Automatic Part-of-Speech Tagging for Security Vulnerability Descriptions

Sofonias Yitagesu
College of Intelligence and Computing
Tianjin University, Tianjin, China
State Key Laboratory of Communication Content Cognition, China
sofoniasyitagesu@yahoo.com

Xiaowang Zhang
College of Intelligence and Computing
Tianjin University, Tianjin, China
State Key Laboratory of Communication Content Cognition, China
xiaowangzhang@tju.edu.cn

Zhiyong Feng
College of Intelligence and Computing
Tianjin University, Tianjin, China
zyfeng@tju.edu.cn

Xiaohong Li
College of Intelligence and Computing
Tianjin University, Tianjin, China
xiaohongli@tju.edu.cn

Zhenchang Xing
Research School of Computer Science
Australian National University, Data61 CSIRO, Australia
zhenchang.xing@anu.edu.au

***Abstract*—In this paper, we study the problem of part-of-speech (POS) tagging for security vulnerability descriptions (SVD). In contrast to newswire articles, SVD often contains a high-level natural language description of the text composed of mixed language studded with codes, domain-specific jargon, vague language, and abbreviations. Moreover, training data dedicated to security vulnerability research is not widely available. Existing neural network-based POS tagging has often relied on manually annotated training data or applying natural language processing (NLP) techniques, suffering from two significant drawbacks. The former is extremely time-consuming and requires labor-intensive feature engineering and expertise. The latter is inadequate to identify linguistically-informed words specific to the SVD domain. In this paper, we propose an automatic approach to assign POS tags to tokens in SVD. Our approach uses the character-level representation to automatically extract orthographic features and unsupervised word embeddings to capture meaningful syntactic and semantic regularities from SVD. The character level representations are then concatenated with the word embedding as a combined feature, which is then learned and used to predict the POS tagging. To deal with the issue of the poor availability of annotated security vulnerability data, we implement a fine-tuning approach. Our approach provides public access to a POS annotated corpus of ∼8M tokens, which serves as a training dataset in this domain. Our evaluation results show a significant improvement in accuracy (17.72%-28.22%) of POS tagging in SVD over the current approaches.**

***Index Terms*—Fine-Tuning, Part-of-Speech tagging, Unsupervised word embedding, Security vulnerability descriptions**

I. Introduction

Security vulnerability descriptions (SVD) can be found from semi-structured (eg., National Vulnerability Databases (NVD)) and unstructured text sources, such as security bulletins, cybersecurity blogs, etc [1], [2]. Parts-of-speech (POS) tagging, assigning POS tags (e.g., verb, noun, etc.) to tokens in SVD, is challenging due to its content includes high-level natural language descriptions of the text composed of a mixed language studded with codes, domain-specific jargon, vague language, and averviations [1]. In reality, identifying a tokens functional role within a SVD and annotating each of the tokens with POS tags is useful for a wide scope of downstream applications ranging from domain-specific NER [3], [4] to more sophisticated uses such as security vulnerability systems [5]–[7]. Hence, POS tagging is the foundation of many downstream applications and a prerequisite for a wide range of systems [8].

The current methods for POS tagging SVD have often relied on manually annotated training data or applying natural language processing (NLP) approaches [9]. For annotating natural language documents, NLP approaches, such as [10]–[12] propose a POS tagger using machine learning techniques, and [13], [14] propose a BiLSTM-based POS tagger. However, as proven by [15]–[18], NLP approaches trained on natural language documents (e.g., Penn Treebank (PTB) [19] are inadequate for annotating security vulnerability-related texts because these approaches do not use any domain-specific training data, which results in the trained models' lack of domain knowledge to identify the POS of security vulnerability tokens critical for the task of security analyses. These approaches work well for annotating tokens in news articles and related artifacts [15]; however, their accuracy decreases as the input moves to the SVD.

On the other hand, in the software engineering domain, there has been an attempt to develop POS tagging tools by relying on manually annotated training data; for example, Ye et al. [20] apply supervised learning techniques to annotate Stack Overflows, [21], [22] propose a rule-based POS tagger for source code annotation. Such approaches can achieve the best performance; however, annotating data requires much human effort, time-consuming, and labor-intensive feature engineering, which becomes quite costly as datasets grow. Overall, both NLP and software engineering approaches cannot be directly applied on SVD; moreover, human annotations for neural models often require a considerable time and effort.

Our idea is to construct a neural network architecture that

trains the source model on a large newswire corpus (high-resource out-of-domain-data) and then use the learned knowledge (parameters, weights) of this model as prior knowledge for the target model, which is further fine-tuned (trained) on a security vulnerability data (low-resource in-domain-data). The source model captures common features found in the security vulnerability corpus. However, there are significant words specific to the vulnerability domain of different lexical structures than regular English words and punctuation. The source model lacks domain knowledge and fails to investigate the unique characteristics of SVD. We empirically prove that the accuracy decreases as the input moves to the SVD. Thus, a fine-tuning approach [23]–[25] is applied to address the need for annotated data for POS tagging of security vulnerability texts. This approach intends to add the domain knowledge to the pre-trained source model (i.e., target model) that identifies the POS of tokens specific to the SVD domain. However, a large number of POS annotated security vulnerabilities are required for effective supervised learning techniques. Unfortunately, training data dedicated to security vulnerability research is not widely available, results in identifying POS of linguistic-informed SVD more challenging.

In this paper, we propose an automatic approach to assign POS tags to tokens in SVD. We first build a "webScraper" tool to crawl an extensive collection of unstructured descriptions of vulnerabilities from NVD and its associated components and automatically build an unannotated vulnerability corpus (Table I). In the POS tagging task, the way how we segment texts into tokens has an impact. Thus, we design a custom tokenizer that handles the security vulnerability jargon. We construct an unsupervised word embedding model to extract domain-specific features. Our embedding computes vector representations of words as they are tokenized. The resulting word vector file is then concatenated with character-level representations as a combined feature, which is then learned and used to predict the POS tagging. In the framework of our fine-tuning approach, we first train a model for a source task and use the learned parameters to initialize the model parameters for training the target task. Our approach provides a POS annotated corpus of ∼8M tokens, which serves as training data in this domain, and supports downstream applications. We make the following contributions:

- We propose an automatic approach to assign POS tags to tokens in SVD. We construct an neural architecture that trains the source model on a large out-of-domain-data, and then use the learned knowledge of this model to the target model, which is further fine-tuned on in-domain-data. In our approach, character-level representation and unsupervised word embeddings are concatenated as a combined feature, which is then learned and used to predict the POS tagging.
- We present an unsupervised word embedding model based on CBOW with a negative sampling approach to extract syntactic and semantic features from security vulnerability text. Besides, we present a custom tokenizer designed to handle security vulnerability-specific vocabularies. These features empower the approach to assigning proper tags to a sequence of security vulnerability tokens.
- We introduce a new security vulnerability POS annotated corpus of ∼8M tokens, which serves as training data for supervised learning that correctly identifies other text reports in this domain. Experiments on the security vulnerability datasets demonstrate that our approach achieves the best performance in accuracy (93.22%) and precision (93.16%). We provide all our implementations; https://bitbucket.org/MerejaKuat/supp_material/src/master/

II. Background and Motivation

A. Background of Security Vulnerabilities

Online security vulnerability databases, such as NVD and its associated components, accumulate track of historical security vulnerability-related issues in semi-structured and unstructured text formats [1], [2]. They give crucial supports and are extensively used by software developers and security analysts. Usually, online users (contributors) report the newly-discovered vulnerability information to these vulnerability databases or security blogs. When the vulnerability report is confirmed, a CVE-ID is assigned, and an issue report to describe the vulnerability situation is created. Security analysts refer to these vulnerability situations for the newly discovered vulnerabilities and manually analyze the textual descriptions of vulnerability issues. The security analyst must first carefully read the vulnerability report (e.g., description), and elicit the keywords such as systems that are likely to be affected, the operating systems environment for which the attack can occur, the versions of products affected, and review the weakness in source code files to find the patch and fix the vulnerable parts.

