filter size
1-max pooling
Fully-connected Totally 3N filters
Output

Fig. 5. Architecture of our shallow CNN

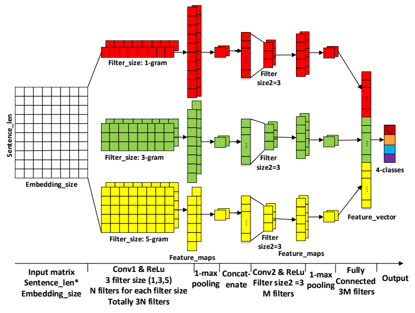


Fig. 6. Architecture of the 2-layer CNN

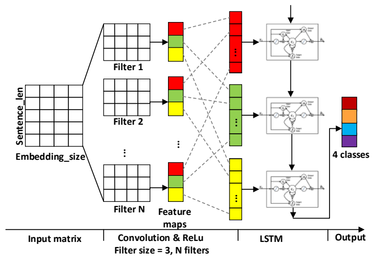


Fig. 7. Architecture of the C-LSTM

of the words comprising the description [17], [34]. Compared with our CNN that learns to extract the most informative n-grams for the prediction task, this baseline treats each word in the vulnerability description equally. In this baseline, we also use 300-dimensional word vector. Therefore, the vulnerability description vector is a 300-dimensional vector as the input to the SVM classifier.

3) CNN with Two Hidden Layers (Word Embedding + 2-Layer CNN): The Baseline3 is set by using word embeddings and a 2-layer CNN (see Fig. 6) which contains two convolution and 1-max-pooling layers. The first hidden layer of the Baseline3 uses the same setting as our CNN. After the convolution, ReLu activation and 1-max pooling, the output of first layer is three N-dimensional feature vectors which are passed to the second hidden layer. For each input N-dimensional feature vector, the second layer uses M = 64 filters with window size h = 3. After the convolution, ReLu activation and 1-max pooling, the second layer outputs three M-dimensional feature vectors which are concatenated into a feature vector in the fully connected layer. Then the feature vector is passed to a SVM classifier for training and prediction.

4) CNN with LSTM RNN (Word Embedding + CNN with LTSM): The Baseline4 is set by using word embeddings and a C-LSTM architecture (see Fig. 7) [11]. The C-LSTM model consists of a CNN layer and a LSTM layer. The first layer uses a CNN with filter window size h = 3. We do not use different sizes of filters because the findings of [11] show that a convolutional layer with single filter size has a better performance in C-LSTM model. The number of filters is N = 128 which is the same as our approach. After the convolution and ReLu activation, the first layer generates N feature maps. Features from the same word windows in these N feature maps are concatenated as $n - h + 1$ (n being the length of vulnerability description) feature vectors each of which has a length of N as shown in Fig. 7. The second layer is the standard LSTM architecture proposed by [35]. In the setting of this baseline, the LSTM architecture contains a hidden unit for each feature vector. The final output of the C-LSTM model is the last hidden unit, which is used as the input to the classifier.

C. Word Embedding Training Corpus

To study the impact of word embeddings learned from different corpus, we prepare three different corpus:

- *CVE vulnerability description corpus* is considered as a vulnerability-specific corpus which contains all the crawled vulnerability descriptions from CVE Details. This corpus has the vocabulary size of 64,184, and the corpus contain 3,003,787 tokens. We learn 50, 150, and 300-dimensional word embeddings on this corpus.
- *Stack Overflow corpus* is regarded as a computer science specific corpus. We prepare the corpus with the title and body of all posts from the Stack Overflow Data Dump published on December 15, 2016, which contains 9,646,800,250 tokens. The vocabulary size is 800,818. We learn 300-dimensional word embeddings on it.
- *GoogleNews-vectors* is a general text corpus directly obtained from the official word2vec website[2]. This corpus contains the word2vec pre-trained three million 300-dimensional English word vectors from Google News.

D. Evaluation Metrics

The multi-class classification results are usually represented as a $N \times N$ confusion matrix M [36], where N denotes the number of classes. Each column of the matrix represents the predicted classes while each row represents the actual classes. The value of each cell M_{ij} is the number of vulnerabilities with the expert-rated severity level L_i that is predicted as the severity level L_j. In this work, we have four severity levels to predict, i.e., $N = 4$. Expert rated severity levels are the actual class, and the predicted severity levels are the predicted class.

