Automated Extraction of Software Names from Vulnerability Reports using LSTM and Expert System

Igor Khokhlov*, Ahmet Okutan†, Ryan Bryla†, Steven Simmons†, Mehdi Mirakhorli†
*Sacred Heart University**, *Rochester Institute of Technology*†
khokhlovi@sacredheart.edu*, {axoeec, rmb7605, sds9278, mxmvse}@rit.edu†

***Abstract*—Software vulnerabilities are closely monitored by the security community to timely address the security and privacy issues in software systems. Before a vulnerability is published by vulnerability management systems, it needs to be characterized to highlight its unique attributes, including affected software products and versions, to help security professionals prioritize their patches. Associating product names and versions with disclosed vulnerabilities may require a labor-intensive process that may delay their publication and fix, and thereby give attackers more time to exploit them. This work proposes a machine learning method to extract software product names and versions from unstructured CVE descriptions automatically. It uses Word2Vec and Char2Vec models to create context-aware features from CVE descriptions and uses these features to train a Named Entity Recognition (NER) model using bidirectional Long short-term memory (LSTM) networks. Based on the attributes of the product names and versions in previously published CVE descriptions, we created a set of Expert System (ES) rules to refine the predictions of the NER model and improve the performance of the developed method. Experiment results on real-life CVE examples indicate that using the trained NER model and the set of ES rules, software names and versions in unstructured CVE descriptions could be identified with F-Measure values above 0.95.**

***Index Terms*—Common Product Enumeration, Common Vulnerability and Exposures, Natural Language Processing, Software Product Name Extraction, Software Vulnerability**

I. Introduction

Software vulnerabilities threaten the confidentiality, integrity, and availability of software systems. They are the culprit behind a broad range of modern cyber-attacks that may lead to code execution, memory or file system access, malicious program installation, confidential data exposure, destruction or modification of sensitive data *etc.* According to recent studies, around one-third of the security professionals state that their organization had been breached as a result of an unpatched software vulnerability [1]. With a blooming black market where exploits are sold to adversaries, detecting and fixing vulnerabilities is a very time-sensitive task. Not fixing software vulnerabilities on time could entirely shut down software systems and/or lead to severe financial consequences for private and public businesses and organizations.

This research is sponsored by the U.S. Department of Homeland Security (DHS) by award # 70RSAT19CB0000020. The published material represents the position of the author(s) and not necessarily that of DHS.

There are many private and public vulnerability management systems (VMS) that provide information about software vulnerabilities, such as their descriptions, disclosure dates, affected software product names and versions, root causes, and consequences. Widely known examples of these systems are the National Vulnerability Database (NVD) [2], VulnDB [3], IBM X-Force [4], VulnCode-DB [5], Exploit DB [6], and Security Focus [7]. Vulnerability reports in VMS associate a set of software product names and versions to each vulnerability to help security professionals understand which software products are affected by the weaknesses identified by a Common Vulnerability and Exposure (CVE). However, prior works show that inconsistent software versions are highly prevalent in the CVEs [8]. Furthermore, extracting the affected software names and versions from a disclosed vulnerability may require a labor-intensive process which may delay the publication of a vulnerability and thereby give attackers more time to exploit it. Just-in-time AI/ML systems are needed to identify affected software names and versions in unstructured CVE descriptions to help shorten the publication time of CVEs and decrease the possibility of an exploit. Considering that there are thousands of CVE Numbering Authority (CNA) web pages, it is extremely hard and time-consuming to extract product names and versions from these highly unstructured pages incorporating a variety of web technologies or frameworks.