The above activity is labor-intensive, tedious, and time-consuming, especially for massive textual information with thousands of vulnerability reports. Manual analysis of security vulnerabilities demands high expertise and imposes a huge burden on security analysts, which inevitably limits productivity. Thus, it is extremely useful to automate this process and recommend potentially vulnerable source files to security experts and developers with a given vulnerability description [4]. Hence, POS tagging is one of the early phases to be automated. In fact, to understand the meaning of the vulnerability reports, each word's POS in the description is of particular importance. Thus, we extract all the available vulnerability issues, and POS annotate each token, and provides public access to support the downstream applications, such as security systems.

B. Motivation by Example

We consider known security vulnerabilities reported to NVD (Fig. 1) and demonstrate the critical challenges of state-of-the-art POS tagging in security vulnerability domain.

CVE-ID	Description
CVE-2018-1000631	Battelle V2I Hub 3.0 is vulnerable to SQL injection. A remote attacker could send specially-crafted SQL statements to the tmx/TmxCtl/src/lib/PluginStatus.cpp and TmxControl::user_info() function, which could allow the attacker to view, add, modify or delete information in the back-end database. Publish Date : 2018-12-28 Last Update Date : 2019-01-11
CVE-2005-2276	Cross-site scripting (XSS) vulnerability in Novell Groupwise WebAccess 6.5 before July 11, 2005 allows remote attackers to inject arbitrary web script or HTML via an e-mail message with an encoded javascript URI (e.g. "j&#X41vascript" in an IMG tag. Publish Date : 2005-07-26 Last Update Date : 2017-07-10

Fig. 1. Motivating Example: Reported Security vulnerabilities

Fig. 1 describes a vulnerability situation where the token's functional role or POS of each token in the description is of particular importance. For example, "$TmxControl :: user\_info()$" function gives essential information to the security analysts and developers about which functions of product "$V2I\ Hub$ 3.0" is vulnerable to $SQL\ injection$. Current approaches split and identified "()" as punctuations and "$TmxControl :: user\_info$" as nouns; however, it refers to a programming APIs function, which is identified as a proper noun in a security vulnerability. Moreover, decimal numbers "3.0" and "6.5" that respectively follow a product "$V2I\ Hub$" and "$WebAccess$" provide critical information about which versions of a product are vulnerable to $SQL\ injection$ or $XSS\ vulnerabilities$. The NLP approaches identify "3.0" and "6.5" as cardinal numbers; however, decimal numbers that follow a software product are expected to be versions of that product and not quantities of it [4]. Hence, in the security vulnerability domain, they are identified as proper nouns. Besides, a string of words starting with capital letters is expected to be a vendor $(e.g., Battelle)$ or product name $(e.g., V2IHub)$, not a company name. Hence, they are identified as proper nouns. The examples show that current approaches fail to accurately capture the lexical structure of domain-specific jargon. They treat the report as well-formatted English texts [1] and assign incorrect POS tags, results in misleading downstream applications. Hence, it is strongly suggested that the SVD should be treated differently from well-formatted documents and needed specialized methods trained using security vulnerability data [1], [15].

III. OVERVIEW OF PROPOSED FRAMEWORK

In this section, we describe an overview of the workflow of our approach (see Fig. 2). Our approach consists of four main components: (1) Corpus Preparation, (2) Word tokenization, (3) Unsupervised Word Embedding, and (4) POS tagging.

1) Corpus Preparation: The first step in the approach is identifying textual resources that describe security vulnerabilities and threats. Using our *"WebScraper"* tool, we automatically extract security vulnerabilities reported in NVD and its associated components and build an unannotated corpus.

2) Word Tokenization: We present a custom tokenizer designed to handle security vulnerability-specific vocabularies. The tokenizer component splits the vulnerability corpus into sentences and identifies vulnerability specific tokens. Ultimately the tokenizer produces a "list of tokens".

3) Unsupervised Word Embedding: Our embedding is trained from scratch to extract features from an unannotated corpus. The embedding model is trained over the combined corpus of the PTB and vulnerability corpora. The trained model is then used as a component of the POS tagger.

4) POS Tagging: We follow a two-stage process: we initially train the POS tagger with PennTreebank (PTB) training data and evaluate the performance in tagging vulnerability tokens, resulting in the accuracy drops. Using gold set data, we fine-tune the POS tagger to accurately annotate tokens in the security vulnerability text. The outcome is a fine-tuned model that can induce a POS tag for new vulnerability tokens with much improvement in accuracy than state-of-the-art methods. Lastly, we feed the unannotated corpus to the fine-tuned tagger to produce (predict) a POS annotated corpus of ∼8M tokens.

IV. OUR APPROACH

This section describes our proposed approach for POS tagging SVD (see Fig. 2).

A. Corpus Preparation

1) Unannotated Corpus Preparation: Unlike traditional POS taggers, e.g., the PTB Project [19], a corpus dedicated to security vulnerability research is not widely available [4]. Thus, we begin our work by preparing a corpus of security vulnerabilities from online data sources. To do so, we develop an automatic "$WebScraper$" tool that scrapes unstructured vulnerability descriptions from NVD, Common Attack Pattern Enumeration and Classification (CAPEC), Common Weakness Enumeration (CWE), Product Dictionary (CPE), Common Vulnerabilities and Exposures (CVE), and Twitter webservers [1]. Our tool is automatic and generic. It scraps the specified website for all available links; it locates the "URL" of the webservers, downloads all XML and JSON vulnerability feeds from 1999 to 2019, extract vulnerability information from all tags, collects the contents of specified tags, and built an unannotated corpus, as shown in Table I.

TABLE I
SCRAPED SECURITY VULNERABILITY DESCRIPTIONS

Security Vulnerability Data Sources	Sample Size	Number of Tokens	Time Period Covered
NVD (CVE and CPE)	32.04MB	5,342,280	1999-2019
CWE	17.98MB	2,288,095	1999-2019
CAPEC	6.09MB	491,927	1999-2019
Twitter	4.95MB	400,000	- Recent
Total	61.06MB	8,622,302	1999-2019

To keep the vulnerability data correct, we only collect the vulnerabilities that have CVE-ID. We extract the unstructured content of specified tags such as vulnerability descriptions, modification_comment, impacts, notes, and so on. The preprocessing is performed by employing the Natural Language Toolkit (NLTK) [11]. However, some non-natural language contents extracted from the description tag of security vulnerabilities are hard to be preprocessed directly using the conventional NLP toolkits. To alleviate this issue, we write a script to preprocess domain-specific contents. Besides, we programmatically remove code segments from the vulnerability data dump. After text cleaning, we obtain a security vulnerability corpus from the vulnerability data dump, which has about 1,000,000 most frequently used words, usually greater than 3 (205,610 unique terms) and about 400,914 vulnerability sentences (8,622,302 tokens).

2) Gold Set Annotation: To the best of our knowledge, there are no publicly available POS annotated training data on the security vulnerability domain. Thus, we create a gold set (a small, high-quality set of POS annotated vulnerability tokens) to fine-tune our POS tagger and evaluate the performance of other approaches. Given the fact that it is hard to manually

[1]https://nvd.nist.gov/, https://capec.mitre.org/, https://cwe.mitre.org/, https://twitter.com/CVEnew?lang=en, https://cve.mitre.org/

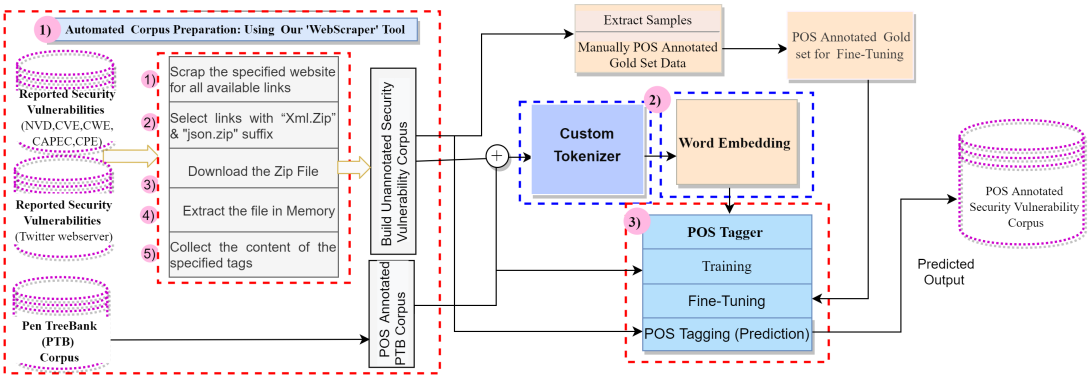

Fig. 2. The overview framework of our approach

annotate all tokens in vulnerability descriptions, we randomly draw 2% (8,018 sentences or 172,446 tokens) samples from the total vulnerability sentences (Table I), and manually POS annotated them with a total of 47 tags from PTB [19] tag sets.