We use Accuracy Precision, Recall and F1-measure that are commonly used to evaluate the effectiveness of prediction problems in the literature [37], [38], [39], [40]. **Accuracy** is defined as correct predictions among the total number of vulnerabilities in the dataset. **Precision** for a severity level L_j represents the proportion of the vulnerabilities correctly classified as L_j among all vulnerabilities predicted as L_j. **Recall** for a severity level L_i is the proportion of vulnerabilities correctly classified as L_i compared with the number of vulnerabilities with expert-rated L_i. **F-measure** for a severity level i is the harmonic mean of recall and precision evenly for that severity level. Since F1-measure takes both precision and recall into consideration, we choose F1-measure as the main evaluation metric. An **overall** metric is the mean of the corresponding metric for the four severity levels.

[2]https://code.google.com/archive/p/word2vec/

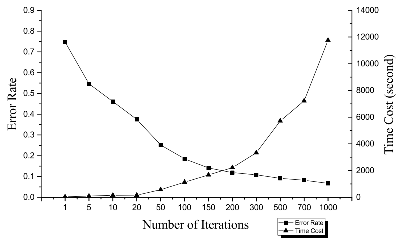


Fig. 8. Average error rate and time cost for different numbers of iterations

E. Deep Learning Hyperparameters Optimization

In this section, we report our experiments to optimize the important hyperparameters of the word embedding model and the CNN model of our approach using the validation dataset. Following the treatment of the sensitive analysis of CNN parameters [25], we fix the other hyperparameters in the experiment of one hyperparameter.

1) The Number of Training Iterations: The number of Training iterations has great impact on the performance of CNN. The training objective is to gradually reduce the error rate, i.e., the loss by the classifier, during each iteration. Generally, the larger the number of iterations, the lower the error rate, but the higher the time cost. In order to balance the error rate and the time cost, we conduct experiments with 12 values for the number of iterations ranged from 1 to 1000 (see Fig. 8). According to our experiment results, we set the number of training iterations to 150, with which the average error rate is about 0.118 and the time cost is about 2000 seconds.

2) The Batch Size for Each Training Step: The batch size is the number of vulnerability descriptions used for training for each step within an iteration. In order to find a suitable batch size, we conduct experiments with three batch size values, i.e., 32, 64, 128. Based on the experimental results shown in the row of "batch_size" in Table II, we set the batch size to 32 which obtains the highest overall accuracy, precision, recall and F1-measure on the validation dataset.

3) The Dimensionality of Output Feature Vectors of CNN: The dimensionality of output feature vectors of CNN affects the complexity and feature extraction capability of the CNN. In our CNN, the dimensionality of output feature vectors for CNN is the number of filters N for each filter window size. We conduct experiments with three different number of filters, i.e., N = 32, 64, or 128. According to the experimental results shown in the row of "num_filters" in Table II, we set the dimensionality of output feature vectors of CNN to 128. That is, we use 128 filters for each filter window size in the convolutional layer.

4) The Dimensionality of Word Embeddings: Word embeddings are essentially feature extractors that encode semantic and syntactic properties of words in their dimensions. We conduct experiments with three word-embedding dimensionality value (i.e., 50, 150, 300) on the vulnerability description corpus. According to the experimental results shown in the row of "embedding dimension" in Table II, we use 300-dimensional word embeddings in the evaluation of our approach.

TABLE II
HYPERPARAMETERS OPTIMIZATION

Hyperparameters	Value	Accuracy	Overall Precision	Overall Recall	Overall F1-Measure
	32	**0.816**	**0.818**	**0.815**	**0.816**
batch_size	64	0.815	0.814	0.813	0.812
	128	0.813	0.813	0.810	0.810
	32	0.803	0.805	0.803	0.800
num_filters	64	0.813	0.813	0.811	0.810
	128	**0.816**	**0.818**	**0.815**	**0.816**
embedding dimension	50	0.761	0.759	0.758	0.759
	150	0.780	0.778	0.777	0.776
	300	**0.816**	**0.818**	**0.815**	**0.816**

F. Research Questions and Findings

We conduct a set of experiments to answer the following research questions (RQs):

RQ1: Do our approach outperforms the baseline methods for vulnerability severity prediction?

Motivation. Our approach and the four baseline methods differ in the features used and the ways of feature extraction. We want to investigate whether and to what extent our approach can improve the results of vulnerability severity prediction.