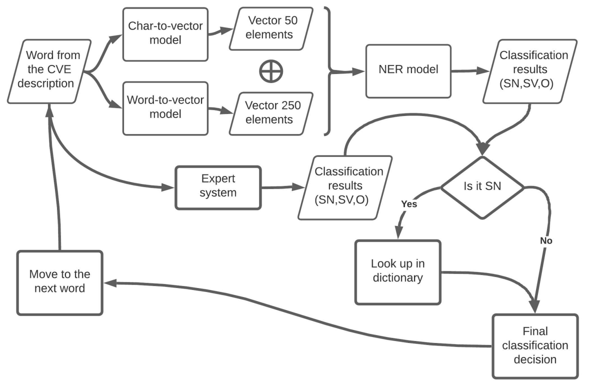

Fig. 1. The training and classification processes used in the paper. CVE description is processed by both developed NER model and developed ES.

This paper presents our customized Named Entity Recognition (NER) model that is trained on the refined training dataset published earlier [8]. The developed NER model is capable of extracting the software names and versions from an unstructured CVE description. Specifically, it uses a bidirectional LSTM model that consumes context-aware feature vectors produced by custom Word-to-Vector (Word2Vec) and Char-to-Vector (Char2Vec) encoders. The LSTM model is used to associate each word in a given CVE description to one of the three classes, *i.e.*, Software Name (SN), Software Version (SV), and Other (O). The Common Product Enumeration (CPE) [9] catalog is used to verify the terms marked as software names. Similarly, based on all available CVEs published before, a set of Expert System (ES) rules are created and those rules are used to improve the software version identification process. Figure 1 shows the architecture of the integrated processes used to extract software names and versions from CVE descriptions. The main contributions of this paper are:

- Natural Language Processing (NLP) methods are leveraged to train Word2Vec and Char2Vec models on CVE data, and these models are used to create context-aware feature vectors that represent the attributes of the terms in CVE descriptions.
- This work uses the CPE dictionary to refine the terms identified as software names. It creates a set of ES rules derived from the relationships of the software names and versions in previously published CVEs. It is shown that the use of these ES rules and the CPE dictionary together improves the overall accuracy of the trained NER model to extract software names and versions from unstructured CVE descriptions.

The refined dataset are going to be published as well.

The rest of the paper is organized as follows: Section II reviews prior works about existing vulnerability platforms and software product name extraction. Section III describes the research methodology used to extract affected products and versions and explains designed experiments. Section IV presents the results for different experiments and Section V provides concluding remarks.

II. Related Work

There have been several previous attempts to obtain valuable information from text-based vulnerability descriptions via NLP. Researchers employed NLP for the analysis of mobile application descriptions to verify requested permissions legitimacy [10]–[14]. Cyber-security professionals and researchers also use NLP-based approaches for software defects analysis based on various sources, for example, CVE itself [15]–[19]. Bozorgi *et al*. [20] extracted information from a set of 4000 CVE descriptions using a bag-of-words approach and trained linear SVM classifiers on the resulting feature vectors. Khazaei *et al*. [21] proposed a method for calculating CVSS scores by training machine learning classifiers on vulnerability descriptions.

Shen and Stringhini [22] realized that current studies of CVEs being exploited in isolation [23], [24], [25] did not provide insight into how these exploits were used as part of more complex cyberattacks, as access to the victim's system is often the first step in order to accomplish the attackers goal [26], [27]. The authors developed a system, ATTACK2VEC, in order to identify and track the emergence of exploits over time and tested it on a set of 8,087 unique security events obtained from a commercial Intrusion Prevention System (IPS) gathered over a period of two years. ATTACK2VEC uses pointwise mutual information (PMI), a popular technique for word associations, to calculate the relationship between security events over a given window of time, with a higher relationship denoting a greater likelihood of co-occurrence. These events are then encoded into a lower-dimensional space, where various metrics are able to be used to detect changes, identify trends, and monitor how CVEs are exploited in real settings. In a case study, an exploit was identified as a culprit recent Equifax data breach (CVE-2017-5638).