We hire 10 participants, 4 are from the security domain (a Ph.D. student at Tianjin University and 3 are from Information Network Security Agency), and 6 are experts in English (Ph.D. in linguistics and assistant professors in Addis Ababa University), to assign POS tags to tokens in the sampled vulnerability descriptions. We request participants to use all POS tags in a PTB tag set. Before annotation, we give all annotators a 6h tutorial on annotating the tokens to reach a consensus. We assign each vulnerability description to two separate participants to resolve disagreements between annotators and ensure the annotation quality. These two participants discuss and make a final judgment on the POS tag of a token. However, we have an annotator with expertise in security and a second annotator with linguistics knowledge sit together, discuss and annotate the most challenging jargon and domain-specific vocabularies.

Furthermore, we randomly select 40 sentences (862 tokens) from the gold set and re-annotated them from scratch. The purpose is to estimate the inter-agreement of the annotations among different annotators. We compare these 862 re-annotated tokens with their annotation results. Only 25 tokens are tagged differently, which results in an inter-agreement rate [26] of 97.09%.

B. Custom Tokenizer

In the POS annotation task, the first issue that we must address is how to segment the text into tokens. Tokenizers designed for well-formatted documents cannot handle SVD, which contain domain-specific jargon [27]. Thus, we design a custom tokenizer that can handle SVD-related vocabularies. Our tokenizer uses regular expressions to match the "$parameter$": "$pattern$" of the function "$(text, pattern)$" to enable it suitable for the security vulnerability jargon. Our tokenization delimits word tokens in vulnerability descriptions by a preceding and the following space. A compound is tokenized as a single lexical unit if it appears as one orthographic word with a hyphen $(Cross-site)$, underscore $(user\_info())$, dot $(PluginStatus.cpp)$, or a string of words starting with a capital letter $(WebAccess)$ or two and more morpho-syntactic tokens if tokens are separated with white space (see Table II).

TABLE II
OUR TOKENIZATION COMPARED TO THE EXISTING.

Input Vulnerability Tokens	Existing Tokenizer	Our Tokenizer
TmxControl::user_info()	Tmx,Control,:,:,user,_,info,(,)	TmxControl::user_info()
WebAccess 6.5	Web,Access, 6,.,5,	WebAccess, 6.5
Cross-site scripting	Cross,site, scripting	Cross-site, scripting

The existing tokenizer split colored text into 9 separate tokens, which does not have meaning

As shown in Table II, our tokenizer preserves the integrity of security vulnerability tokens and the sentence structure. Hence, the file name $(file.name)$, $(token\_token)$, or (token-token) is a single word. For example, it treats "$TmxControl :: user\_info()$" as a single token, instead of a sequence of 9 tokens. Moreover, our tokenizer does not split method names from parentheses; parentheses and dots that appear in a method are regarded as part of the method. However, it studies separate parentheses, i.e., "(" and ")", as punctuations.

C. Unsupervised Word Embedding

According to [28]–[30], popular word embedding algorithms like word2vec [31] and GloVe [32], which are initially developed for text, cannot be directly applied to a domain-specific task. Security vulnerability data available in a variety of forms and includes domain-specific vocabularies and jargon [15]. Thus, we present unsupervised word embedding for security vulnerability notions learned using an unannotated vulnerability text and then used as a component of our POS models. Our embedding is based on CBOW [31], described as follows: Let W be a vulnerability corpus, a sequence of words w_1^T, a window by the parameter c, c words at the left and right of the target word; thus we use the c context vectors to predict the target word. The probability of a Softmax is defined by [31], and described in [33] as:

$$p\left(w_t|w_c\right) = \frac{\exp\left(v_{w_t}^T u_{w_c}\right)}{\sum_{w=1}^{W} \exp(v_w^T u_{w_c})} \quad (1)$$

where u_w is a target vector, and v_w is a context vector. Since Softmax is too expensive, negative sampling approaches [31], [34] is used, which samples k context words that do not appear in the current window. As shown in Fig. 3 (a), the context

words are embedded, followed by a dot product is computed. We compare how similar the embeddings for the context (c) and target (t) words are by computing the cosine similarity of their corresponding vectors, and use this similarity to assess.

D. Parts-of-Speech Tagging

Existing POS taggers trained on well-formatted documents are inadequate to identify linguistic-informed words specific to the security vulnerability domain [15]. Thus, we present an automated approach for POS tagging of vulnerability text.

1) Problem: Given a description of security vulnerabilities, we define the problem of POS tagging as a task of assigning POS tags (eg., nouns, verbs, and so on) to each sequence of tokens in the description. Formally, let a sequence of n untagged security vulnerability tokens as w_1^n and a finite set of m tags (eg., nouns, verbs, and so on) t_1^m. Thus, for any given sequence of n tokens, it holds $w_1^n \in \mathcal{W}^n$ and for any given sequence of tags we have $t_1^m \in \mathcal{T}^m$, where $\mathcal{W}$ and $\mathcal{T}$ are the number of possible tokens and tags respectively. Our goal is to construct a model that predict an optimal sequence of POS tags $\hat{t}_1^m$ that corresponds to an input token sequence $\hat{w}_1^n$ such that the posterior probability $P\left(\hat{t}_1^m | \hat{w}_1^n\right)$ is optimal.

2) BiLSTM Model Architecture: We construct a neural architecture based on BiLSTM [35] that performs both a forward and backward pass over the input sequence [36], [37]. The output vector h_n is then computed by concatenating the corresponding forward and backward passes: $h_n = \text{LSTM}_f\left(x_{1:n}\right) \oplus \text{LSTM}_b\left(x_{n:1}\right)$. For the POS-tagging task [38], [39], this has an impact on lessening the vanishing gradient for long sentences. Fig. 3 shows our proposed architecture.

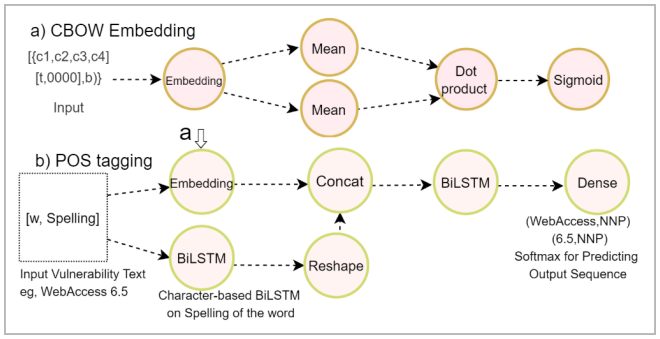

Fig. 3. Example of a neural architecture in our POS tagger

Fig. 3 shows that we feed the raw data corpus as it spelled. To preserve both semantic and syntactic information of words [40], we join character-level and word-level embedding to get a combined embedding feature. Hence, each word from the input SVD is represented by a combination of two vectors. We pre-trained word-level embedding from an unannotated security vulnerability corpus. To deal with out-of-vocabulary(OOV) words, we represent each word with a vector that contains morphological information generated by character-based BiLSTM. In general, character embeddings $(c1, ..., cr)$ are learned by passing a BiLSTM over characters:

$$h_i^{l,forward} = \text{LSTM}\left(c_i, h_{i-1}^{l,forward}\right)$$
$$h_i^{l,forward} = \text{LSTM}\left(c_i, h_{i-1}^{l,forward}\right)$$

Then, we utilize the last hidden vectors from each of the LSTM components, concatenate them together, and pass the result through a separate non-linear BiLSTM layer.

$$h_t = \left[h_i^{l,forward}; h_i^{l,backward}\right]$$

Now, we have two feature representations for each word, where $w(t)$ is an embedding learned on the word-level, and $c(t)$ is a representation created from individual characters at the t^{th} word of the input text. Our approach is to concatenate the two vectors $(\vec{w} \oplus \vec{c})$ and use this as the new word-level representation for the sequence labeling task: $w_{\text{concat}} = [w; c]$. Thus, we pass a sequence of vectors $D \in \mathbb{R}^{n \times d}$ via a separate BiLSTM, where n is the length of the sequence and d is the dimensionality of the word vectors corresponding to the sequence's tokens. At each time step t, the output of the LSTM h_t is passed to a softmax output layer, which generates a probability distribution over the tag vocabulary $\mathcal{V}$. This technique enables for the optimal tag to be chosen using $\hat{y}_t = \arg\max_{y \in \mathcal{V}} P\left(y \mid w_t\right)$

3) Fine-tuning: Let $D_s = (X_i^s, Y_i^s)$, where $i = 1$ to K be the training set of K samples from the source dataset and $D_t = \left(X_j^t, Y_j^t\right)$, where $j = 1$ to N be the training set of N samples from the target domain. Our goal is to improve tag prediction accuracy on the target domain D_t by using the knowledge learned from the source domain D_s, as shown in Fig. 4.