Approach. We apply our approach and the baseline methods to the same test dataset, and compare the accuracy, overall precision, recall, and F1-measure metrics of different approaches.

Result. From the results in the last column of Table III, our CNN achieves the best performance in all the overall metrics, compared with the four baseline methods. We can see that the overall F1-measure of our approach outperforms the Baseline1 to Baseline4 by 26.9%, 23.6% , 4.3% and 4.5%, respectively. Similar improvements on accuracy (see Table IV) and overall precision and recall can be observed.

Our approach significantly outperforms the classification methods using traditional text features or only word embedding features. Furthermore, it is not always the case that the more complex a deep learning architecture, the more effective the architecture. For short-text vulnerability descriptions, the longer sentence-level features captured by the complex CNN and LSTM architectures are not effective for vulnerability severity prediction.

RQ2: How effective is our approach in predicting different levels of vulnerability severity, compared with the baseline methods?

Motivation. Our task is a multi-class text classification problem. In addition to the overall performance of the four severity levels as a whole, we would like to investigate the effectiveness of our approach for each level of vulnerability severity.

Approach. We construct the confusion matrix of the prediction results of each method, and compare the precision, recall and F1-measure metrics for each level of vulnerability severity by our approach and the four baseline methods.

Result. Table III presents the precision, recall and F1-measure of different methods for each severity level. We can see that our approach performs the best in all the metrics for all severity levels, except the recall for the critical level where the Baseline4 is 0.008 higher than our approach. The standard deviation of a metric across different severity levels is shown

TABLE III
PRECISION, RECALL AND F1-MEASURE OF OUR APPROACH AND THE BASELINE METHODS

			Critical	High	Medium	Low	Mean ± StdDev
Precision	Baseline1	TFIDF + SVM	0.626	0.480	0.441	0.636	0.546 ± 0.086
	Baseline2	word embedding + SVM	0.709	0.481	0.512	0.659	0.590 ± 0.096
	Baseline3	word embedding + 2-layer CNN	0.787	0.712	0.740	0.855	0.774 ± 0.054
	Baseline4	word embedding + CNN with LSTM	0.811	0.678	0.706	0.896	0.773 ± 0.087
	Our Approach	word embedding + 1-layer CNN	**0.854**	**0.728**	**0.785**	**0.904**	**0.818 ± 0.067**
Recall	Baseline1	TFIDF + SVM	0.624	0.470	0.411	0.690	0.549 ± 0.113
	Baseline2	word embedding + SVM	0.599	0.642	0.410	0.674	0.581 ± 0.102
	Baseline3	word embedding + 2-layer CNN	0.843	0.707	0.701	0.842	0.773 ± 0.069
	Baseline4	word embedding + CNN with LSTM	**0.856**	0.713	0.691	0.820	0.770 ± 0.070
	Our Approach	word embedding + 1-layer CNN	0.848	**0.793**	**0.759**	**0.861**	**0.815 ± 0.041**
F1-Measure	Baseline1	TFIDF + SVM	0.625	0.475	0.426	0.662	0.547 ± 0.099
	Baseline2	word embedding + SVM	0.649	0.550	0.455	0.667	0.580 ± 0.085
	Baseline3	word embedding + 2-layer CNN	0.814	0.710	0.720	0.849	0.773 ± 0.060
	Baseline4	word embedding + CNN with LSTM	0.833	0.695	0.698	0.856	0.771 ± 0.074
	Our Approach	word embedding + 1-layer CNN	**0.851**	**0.759**	**0.771**	**0.882**	**0.816 ± 0.052**

in the Mean ± StdDev column of Table III, from which we can see that the standard deviation of our approach is the smallest (0.052 in terms of F1-measure), compared with that of the four baseline methods for different severity levels.

Our approach is more effective in predict different levels of vulnerability severity and performs more consistently, compared with the baseline methods.

RQ3: How much training data is required to train a robust prediction model in our approach?

Motivation. Our CNN is trained in a supervised way. This RQ is set to explore the impact of training data size on the model performance and the practicality of our approach.

Approach. We randomly split our training dataset into equal-size subsets of 1000 vulnerabilities. Training is performed from one subset to all the 18 subsets with one-subset increment. We use the same testing data for evaluation.