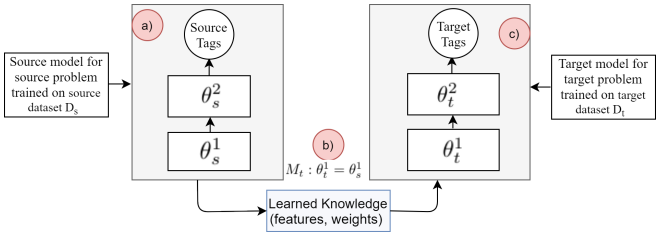

Fig. 4. Fine-tuning Architecture

In Fig. 4, we have a source neural model M_s for a source problem with a set of parameters θ_s, where $\theta_s = \left(\theta_s^1, \theta_s^2\right)$; and a target model M_t for the target problem with a set of parameters θ_t where $\theta_t = \left(\theta_t^1, \theta_t^2\right)$.

a) Source Model: We learn the source model on annotated data from the source problem on a large newswire dataset D_s. This model, because it does not use any domain-specific training data, can identify natural language texts; however, it does not identify security vulnerability jargon. For example, the software versions such as "3.0" and "6.0" and APIs such as "$TmxControl :: user\_info()$" are identified as cardinal numbers and nouns; however, they are all proper nouns.

b) Transform: We transfer the learned knowledge in the form of features or weights learned from the source model M_s for training newer models for the target task: $M_t : \theta_t^1 = \theta_s^1$.

c) Target Model: The target model is fine-tuned by training the target model on the manually POS annotated security vulnerability dataset (gold set) D_t. Fine-tuning has proven effective in low-resource in-domain-data [23]–[25]. It can be performed either the last [41] or several of the last layers of a pre-trained model and leaving the remaining layers frozen [42]. We fine-tune the last layers of the target model by minimizing the remaining layers' training rate to 0.001.

The target model can learn from the small annotated gold sets examples, and identify the software versions such as 3.0 and 6.0, and API ('$TmxControl :: userinfo()$') as proper noun.

We adapt the collections of POS tags, such as nouns, verbs, adjectives, and so on, from the PTB [19] tagset, and we follow their guidelines. We refer the reader to PTB [19] extended POS tagset. We use the morphological rules to assign all possible POS tags to each token. However, the gold sets provide the most critical ground truth for the model to learn how POS tags are assigned to each security vulnerability tokens. For each token in the SVD, a tagger returns all possible POS tags. We use "()" to represent one token with POS tags. The tagger takes SVD (see Fig. 1) and provides (prediction output) a POS annotated corpus of ∼8M tokens, as shown in Fig. 5.

Input (Fig.1)	POS Tagging Output (Prediction)
CVE-2018-1000631	(Battelle,NNP)(V2I,NNP)(Hub,NNP)(3.0,NNP)(is,VBZ)(vulnerable,JJ)(to,TO)(SQL,NNP)(injection,NN)(A,DT)(remote,JJ)(attacker,NN)(could,MD)(send,VB)(specially_crafted,JJ)(SQL,NNP)(statements,NNS)(to,TO)(the,DT)(tmx/TmxCtl/src/lib/PluginStatus.cpp,NNP)(and,CC)(TmxControl::user_info,NNP)(function,NN)(which,WDT)(could,MD)(allow,VB)(the,DT)(attacker,NN)(to,TO)(view,VB)(add,NN)(modify,NN)(or,CC)(delete,VB)(information,NN)(in,IN)(the,DT)(back-end,JJ)(database,NN)
CVE-2005-2276	(Cross_site,JJ)(scripting,VBG)(XSS,NNP)(vulnerability,NN)(in,IN)(Novell,NNP)(Groupwise,NN)(WebAccess,NNP)(6.5,NNP)(before,IN)(July,JJ)(11,CD)(2005,CD)(allows,VBZ)(remote,JJ)(attackers,NNS)(to,TO)(inject,VB)(arbitrary,JJ)(web,JJ)(script,NN)(or,CC)(HTML,NNP)(via,IN)(an,DT)(e_mail,JJ)(message,NN)(with,IN)(an,DT)(encoded,JJ)(javascript,NN)(URI,NNP)(e.g,NN)(j&#X41vascript,NNP)(in,IN)(an,DT)(IMG,NNP)(tag,NN)

Fig. 5. An annotated security vulnerability descriptions by our tagger

Fig. 5 shows that SVD tokens are identified as proper noun, for example, "$Battelle$" and "$Novell$" are vendors and identified as "NNP" (i.e., proper noun). Likewise, "$TmxControl :: user\_info()$" function is identified as "NNP", while it was incorrectly identified as noun. Natural language techniques identify "3.0" and "6.5" as "CD" (i.e., cardinal numbers); however, decimal numbers that follow a software product are versions of that product and not quantities of it. Hence, in the SVD domain, they are identified as "NNP". Moreover, our tagger identifies the difference between a decimal number and the version number. For example, "11", "2005" are tagged as "CD".

V. Experiments and Evaluations

In this section, we construct four main sets of experiments to evaluate our approach to automatically annotating likely POS tags to a sequence of security vulnerability tokens by investigating the following research questions:

RQ1: How effective is our POS tagger on assigning POS tags to each token in SVD?

RQ2: How effective are the POS tagging approaches on SVD, compared with their performance on well-formatted texts?

RQ3: To what extent domain-specific word embedding features impact the performance of our POS tagger.

RQ4: To what extent the particular features affect the accuracy of our POS tagger.

A. Experimental Setup

1) Model configurations: Our POS tagger is implemented based-on TensorFlow [43], Keras [44] and SGD [45]. A BiLSTMs with 256 units are used and optimized using ADAM [46] with minibatch size 32 for a total of epochs for embedding 75 and 185 for POS tagging. We chose the parameters that show the best accuracy on the development set, and we report the score on the test set. A grid search is employed to select appropriate hyperparameter values. In all cases, a 10-fold cross-validation methodology is applied. Then, the 10-fold results are averaged and used to select the best models and variants for comparison.

2) Training Procedure: Fig. 6 describes the implementation framework that we use to build and train word embedding and POS tagger models.

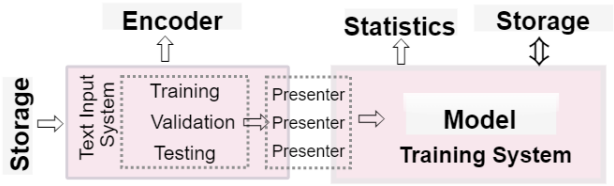

Fig. 6. An implementation framework within BiLSTM architecture

In Fig. 6, the input system reads texts from the file—we scrap from online data sources (see Table I). Then, it applies text processing and partitions the data into training, validation, and test dataset. For every dataset, we define an object called presenter, which prepares the dataset as the model expects. The presenter object defines the tensor as a batch, and it feeds to the model as a continuous subsequence. The training needs Keras embedding layer as an input feature, unique words that are converted to an integer, dictionary size, types of input dimension, and embedded vector dimension. In all cases, the data is partitioned into training (70%), validation (10%), and test dataset (20%). The training data is given, and the model is identified through backpropagation. Likewise, the validation data set is given, and we capture the statistics. Lastly, we get the required model by assembling those software components. Our framework is like a library, similar to Keras and tensor flow. We develop a library for our purpose of using Keras and other python libraries.

3) Evaluation Metrics: We use the standard evaluation metrics, i.e., accuracy, precision, recall, and F1, to measure the effectiveness of our POS tagger. Accuracy measures the percentage of word tokens in a corpus which are correctly tagged. For each category of POS tags, precision measures what percentage the output tags are correct. The recall measures the extent to which all correct annotations are found in the output of the tagger. F1-score is the harmonic mean of precision and recall. The evaluation metrics are calculated as:

$$P = \frac{TP}{TP+FP}, R = \frac{TP}{TP+FN}, \text{ and } F1 = \frac{2\times P\times R}{P+R} \quad (2)$$

4) Dataset: We perform experiments on two datasets:

(a) PTB [19]: It is a POS annotated well-formatted documents collected from the Wall Street Journal (WSJ) and publicly available. It contains 3,914 text sentences consisting of 100,676 tokens, which is only used as a training data for the source model.

(b) Security vulnerability descriptions (see Table I): It is an unannotated SVD extracted from online data sources. It contains 400,914 sentences consisting of 8,622,302 security vulnerability tokens. A portion of unannotated SVD, 2%(8,018 sentences or 172,446 tokens), is manually POS annotated (see

gold set annotation). We use 7,000 sentences or 150,546 tokens for fine-tuning; the rest 1,000 sentences or 21,506 tokens are used for comparisons with other approaches.

*B. Evaluating Parts-of-Speech Tagger **(RQ1)***

In this section, we perform two subsets of experiments to investigate the effectiveness of our POS tagger on assigning POS tags to a sequence of security vulnerability tokens. Since we do not have POS annotated training data, we conduct a two-step experiment by implementing the fine-tuning approach.

Experiment 1: In the first experiment, we train the POS tagger (source model) using the POS annotated training data from PTB. We then evaluate the tagger's performance with SVD (we use 1k sentences or 21,506 tokens of gold sets). The evaluation is performed automatically in Keras; It evaluates the performance by comparing the tags assigned by the tagger and our gold test set. However, while our goal is to build a tagger that automatically annotates the security vulnerability tokens with high precision and recall, as shown in (Table III row2), the performance drops to 65%. The result indicates that the taggers trained on well-formatted documents are ineffective at annotating vulnerability tokens.

Experiment 2: In the second experiment, we train the fine-tune tagger (target model) on security vulnerability training data (gold set). Fine-tuning requires a small amount of training data; thus, we leverage a manually POS annotated gold set (7k sentences or 150,546 tokens) and fine-tune the tagger with a minimal training rate. We re-train the last layer of the model by keeping the rest layers of the architecture's training rate 0.0001. We then evaluate the tagger's performance when applying it to the security vulnerability tokens (we use the same gold set (1k sentences or 21,506 tokens) that we used in the first experiment). As shown in (Table III row3), the fine-tuned POS taggers performance rises to 93.16%.

TABLE III
THE PERFORMANCE OF OUR POS TAGGER WHEN TRAIN IT WITH PTB AND SECURITY VULNERABILITY DATASETS.

System	Training Data	Prec.	Recall	F1	Acc
POS tagger (source model)	Well-formatted text	65%	54.98%	59.57%	**95.01%**
POS tagger + Fine-tune	Gold set data	**93.16%**	**91.06%**	**92.18%**	93.22%
Improvement		28.16%	36.08%	32.61%	1.79%

Table III shows the achievement of our POS tagger (source model) when trained it with well-formatted text (PTB), and the target model—fine-tune the source model with a security vulnerability gold set. Overall, the PTB trained tagger correctly identify 95.01% of the well-formatted English texts, while the fine-tuned tagger achieved an accuracy of 93.22%. Our results indicate that the fine-tuned POS taggers accuracy declines by 1.79% compared to the PTB trained POS tagger. However, the recall is important, and it rises from 54.98% to 91.06%. Moreover, the precision drastically up from 65% to 93.16%. Similarly, the F1 climbs from 59.57% to 92.18%.

The result shows that the POS tagger (source model) trained with PTB data cannot achieve a recall above 59.57%, indicating that it will fail to annotate nearly three-fifth of vulnerability tokens. However, a fine-tuned tagger has an F1 score of 92.18%. Based on the results, we can verify that our approach is useful and adequate to annotate vulnerability-specific vocabularies and jargon. The fine-tuned POS tagger can be useful for annotating other vulnerability tokens in this domain. In comparison, PTB trained tagger gives poor results as the input moves to the SVD.

Furthermore, as the evaluation gold set is small (1k), we manually analyze why two experiments provide two different results. Our manual investigation shows that the PTB trained POS tagger can identify only those well-formatted natural language texts and failed to identify that domain-specific jargon. In comparison, the fine-tuned POS tagger can identify the vulnerability tokens as well as the well-formatted texts.

Thus, we verify that the tagger trained with general English text can not be directly used in the security vulnerability domain. However, we can verify that we can initially train the POS tagger (source model) with the publicly available well-formatted general texts and fine-tune the tagger by re-training only some layers of the architecture with a small amount of gold set rather than manually preparing the training data.

To sum up, we show that our fine-tuning approach is effective, a gold set is too sensitive, and errors in the gold set annotation affect the taggers overall performance. Thus, the gold set can be small, but it should be a high-quality set of POS annotated vulnerability tokens. This gold set specifies for each example what the correct output of the tagger should be. Our tagger can learn from these examples, and the correctness of its performance can be measured on annotated examples that are not seen during training. The other important task is our custom tokenization; it treats the vulnerability or software engineering data such as functions, methods, and codes as a token rather than splitting them as NLP tokenizers do. We can verify that the custom tokenization affects the performance of the tagger; without this, it performs only 84.5%.

*C. The Comparison with Other Approaches **(RQ2)***

1) Compare Against Pre-trained Taggers: In this subsection, we compare the performance of our POS tagger with other pre-trained POS tagging approaches to assigning likely POS tags to a sequence of tokens. Due to the lack of SVD taggers to perform straight comparisons, we choose two popular "off-the-shelf" taggers; namely, Stanford[2] and NLTK[3], that are pre-trained with well-formatted documents (PTB) collected from WSJ. First, we evaluate both pre-trained POS taggers with the gold evaluation set, 1k sentences, or 21,506 SVD tokens. Second, we evaluate those taggers with PTB POS annotated evaluation set (1k sentences), a well-formatted English text. For all experiments comparing pre-trained taggers with our tagger, the evaluation set used by all the taggers is always the same. Besides, all pre-trained taggers and our tagger use the same PTB tagset.

We investigate the effectiveness of those "off-the-shelf" POS taggers on SVD, compared with their performance on a well-formatted document (see Fig. 7). We apply the POS taggers on the gold set and evaluate their performance by

[2]https://nlp.stanford.edu/software/tagger.shtml
[3]https://www.nltk.org/api/nltk.html

comparing the tags assigned by these POS taggers and our gold set. The tagger is regarded as correct if the tag it guesses for a given token is the same as the gold set tag. To assure impartial comparison, we only compare the POS tagging accuracy to measure the achievement of POS taggers on SVD.

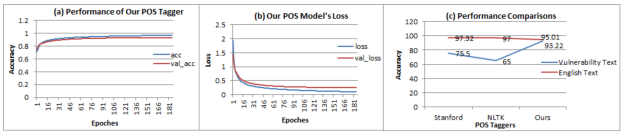

Fig. 7. (a) is the performance of our POS tagger, (b) is the loss, and (c) is performance comparison of all the three taggers with 1k evaluation gold set.

Table IV shows the accuracy of POS taggers on gold evaluation set and their accuracy on a well-formatted text.

TABLE IV
PERFORMANCE OF THE POS TAGGERS ON SECURITY VULNERABILITY DESCRIPTIONS, COMPARED WITH THEIR PERFORMANCE ON A WELL-FORMATTED ENGLISH LANGUAGE TEXTS.

System	Vulnerability Descriptions	English Text	Error
Stanford Tagger v3.9.2 (2018)	75.5%	**97.32%**	21.82%
NLTK Tagger v3.4.5 (2019)	65%	97%	32.00%
Fine-tune POS Tagger (Our)	**93.22%**	95.01%	**1.79%**

From Table IV, we note that the POS taggers achieve an accuracy of 65% to 93.22% on the sample gold set data. The fine-tune POS tagger performs the best, 93.22%, which outperforms that of the Stanford Tagger (75.5%) and NLTK (65%). Stanford and NLTK Taggers perform poorly in recognizing SVD, and accuracy drops from 97.32% to 75.5% in Stanford Tagger and 97% to 65% in NLTK Tagger. The last column in Table IV shows the ratio of the accuracy of POS taggers on SVD and their accuracy on the well-formatted English text. The error reduction is as significant as 32.00% in NLTK and 21.82% in Stanford Tagger. Although the accuracy of our POS Tagger reduces comparatively, the difference between its performance on security vulnerability texts and the well-formatted document is, only 1.79%.