Result. Fig. 9 shows the overall F1-measure of our CNN with different sizes of training data. With as minimal as 1000 vulnerabilities, the F1-measure of our approach already outperforms the F1-measure of the Baseline1 and Baseline2 by 15.8% and 12.5%, respectively. We see staged increase of F1-measure with the increase of training data, i.e., from 1000 to 2000, 4000 to 5000, 7000 to 1000, and 16000 to 18000. In between the two increase stages, the F1-scores remains stable with respect to the increasing training data. This implies that increasing training data improves the model performance in general. Increasing training data has larger impact on smaller training dataset, and the performance improvement become smaller and slower as the training dataset becomes larger. This is because more and more unseen data is required to expand the model, otherwise the model will remain stable after a performance improvement.

Our approach is practical as it needs only a very small training data (e.g., 1000 vulnerabilities) to significantly outperforms the Baseline1 and Baseline2. A reasonably-sized training dataset (e.g., 7000 to 10,000 vulnerabilities, about one-year accumulation of vulnerability data) can train a robust and stable prediction model.

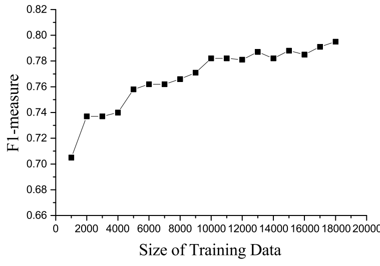


Fig. 9. Impact of training data size on model performance

RQ4: How reliable is a trained CNN model in face of ongoing disclosure of new vulnerabilities?

Motivation. As vulnerabilities are ongoing discovered (about 6000+ vulnerabilities per year in 2015 and 2016), we would like to investigate whether our model trained with past vulnerabilities is reliable for predicting the severity of newly discovered vulnerabilities, and if needed, how frequent the model should be retrained with the new vulnerability data.

Approach. We prepare a before-2016 training dataset with 10000 (2500 for each severity level) vulnerabilities discovered before Dec 31, 2015. We choose 10000 vulnerabilities as the experiment results in RQ3 show that 10000-vulnerabilities training data reaches a good and stable prediction performance, and also considering the number of available low-level vulnerabilities. We prepare a before-2016 test dataset with 2000 (500 for each severity level) vulnerabilities. We prepare three 2016 test datasets with vulnerabilities (600 for each dataset, 150 for each severity level) as shown in the first column of Table V. These three test datasets simulate ongoing discovered vulnerabilities after model training.

Result. As shown in Table V, the model trained with before-2016 vulnerabilities has relatively better performance on the before-2016 test data (0.02 higher in F1-measure) than on the three 2016 test datasets. The model performance remains stable for the vulnerabilities discovered in the first six months of 2016, but has a performance drop (0.02 drop in F1-measure) for the vulnerabilities discovered afterwards. In six months, there are about 3000+ newly discovered vulnerabilities. The

TABLE IV
ACCURACY OF OUR APPROACH AND THE BASELINE METHODS

		Accuracy
Baseline1	TFIDF + SVM	0.552
Baseline2	word embedding + SVM	0.583
Baseline3	word embedding + 2-layer CNN	0.775
Baseline4	word embedding + CNN with LSTM	0.772
Our Approach	word embedding + 1-layer CNN	**0.816**

TABLE V
PERFORMANCE ON NEW VULNERABILITIES AFTER MODEL TRAINING

Testing Dataset	Accuracy	Overall Precision	Overall Recall	Overall F1-Measure
Before-2016	0.783	0.790	0.782	0.779
Jan 1-Mar 30, 2016	0.760	0.774	0.763	0.757
Apr 1-Jun 30, 2016	0.761	0.770	0.759	0.754
Jul 1-Sept 30,2016	0.739	0.747	0.737	0.734

TABLE VI
IMPACT OF WORD EMBEDDINGS LEARNED FROM DIFFERENT CORPUS

Corpus	Accuracy	Overall Precision	Overall Recall	Overall F1-Measure
Google News	0.803	0.807	0.802	0.801
StackOverflow Posts	0.815	0.815	0.814	0.814
Vuln. Description	**0.816**	**0.818**	**0.815**	**0.816**

model needs to be expanded to accommodate new features in these vulnerabilities. This is in sync with our experiment results in RQ3 which shows that increasing training data by 2000-4000 vulnerabilities can improve model performance.

The model performance may deteriorate on newly discovered vulnerabilities, but the deterioration is minor and slow. The model only needs to be retrained every 6 months.