2) Compare Against Re-trained Taggers: Stanford and NLTK tagger provide a set of features and optimization options for users to configure. Thus, we re-train both Stanford and NLTK taggers using the SVD training set. For a fair comparison, we use the same training set (7k gold set) and evaluation set (1k gold set) that we used for fine-tuning. Hence, all taggers, including our fine-tuned tagger, are trained and evaluated with the same gold set. Similar to our POS tagger, we partition the 7k gold set into training (70%), validation (10%), and testing (20%). We also use the same evaluation methods as our fine-tuned POS tagger. We apply the POS taggers on the evaluation set and determine their performance by comparing the tags assigned by those POS taggers and our gold set annotation. The tagger is regarded as correct if the tag it guesses for a given token is the same as the gold set tag. Then, we report the accuracy, precision, recall, and F1 to measure POS taggers' performance on SVD (see Table V).

Table V shows the evaluation results for all the three POS tagging approaches on the security vulnerability dataset. Overall, 93.22% of the security vulnerability tokens are correctly tagged by our POS tagger, while Stanford and NLTK taggers achieve an accuracy of 90.8% and 87.1%, respectively. Results from our evaluation show that the NLTK tagger gives comparatively lower precision and recall. While its precision is 85.5%, the recall drops to 80%. Compared with NLTK, Stanford tagger improves both precision and recall by 3.5% and 5.5%, respectively. Among the three compared approaches, our POS tagger achieves the best precision and recall. While the precision is 93.16%, the recall reaches to 91.06%. Thus, our tagger achieves 7.66% higher precision and 11.06% higher recall compared to the NLTK tagger and 4.16% higher precision and 5.56% higher recall than the Stanford tagger. Moreover, our tagger has the highest F1 score (92.18%), while Stanford achieved (88.3%) and the lowest NLTK (84%). The results indicate that our POS tagger is successful in annotating vulnerability tokens than Stanford and NLTK taggers. For example, our tagger has precision (93.16%), which indicates that when tokens are annotated, in more than 93.16% of the cases, they are indeed annotated with the correct POS tags.

TABLE V
PERFORMANCE COMPARISONS OF THE POS TAGGERS ON SECURITY VULNERABILITY DESCRIPTIONS

System	Acc.	Prec.	Recall	F1
Stanford Tagger v3.9.2 (2018)	90.8%	89.00%	85.5%	88.3%
NLTK Tagger v3.4.5 (2019)	87.1%	85.5%	80.00%	84.00%
Fine-tune POS Tagger (Our)	**93.22%**	**93.16%**	**91.06%**	**92.18**

Furthermore, we draw samples randomly from the POS annotated results, 100 samples from each tagger's output, and manually re-annotated them from scratch. We compare those re-annotated tokens with the taggers' annotation results. We see that the most common incorrect tag classes made by re-trained taggers are proper nouns. For example, 8 proper nouns are annotated differently, results in error rates of re-trained taggers on the proper noun 92%.

*D. Evaluating Word Embedding **(RQ3.)***

In this subsection, we perform three subsets of unsupervised word embedding experiments to investigate the effect of embedding features on the performance of our POS tagger. In all experiments, the effectiveness of embedding models is evaluated by the performance of the tagger when the pre-trained embedding model is used as a component in a pre-trained fashion. Except for the first experiments, others are based on CBOW with a negative sampling approach. Our CBOW is trained to predict whether or not the predicted from input context words, and the target is the same by computing their dot products. Labels are of value 0 and 1 depending on the input context words are negative or positive samples.

Experiment 1: We use the pre-trained word2vec[4] tool to extract features from unannotated security vulnerability corpus. Word2Vec is pre-trained on the Google News dataset. We give the vulnerability corpus as input, and the tool returns the word vector as output. We let the resulting word vector file be used as a feature to our POS tagger. Table VI raw2 shows that the tagger achieves an accuracy of 87%.

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

Experiment 2: We train a word embedding using well-formatted English texts (PTB) from scratch. We feed the vulnerability corpus to the pre-trained model and compute vector representations of words. The resulting word vector file is then used as a component of the POS tagger. Table VI row3 shows that the POS tagger achieve an accuracy of 90.8%.

Experiment 3: We further investigate by training the word embedding model using the merged corpus of PTB and security vulnerability text. We first construct a vocabulary from the training data and then learns the vector representation of words. We experiment with different embedding vector and context window sizes, and our model works the best with an embedding dimension size of 100 and a context window size of 3. To evaluate the quality of word embeddings, we compute the cosine similarity of the context and the target words. Our focus is to improve the embedding model to cluster the vector representations of security vulnerability words so that similar words be clustered in the same category. Similar to the previous experiments, the resulting embeddings, vector representations of the merged text corpus, are used as a component of our POS tagger. Table VI row4 shows that the POS tagger achieves an accuracy of 93.22%.

TABLE VI
THE EFFECT OF EMBEDDING FEATURE ON THE PERFORMANCE OF OUR POS TAGGER, WHEN TRAINED WITH DIFFERENT DATASETS.

Embedding	Description	Training Data	Embedding (Acc.)	Tagger (Acc)
Word2vec-toolkit	Pre-trained	Well-formatted Text	-	87%
Our Embedding	Trained	Well-formatted (PTB)	98%	90.8%
Our Embedding	Trained	Combined corpus	97.8%	**93.22%**

"word2vec−toolkit" −pre−trainedvectorstrainedonpartofGoogleNewsdataset(about100billionwords)

Table VI shows our POS tagger's achievement when the pre-trained word embedding model is used as a component. Overall, we find that training the word embedding by combining the PTB dataset and security vulnerability text can boost tagging accuracy by 6.33% and 2.53%, respectively. When word2vec embedding is used as a feature, our tagger achieves relatively lower accuracy (87%). However, using the PTB-trained word embedding, the tagger achieves 90.8% accuracy while on the combined corpus, the tagger's accuracy climbs to 93.22%. From the table, it can be inferred that word2vec pre-trained on general corpora is poorly capture domain-specific features; the POS tagger's accuracy drops to 87%. The problem of pre-trained word2vec is that, when the trained model gets something the model does not know during model training, it categorizes all in one class called 'unknown'. Those words classified into unknown are not similar to each other. As a result, their semantic meaning is lost. We also report the achievement of the unsupervised word embedding trained using the combined corpus, which provides an accuracy of 97.8%, while the loss is 0.06, in 75 epoch, shown in Fig. 8.

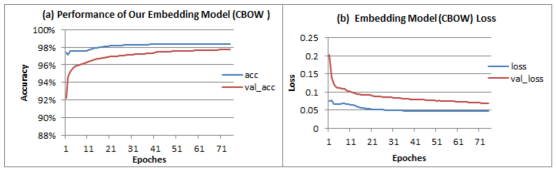

Fig. 8. Performance of our word Embedding model when trained with PTB plus security vulnerability texts, (a) is the performance, (b) is the model loss

We only measure the training accuracy and loss of the embedding model, since the training is unsupervised, the evaluation needs domain knowledge. We verify that training a domain-specific word embedding has a significant impact on the performance of POS tagging.

*E. Feature Ablation **(RQ4)***

We perform feature ablation experiments to investigate the effect of a particular feature on the accuracy of our POS tagger, as shown in Table VII.