RQ5: How sensitive is our approach to input word embeddings learned from different corpus?
Motivation. The corpus used to learn word embeddings may affect word-level features captured in word embeddings, and subsequently affect the performance of the CNN model. This RQ is set to investigate whether and to what extent the performance of our approach is affected by different word-embedding training corpus.
Approach. We learn 300-dimensional word embeddings separately from the three different word-embedding training corpus (See Section IV-C) and use the resulting word embeddings to represent vulnerability descriptions as input to our CNN.
Result. As shown in Table VI, word embeddings from Google News corpus slightly degrade the performance of our approach, compared with the two domain-specific word embeddings, in terms of accuracy, precision, recall and F1-measure by 1.3%, 1.1%, 1.3% and 1.5%, respectively. Although the performance of our approach degrades with general word embeddings, it still outperforms all the four baseline methods.

Our CNN is robust with respect to the input word embeddings learned from a general-text corpus. However, domain-specific word embeddings leads to better performance.

G. Discussion

This section presents the qualitative analysis of some examples to illustrate the capability of our approach and discusses threads to validity of our experiments.

1) Qualitative Analysis: The confusion matrix of our approach is shown in Table VIII. We consider prediction errors up or down one level of the actual severity level as *moderate errors* (highlighted in blue), errors up or down two levels as *significant errors* (highlighted in orange), and errors up or down three levels as *critical errors* (highlighted in red). Among the 2374 vulnerabilities in the test dataset, our approach makes 436 (18.4%) prediction errors. Among these 436 prediction errors, 346 (79.4%) are moderate errors, 85 (19.5%) are significant errors, and only 5 (1.1%) are critical errors (i.e., 5 low-level vulnerabilities are predicted as critical).

We consider up or down one level errors as moderate, because CVSS scores are progressive instead of sharp contrast. For example, for the vulnerability (CVE-2015-7752) in Table VII, the expert rated CVSS score is 7.8 (i.e., high severity), but this score is only 0.9 higher than the upper bound (6.9) of the predicted medium level. Therefore, we focus our analysis on significant and critical errors, which accounts for 20% of all prediction errors and 3.8% of all vulnerabilities in the test dataset. Table VII shows examples of these errors for each severity level (highlighted in blue). It also show examples of correct predictions as contrast. We highlight our CNN model's attention on words and phrases in vulnerability descriptions in bold font. We can see that our CNN model indeed captures discriminative words and phrases in vulnerability descriptions. But there are three common situations which may confuse the model and lead to prediction errors:

- Vulnerability descriptions which contain more attention words and phrases (e.g., CVE-2012-3128, CVE-2015-1123) tend to be classified as a relatively higher severity.
- Vulnerability descriptions with lots of digits, product names and file directories etc. (e.g., CVE-2015-7752) may lead to error classification since the sentence tokenizer we use is not reliable to tokenize these words which often leads to noise in the description vectors, and subsequently affects prediction accuracy.
- Vulnerability descriptions with very short length (e.g., CVE-2016-1000112) have very limited information to make reliable severity prediction.

2) Threats to validity: Threats to internal validity relate to errors in our experimental data and tool implementation. We have double checked our experimental data and tool implementation. We have also manually checked the selected vulnerabilities in our dataset to ensure that they have the right vulnerability severity levels. Threats to external validity relate to the generalizability of our results. In this study, we use a medium-size training and test dataset. This allows us to perform manual analysis to understand the capability and limitations of our approach. In the future, we will reduce this threat by extending our approach to more vulnerabilities for training and testing and adapting it to other vulnerability databases and scoring systems (e.g., Microsoft Security Bulletin [2]). The adoption of deep learning techniques will greatly reduce the efforts for such extension and adaptation.

TABLE VII
EXAMPLES OF DIFFERENT LEVELS OF VULNERABILITY SEVERITY AND PREDICTION RESULTS

Expert-Rated Level	CVE ID	Vulnerability Description	Predicted Level
Critical	CVE-2017-2937	Adobe Flash Player versions 24.0.0.186 and earlier have an **exploitable use** after free vulnerability in the ActionScript FileReference class, when **using class inheritance**. Successful **exploitation** could lead to **arbitrary code execution**.	Critical
	CVE-2016-1000112	**Unauthenticated remote** .jpg **file** upload in contus-video-comments v1.0 wordpress plugin.	Medium
High	CVE-2016-8704	An **integer overflow** in the process_bin_append_prepend function in Memcached, which is responsible for **processing multiple commands** of Memcached **binary protocol**, can be **abused** to cause heap **overflow** and lead to **remote code execution**.	High
	CVE-2015-7752	The SSH server in Juniper Junos OS before 12.1X44-D50, 12.1X46 before 12.1X46-D35, ...(11 more xxx before yyy)..., and 15.1X49 before 15.1X49-D20 allows **remote attackers** to cause a **denial of service (CPU consumption)** via unspecified SSH traffic.	Medium
Medium	CVE-2016-8402	An **information disclosure** vulnerability in **kernel components** including the ION subsystem, ... could enable a **local malicious** application to **access data** outside of its **permission** levels. Product: Android. Versions: Kernel-3.10, Kernel-3.18. Android ID: A-31495231.	Medium
	CVE-2015-1123	WebKit, as used in Apple iOS before 8.3 and Apple TV before 7.2, **allows remote attackers** to **execute arbitrary code** or cause a **denial of service (memory corruption and application crash)** via a crafted web site ...	Critical
Low	CVE-2016-0431	Unspecified vulnerability in Oracle Sun Solaris 11 allows **local users** to **affect availability** via **unknown** vectors related to Solaris Kernel Zones, different than CVE-2016-0419.	Low
	CVE-2012-3128	Unspecified vulnerability in Oracle SPARC T-Series **Servers running System** Firmware 8.2.0 and 8.1.4.e or earlier allows **local users** to **affect confidentiality, integrity**, and **availability** via **unknown** vectors related to **Integrated** Lights Out Manager.	High

TABLE VIII
CONFUSION MATRIX

labels	Critical	High	Medium	Low
Critical	514	**75**	**17**	**0**
High	**56**	461	**51**	**13**
Medium	**27**	**69**	437	**43**
Low	**5**	**28**	**52**	526

V. RELATED WORK

A. *Vulnerability Related Analysis*

Vulnerability rating systems from industry include CVSS, Microsoft Security Bulletin Severity Rating System [2], US-CERT Vulnerability Notes Database [3], SANS Critical Vulnerability Analysis Archive [4]. There is also much research on designing vulnerabilities rating systems with different types of input information [41], [42]. Our work is not about the design of vulnerabilities rating system, but to predict the vulnerability severity according to the CVSS.

Researchers have worked on vulnerability prediction with various combinations of predictors including software metrics, code churn, developer activity metrics, code complexity etc. [38], [43], [44], [45], [46]. Most of the work focuses on recognizing vulnerable components [38], [43], or assess in which ways a system is more likely to be attacked [47], [48]. Bozorgi et al. [49] predict whether and how soon a vulnerability may be exploited by training a classifier on features from text fields, time stamps, cross-references, and other entries in vulnerability reports. Different from existing work, our work predicts severity level of vulnerabilities, and we make prediction based on only vulnerability description.

Vulnerability prediction is related to fault prediction since both vulnerabilities and faults may occur during development process by human mistakes and may lead to serious consequences [50]. However, vulnerabilities and faults are different in that Vulnerabilities always represent abusive functionality instead of wrong or insufficient functionality associated with faults [51] and the size of reported vulnerabilities are much fewer than the reported faults in many projects [52].

B. *Deep Learning in Software Engineering*

Deep learning techniques have been adopted in software engineering studies recently due to its ability to discover intricate properties in large software data [53]. Xu et al. [37] adopt word embeddings and CNN to automatically capture semantics of linkable question and answer posts in Stack Overflow for predicting different types of relations between posts. Chen et al. [54] use word embeddings and CNN to map documents in two languages in a dual-language vector space for cross-lingual question retrieval. Wang et al. [40] predict defects by leveraging Deep Belief Network (DBN) to learn semantic features from token vectors extracted from the Abstract Syntax Trees (ASTs). White et al. [55] apply RNNs to lexical tokens from source code files, which obtains a higher accuracy for code completion than n-gram language model. Mou et al. [56] propose a tree-based CNN for programming language processing including classifying programs and detecting code patterns. Gu et al. [57] trains a RNN encoder-decoder model with pairs of method comments and API call sequences for API usage search by natural language queries.

Inspired by the effectiveness of adopting deep learning techniques in these studies, we design a deep learning architecture to tackle a new multi-class text classification problem for vulnerability analysis, i.e. predict the severity of software vulnerability by only vulnerability description.