TABLE VII
THE EFFECTS OF INDIVIDUAL FEATURES ON TAGGER'S PERFORMANCE

Features	Accuracy
Our POS tagger with all features	**93.22%**
Without punctuation POS tag dictionary/with 36 basic PTB POS tags	91%
Without Character-Based BiLSTM (Orthographic)	89.8%
Without contextual features	92.2%
Without our custom tokenizer/with Stanford Tokenizer	84.5%
Without our Word embedding/with pre-trained word2vec Embedding	87%%

As illustrated in Table VII, we ablate one feature at a time and test the resulting accuracy. Overall, our custom tokenizer and word embedding features are essential in tagging security vulnerability texts; without these, the tagging accuracy drops to 84.5% and 87.7%, respectively. Our results also show that using a character-based BiLSTM feature can boost tagging accuracy by 3.42%, which indicates that extracting features that depend on writing has a significant impact on the tagger's performance. Our tagger performs the best accuracy (93.22%) by combining all its features. Contextual and tag dictionary features have relatively less impact on the tagger's accuracy, increases by 1.02%, and 2.20%, respectively. Overall, we observe that the lack of one particular feature impacts the tagger's accuracy. Despite this, domain-specific word embedding and tokenization significantly impact the tagger's accuracy.

F. Error Analysis

Despite having the best results of the experiments, our approach generates an incorrectly annotated security vulnerability tokens, which is discussed in this section.

First, our approach processes unstructured descriptions of vulnerabilities, which are initially reported by online users (contributors). These descriptions contain mixed language studied with codes, and some are vague (see Fig. 1). Due to the noisy characteristics of online data feeds, the POS tagger sometimes gives an error result when annotating grammatically incorrect sentences, which results in an accuracy decline. We cannot avoid mistakes due to human characterizing (ungrammatical sentences) at the source data. However, we believe that we minimize these problems by using character-level and word representations as a combined feature.

Second, some tokens in the end or beginning of method names are incorrectly identified as verbs by our tagger. For example, the method names "$mostActiveCommitters.do$" and "$append()$" where "do" and "$append$" are isolated from the method names and are annotated as verbs. As we mentioned earlier, our custom tokenizer keeps only spaces between tokens, and it treats as one token otherwise. Hence, "$mostActiveCommitters.do$" and "$append()$" are treated

as one token, and annotators identified them as proper nouns. However, in some cases, the annotators give a space between the method name "$append$" and left parenthesis "(," resulting in "$append$" is incorrectly annotated as a verb; similarly, they give space before the token "do," and it is incorrectly annotated as a verb. This error indicates how the gold set examples are too sensitive. Our tagger learns from these examples. The tagger's correctness is measured on those annotated examples (gold set) that were not seen during training. To investigate how big the problem is, we write a script and run on the samples. We identified very few differences between human annotations. The problem is not a lack of knowledge by expertise; rather, manual works can not be error-free.

G. Threats to Validity

1) Internal Validity: In our approach, the first threat to internal validity may begin when SVDs are reported by online users (contributors). The natural characteristics of these online data feeds are noisy and ungrammatical. Although we automatically scrape those online data feeds, we cannot avoid mistakes due to human characterizing at the source data reporting, which may affect our approach in assigning POS tags to grammatically incorrect sentences. The popular security vulnerability advisor sources such as NVD have analysts who manually fix such errors before vulnerability reports are publicly accessible in their repository. In the future, we will extend our approach to minimize the threat related to underreporting vulnerability information from the contributors.

The other threat to internal validity is that of gold set annotation, POS tags being challenging to determine in some cases. We invite 10 participants to annotate the gold set. This document is regarded as a ground truth data to evaluate our models only if all the ten annotators agree with it. As of any hand-operated annotations, there might be some cases where the participants may not have accurately annotated the POS tag. For example, the annotators' inter agreement rate is 97.09%), indicating some errors (2.91%). We attempted to mitigate this threat by double-checking at the most challenging jargon and domain-specific vocabularies. To limit this threat, we have an annotator with expertise in security vulnerability and a second annotator with linguistics (particularly in the POS tagging task) knowledge sit together, discuss and annotate the most challenging jargon, and domain-specific vocabularies.

2) External validity: It is concerned with whether the experimental results can be generalized to our datasets. To maximize the validity to some extent, we crawl security vulnerability reports from NVD and its associated components such as CWE, CAPEC, and CVE databases. Moreover, using the universal characteristics of CVE-ID, we extract vulnerability information from cybersecurity blogs such as Twitter vulnerability web services. Although these vulnerability reports contain 8,622,302 tokens, there are other security vulnerability advisory sources such as OSVDB that keep track of vulnerability information that is not included in this study. Still, we believed that the general structure of vulnerability descriptions and reporting formats are similar due to the universal characteristics of CVE-ID. However, there may be insignificant differences in characterizations of security vulnerabilities in other advisory sources. Accordingly, the results can be generalized to vulnerability reports in this domain.

VI. Related Work

POS tagging in the security vulnerability domain has not been investigated. However, in the software engineering domain, recently, there has been growing concern in developing POS tagging tools, including software-specific POS tagger for Stack Overflow [20], heuristics-based POS tagging of source code [18], [21], [22], [47], and machine learning approach [3]. On the other hand, POS taggers are built for software documentation and bug reports. For example, [48], [49] study the POS tagging of bug reports. Tian et al. [50] compared POS tagging techniques on bug reports and show that the news-trained taggers' performance drop on bug reports. More recently, Pârtachi et al. [51] devise an approach called, POST-to do POS tagging on text which contains both natural language and source code. Compared to StORMeD [52], POSIT, which is based on BiLSTM neural network, advances the state-of-the-art on mixed text tagging. However, except POST, those works are based on heuristics and supervised learning techniques, which require much expensive human effort. In contrast, our POS tagger processes SVD rather than source code, Stack Overflow, and bug reports. We apply state-of-the-art deep learning approaches instead of traditional machine learning and hand-crafted rule-based approach. Comparing to those works, we provide a POS tagger designed for processing security vulnerability information from online data sources.

VII. Conclusion

In this paper, we present a fine-tuning approach that trains a source model on a large newswire corpus and then uses the learned knowledge as prior knowledge for training the target model on a small amount of security vulnerability data. Our evaluation shows that our tagger correctly identifies a broad set of security vulnerability tokens and natural language contents, which are extremely useful to downstream applications such as security systems, and support cybersecurity analysts and developers to gain accurate vulnerability information as quickly as possible. We believe that the findings from this work, including the published resources, such as POS annotated corpus (∼8M tokens), web scraper tool, and trained models will be at the forefront of other future studies, which will add insight in this domain and allow for automated processing and enhanced research. In the future, we plan to introduce new POS tags and extend our work for POS tagging of software engineering discussion forums, which can support many downstream applications.

Acknowledgments

This work is supported by the National Key Research and Development Program of China (2017YFB1401200), the National Natural Science Foundation of China (NSFC) (61832014), and the Open Project Funding Project of State Key Laboratory of Communication Content Cognition (A32002).

REFERENCES

[1] R. A. Bridges, K. M. T. Huffer, C. L. Jones, M. D. Iannacone, and J. R. Goodall, "Cybersecurity automated information extraction techniques: Drawbacks of current methods, and enhanced extractors," in *16th IEEE International Conference on Machine Learning and Applications, ICMLA 2017, Cancun, Mexico, Dec.*, 2017, pp. 437–442.

[2] A. Joshi, R. Lal, T. Finin, and A. Joshi, "Extracting cybersecurity related linked data from text," in *2013 IEEE Seventh International Conference on Semantic Computing, Irvine, CA, USA, Sept.*, 2013, pp. 252–259.

[3] S. S. Weerawardhana, S. Mukherjee, I. Ray, and A. E. Howe, "Automated extraction of vulnerability information for home computer security," in *Foundations and Practice of Security - 7th International Symposium, FPS 2014, Montreal, QC, Canada, Nov.*, 2014, pp. 356–366.

[4] H. Gasmi, J. Laval, and A. Bouras, "Information extraction of cybersecurity concepts: An lstm approach," *Applied Sciences*, vol. 9, no. 19, 2019.

[5] T. D. Günes, L. Tran-Thanh, and T. J. Norman, "Identifying vulnerabilities in trust and reputation systems," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, Aug.*, 2019, pp. 308–314.

[6] H. Chen, J. Liu, R. Liu, N. Park, and V. S. Subrahmanian, "VEST: A system for vulnerability exploit scoring & timing," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, Aug*, 2019, pp. 6503–6505.

[7] S. Mittal, P. K. Das, V. Mulwad, A. Joshi, and T. Finin, "Cybertwitter: Using twitter to generate alerts for cybersecurity threats and vulnerabilities," in *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2016, San Francisco, CA, USA, Aug.*, 2016, pp. 860–867.