VI. CONCLUSION AND FUTURE WORK

This paper identifies the need for a new type of vulnerability analysis, i.e., predict severity level of software vulnerability based on only vulnerability description. This new type of vulnerability analysis can simplify the vulnerability management and prioritization for non security experts, because it requires only the "surface-level" information that describes how a vulnerability works. We present an effective deep learning based text classification method to support this new type of vulnerability analysis. The choice of deep learning methods is driven by their ability to automatically learn word and sentence representations for the prediction task, thus removing the need for manual feature engineering which is a challenging empirical process due to the diversity of software vulnerability and the richness of information in vulnerability description. Our evaluation demonstrates the efficiency of model training, the effectiveness of our model on short-text vulnerability descriptions, and the reliability of the model in face of ongoing disclosure of vulnerabilities. In the future, we will adapt our approach to other vulnerability databases and scoring systems. We will extend our work to predict vulnerability severity level based on vulnerable code fragments. It is a challenge task to embed code into meaningful vector representation and extract features from code using deep learning techniques.

ACKNOWLEDGMENT

This work has partially been sponsored by the National Science Foundation of China (No. 61572349, 61272106). Xiaohong Li is the corresponding author.

References

[1] A. Arora, R. Krishnan, R. Telang, and Y. Yang, "An empirical analysis of software vendors' patching behavior: Impact of vulnerability disclosure," *ICIS 2006 Proceedings*, p. 22, 2006.

[2] C. Microsoft, "Microsoft security response center security bulletin severity rating system," https://technet.microsoft.com/zhcn/security/gg309177.aspx, 2002, [Online; accessed 30-March-2017].

[3] US-CERT, "Uscert vulnerability note field descriptions," http://www.kb.cert.org/vuls/html/fieldhelp, 2006, [Online; accessed 30-March-2017].

[4] I. SANS, "Sans critical vulnerability analysis archive," http://www.sans.org/newsletters/cva/, [Online; accessed 30-March-2017].

[5] FIRST, "Common vulnerability scoring system (cvss) version 2.0," https://www.first.org/cvss/v2/guide#i1.2, 2007, [Online; accessed 30-March-2017].

[6] C. MITRE, "Common vulnerabilities and exposures (cve)," https://cve.mitre.org/, [Online; accessed 30-March-2017].

[7] C. Details, "Cve details," http://www.cvedetails.com/, [Online; accessed 30-March-2017].

[8] M. Sanderson, D. Christopher, H. Manning *et al.*, "Introduction to information retrieval," *Natural Language Engineering*, vol. 16, no. 1, p. 100, 2010.

[9] T. K. Landauer, P. W. Foltz, and D. Laham, "An introduction to latent semantic analysis," *Discourse processes*, vol. 25, no. 2-3, pp. 259–284, 1998.

[10] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[11] C. Zhou, C. Sun, Z. Liu, and F. Lau, "A c-lstm neural network for text classification," *arXiv preprint arXiv:1511.08630*, 2015.

[12] NIST, "National vulnerability database home," https://nvd.nist.gov/, [Online; accessed 30-March-2017].

[13] A. S. Advisories, "Severity levels for security issues," https://www.atlassian.com/trust/security/security-severity-levels, [Online; accessed 30-March-2017].

[14] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[15] E. Nalisnick, B. Mitra, N. Craswell, and R. Caruana, "Improving document ranking with dual word embeddings," in *Proceedings of the 25th International Conference Companion on World Wide Web*. International World Wide Web Conferences Steering Committee, 2016, pp. 83–84.

[16] I. Iacobacci, M. T. Pilehvar, and R. Navigli, "Embeddings for word sense disambiguation: An evaluation study." in *ACL (1)*, 2016.

[17] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu, "From word embeddings to document similarities for improved information retrieval in software engineering," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 404–415.

[18] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[19] S. Reed and N. De Freitas, "Neural programmer-interpreters," *arXiv preprint arXiv:1511.06279*, 2015.

[20] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[21] G. Salton, A. Wong, and C.-S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[22] J. Turian, L. Ratinov, and Y. Bengio, "Word representations: a simple and general method for semi-supervised learning," in *Proceedings of the 48th annual meeting of the association for computational linguistics*. Association for Computational Linguistics, 2010, pp. 384–394.

[23] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, no. Aug, pp. 2493–2537, 2011.

[24] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[25] Y. Zhang and B. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," *arXiv preprint arXiv:1510.03820*, 2015.

[26] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[27] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[28] D. Britz, "Implementing a cnn for text classification in tensorflow," 2015.

[29] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[30] J.-P. Vert, K. Tsuda, and B. Schölkopf, "A primer on kernel methods," *Kernel Methods in Computational Biology*, pp. 35–70, 2004.

[31] I. Ororbia, G. Alexander, T. Mikolov, and D. Reitter, "Learning simpler language models with the delta recurrent neural network framework," *arXiv preprint arXiv:1703.08864*, 2017.

[32] A. Severyn and A. Moschitti, "Learning to rank short text pairs with convolutional deep neural networks," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2015, pp. 373–382.

[33] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*. IEEE, 2008, pp. 346–355.

[34] T. Kenter and M. De Rijke, "Short text similarity with word embeddings," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 2015, pp. 1411–1420.

[35] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[36] S. V. Stehman, "Selecting and interpreting measures of thematic classification accuracy," *Remote sensing of Environment*, vol. 62, no. 1, pp. 77–89, 1997.

[37] B. Xu, D. Ye, Z. Xing, X. Xia, G. Chen, and S. Li, "Predicting semantically linkable knowledge in developer online forums via convolutional neural network," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2016, pp. 51–62.

[38] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, "Predicting vulnerable software components via text mining," *IEEE Transactions on Software Engineering*, vol. 40, no. 10, pp. 993–1006, 2014.

[39] X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, "Hydra: Massively compositional model for cross-project defect prediction," *IEEE Transactions on software Engineering*, vol. 42, no. 10, pp. 977–998, 2016.

[40] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 297–308.

[41] J. Luo, K. Lo, and H. Qu, "A software vulnerability rating approach based on the vulnerability database," *Journal of Applied Mathematics*, vol. 2014, 2014.

[42] H. Ghani, J. Luna, A. Khelil, N. Alkadri, and N. Suri, "Predictive vulnerability scoring in the context of insufficient information availability," in *Risks and Security of Internet and Systems (CRiSIS), 2013 International Conference on*. IEEE, 2013, pp. 1–8.

[43] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 529–540.

[44] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE Transactions on Software Engineering*, vol. 37, no. 6, pp. 772–787, 2011.

[45] Y. Shin and L. Williams, "An empirical model to predict security vulnerabilities using code complexity metrics," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*. ACM, 2008, pp. 315–317.

[46] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," *Journal of Systems Architecture*, vol. 57, no. 3, pp. 294–313, 2011.

[47] M. Howard, J. Pincus, and J. M. Wing, "Measuring relative attack surfaces," in *Computer security in the 21st century*. Springer, 2005, pp. 109–137.

[48] L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia, "An attack graph-based probabilistic security metric," *Lecture Notes in Computer Science*, vol. 5094, pp. 283–296, 2008.

[49] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond heuristics: learning to classify vulnerabilities and predict exploits," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2010, pp. 105–114.

[50] Y. Shin and L. Williams, "Can traditional fault prediction models be used for vulnerability prediction?" *Empirical Software Engineering*, vol. 18, no. 1, pp. 25–59, 2013.

[51] F. Camilo, A. Meneely, and M. Nagappan, "Do bugs foreshadow vulnerabilities? a study of the chromium project," in *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*. IEEE, 2015, pp. 269–279.

[52] O. H. Alhazmi, Y. K. Malaiya, and I. Ray, "Measuring, analyzing and predicting security vulnerabilities in software systems," *Computers & Security*, vol. 26, no. 3, pp. 219–228, 2007.

[53] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[54] G. Chen, C. Chen, Z. Xing, and B. Xu, "Learning a dual-language vector space for domain-specific cross-lingual question retrieval," in *Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on*. IEEE, 2016, pp. 744–755.

[55] M. White, C. Vendome, M. Linares-Vásquez, and D. Poshyvanyk, "Toward deep learning software repositories," in *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*. IEEE, 2015, pp. 334–345.

[56] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, "Convolutional neural networks over tree structures for programming language processing." in *AAAI*, 2016, pp. 1287–1293.

[57] X. Gu, H. Zhang, D. Zhang, and S. Kim, "Deep api learning," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2016, pp. 631–642.