[8] Y. Zhou, Y. Tong, T. Chen, and J. Han, "Augmenting bug localization with part-of-speech and invocation," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 27, no. 6, pp. 925–950, 2017.

[9] P. M. Vu, T. T. Nguyen, and T. T. Nguyen, "Alpaca: Advanced linguistic pattern and concept analysis framework for software engineering corpora," in *Proceedings of the 2019 ACM Southeast Conference*, 2019, pp. 249–252.

[10] S. Bird, "NLTK: the natural language toolkit," in *21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Sydney, Australia, Jul.*, 2006, pp. 69–72.

[11] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*, 2009.

[12] C. D. Manning, "Part-of-speech tagging from 97% to 100%: Is it time for some linguistics?" in *Computational Linguistics and Intelligent Text Processing - 12th International Conference, CICLing 2011, Tokyo, Japan, Feb.*, 2011, pp. 171–189.

[13] D. Q. Nguyen and K. Verspoor, "An improved neural network model for joint POS tagging and dependency parsing," in *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, 2018, pp. 81–91.

[14] A. Garimella, C. Banea, E. H. Hovy, and R. Mihalcea, "Women's syntactic resilience and men's grammatical luck: Gender-bias in part-of-speech tagging and dependency parsing," in *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL, Florence, Italy, Jul. 28- Aug. 2*, 2019, pp. 3493–3498.

[15] R. A. Bridges, C. L. Jones, M. D. Iannacone, and J. R. Goodall, "Automatic labeling for entity extraction in cyber security," *CoRR*, vol. abs/1308.4941, 2013.

[16] S. More, M. Matthews, A. Joshi, and T. Finin, "A knowledge-based approach to intrusion detection modeling," in *2012 IEEE Symposium on Security and Privacy Workshops, San Francisco, CA, USA, May*, 2012, pp. 75–81.

[17] X. Liao, K. Yuan, X. Wang, Z. Li, L. Xing, and R. A. Beyah, "Acing the IOC game: Toward automatic discovery and analysis of open-source cyber threat intelligence," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, Oct.*, 2016, pp. 755–766.

[18] W. Olney, E. Hill, C. Thurber, and B. Lemma, "Part of speech tagging java method names," in *2016 IEEE International Conference on Software Maintenance and Evolution, ICSME, Raleigh, NC, USA, Oct.*, 2016, pp. 483–487.

[19] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of english: The penn treebank," *Comput. Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[20] D. Ye, Z. Xing, J. Li, and N. Kapre, "Software-specific part-of-speech tagging: an experimental study on stack overflow," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, Apr.* ACM, 2016, pp. 1378–1385.

[21] S. Gupta, S. Malik, L. L. Pollock, and K. Vijay-Shanker, "Part-of-speech tagging of program identifiers for improved text-based software engineering tools," in *IEEE 21st International Conference on Program Comprehension, ICPC, San Francisco, CA, USA, 20-21 May*, 2013, pp. 3–12.

[22] R. S. Alsuhaibani, C. D. Newman, M. L. Collard, and J. I. Maletic, "Heuristic-based part-of-speech tagging of source code identifiers and comments," in *5th IEEE Workshop on Mining Unstructured Data MUD@ICSME, Bremen, Germany, Sep.*, 2015, pp. 1–6.

[23] S. Meftah and N. Semmar, "A neural network model for part-of-speech tagging of social media texts," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May*, 2018.

[24] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, Jun., Volume 1*, 2019, pp. 4171–4186.

[25] X. Han and J. Eisenstein, "Unsupervised domain adaptation of contextualized embeddings for sequence labeling," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, Nov.*, 2019, pp. 4237–4247.

[26] J. R. Landis and G. G. Koch, "An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers," *Biometrics*, pp. 363–374, 1977.

[27] D. Ye, Z. Xing, C. Y. Foo, Z. Q. Ang, J. Li, and N. Kapre, "Software-specific named entity recognition in software engineering social content," in *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, Mar. Volume 1*, 2016, pp. 90–101.

[28] A. Roy, Y. Park, and S. Pan, "Learning domain-specific word embeddings from sparse cybersecurity texts," *CoRR*, vol. abs/1709.07470, 2017.

[29] A. L. Beam, B. Kompa, I. Fried, N. P. Palmer, X. Shi, T. Cai, and I. S. Kohane, "Clinical concept embeddings learned from massive sources of medical data," *CoRR*, vol. abs/1804.01486, 2018.

[30] P. K. Sarma, Y. Liang, and B. Sethares, "Domain adapted word embeddings for improved sentiment classification," in *Proceedings of the Workshop on Deep Learning Approaches for Low-Resource NLP, DeepLo@ACL, Melbourne, Australia, Jul.*, 2018, pp. 51–59.

[31] 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 26: 27th Annual Conference on Neural Information Processing Systems 2013, Lake Tahoe, Nevada, United States, Dec.*, 2013, pp. 3111–3119.

[32] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, Doha, Qatar, a Special Interest Group of the ACL, Oct.*, 2014, pp. 1532–1543.

[33] B. Wang, A. Wang, F. Chen, Y. Wang, and C. J. Kuo, "Evaluating word embedding models: Methods and experimental results," *CoRR*, vol. abs/1901.09785, 2019.

[34] O. Levy, Y. Goldberg, and I. Dagan, "Improving distributional similarity with lessons learned from word embeddings," *Trans. Assoc. Comput. Linguistics*, vol. 3, pp. 211–225, 2015.

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

[36] X. Ma and E. H. Hovy, "End-to-end sequence labeling via bi-directional lstm-cnns-crf," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, Aug. Berlin, Germany, Volume 1*, 2016.

[37] C. Dyer, M. Ballesteros, W. Ling, A. Matthews, and N. A. Smith, "Transition-based dependency parsing with stack long short-term memory," in *Proceedings of the 53rd Annual Meeting of the Association for*

Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, Jul., Beijing, China, Volume 1, 2015, pp. 334–343.

[38] T. Gui, Q. Zhang, H. Huang, M. Peng, and X. Huang, "Part-of-speech tagging for twitter with adversarial neural networks," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, Sep.*, 2017, pp. 2411–2420.

[39] Q. Huang, L. Deng, D. O. Wu, C. Liu, and X. He, "Attentive tensor product learning," in *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, Jan.*, 2019, pp. 1344–1351.

[40] B. Plank, A. Søgaard, and Y. Goldberg, "Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, Berlin, Germany, Volume 2*, 2016.

[41] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, Jun.*, 2014, pp. 647–655.

[42] M. Long, Y. Cao, J. Wang, and M. I. Jordan, "Learning transferable features with deep adaptation networks," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, Jul.*, 2015, pp. 97–105.

[43] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, and J. Dean, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, Nov.*, 2016, pp. 265–283.

[44] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://keras.io

[45] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *19th International Conference on Computational Statistics, COMPSTAT 2010, Paris, France, Aug.*, 2010, pp. 177–186.

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May*, 2015.

[47] C. D. Newman, R. S. Alsuhaibani, M. L. Collard, and J. I. Maletic, "Lexical categories for source code identifiers," in *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24*, 2017, pp. 228–239.

[48] F. N. A. A. Omran and C. Treude, "Choosing an NLP library for analyzing software documentation: a systematic literature review and a series of experiments," in *Proceedings of the 14th International Conference on Mining Software Repositories, MSR, Buenos Aires, Argentina, May*, J. M. González-Barahona, A. Hindle, and L. Tan, Eds. IEEE Computer Society, 2017, pp. 187–197.

[49] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation," in *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR, San Francisco, CA, USA, May*, 2013, pp. 2–11.

[50] Y. Tian and D. Lo, "A comparative study on the effectiveness of part-of-speech tagging techniques on bug reports," in *22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER, Montreal, QC, Canada, Mar.*, 2015, pp. 570–574.

[51] P. Pârtachi, S. K. Dash, C. Treude, and E. T. Barr, "POSIT: simultaneously tagging natural and programming languages," in *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July*, 2020, pp. 1348–1358.

[52] L. Ponzanelli, A. Mocci, and M. Lanza, "Stormed: Stack overflow ready made data," in *12th IEEE/ACM Working Conference on Mining Software Repositories, MSR 2015, Florence, Italy, May 16-17*, 2015, pp. 474–477.