A common method to extract information from the text that is utilized in our approach as well is to use a Name Entity Recognition (NER) model, which can be trained to recognize product names and versions given an appropriate data set. Word2Vec is a word embedding algorithms technique developed by Mikolov *et al*. [28] that calculates the similarity of two words within the same document. Jang *et al*. [29] explored the performance of word2vec using CNNs (Convolution Neural Networks) to classify a collected set of 122,258 news articles and 291,301 tweets, as they typically have factors that diminish the effective learning (i.e., use of alternative words, advertisements). They achieved a high F1-score (0.9351 for news articles and 0.9097 for tweets) using word2vec algorithms alongside CNNs compared to a random vector (0.8409 for news articles and 0.836 for tweets). Ye *et al*. [30] attempted an ML-based approach for identifying software-specific entities such as frameworks, API, and programming languages. Applying their method, S-NER, over a set of Stack Overflow posts, they achieved a high F1-score (93.358) identifying programming languages and had scores over 0.7 for several other categories (i.e., APIs, libraries) compared to a baseline method.

Wu *et al*. [31] use three separate models then blend the results to determine which words of a sentence are product names. The first is a "Standard Matching Model", which simply uses the product names from the training data and tries to find them in sentences from the test data. The second model analyzes the semantics of the sentence and determines which words are "Special Words" by checking if they have a specific combination of characters, for example, multiple uppercase letters at different places, such as in "ThinkPad". The third model is a "Conditional Random Field" model (CRF), which breaks a sentence into individual words where each is classified as either "B", the beginning of the product name, "I" inside the product name, or "O" for others. This model utilizes a set of features such as length of the word, number of uppercase letters, number or dash-characters, etc., in order to predict the product name. Created models are then blended together to identify product names that may not have been found individually.

Bi-directional LSTM [32], [33] scans sentences in two directions: forward and backward, allowing it to learn word relations better. Waareus *et al*. [34] convert the words of the sentence into vector representations to capture the words semantic use. At the same time, the model extracts character embeddings with a CNN, word case (capitalized, uppercase, lowercase) features and is compared to a security lexicon consisting of CVEs from the NVD database. These features are concatenated and passed to an RNN to extract the name and version from the sentence. The RNN consists of a bi-directional LSTM layer, which passes its output to a CRF layer. These combined prevent the model from having exploding or vanishing gradient, while the CRF layer classifies the words as either "B", "I", or "O" similar to the study by Wu *et al*. [31]. During our experiments, the feature vectors produced by the Word2Vec model are fed to a bi-directional Long short-term memory (LSTM) network, generating vectors of confidence levels for each word in the vulnerability description.

Our approach partially follows a recent paper by Dong *et al*. [35] which identifies product names and versions within vulnerability descriptions using Deep Learning (DL). However, our method augments a DL model with ES, and that helps to identify additional software product names and versions within CVE description that were not detected by a pure DL-based model. Dong *et al*. [35] developed a system, VIEM, to determine the inconsistencies between the vulnerability description in the National Vulnerability Database (NVD) and unstructured CVE descriptions. They used a NER model along with Relation Extraction (RE) to match product versions and names. The NER model tags words or groups of words as "Software Name" (SN), "Software Version," (SV) or "Other" (O). Based on the position in the sentence, the RE model determines which version groups should be matched with which name groups. The model achieved very high precision and recall on a set of 70,569 vulnerability reports and revealed that only 59.82% of the vulnerability reports matched their NVD entries, with inconsistencies growing over time. Additionally, case studies confirmed that the affected software versions were often over or under-claimed. While authors had 13 CVE types in their training data, they chose the "memory corruption" type to train their NER model since it contained the largest number of CVEs. Moreover, they claimed that training the NER model on all data does not affect or even decrease the overall NER performance. Because our approach is partially based on this work, we studied it in more detail and discovered that while the claim is true for the SN, in fact training on all data improves the performance of the SV extraction. In addition, even though the trained NER model already achieved high performance in the SN and SV extraction, additional methods we used in this paper increases NER model's performance even further. Leveraging our expertise in the cybersecurity and software security domains, we developed an ES that extracts additional SN and SV instances from CVE descriptions. We increased the SV extraction performance of the NER model that is trained on all data, without sacrificing its SN extraction performance. More details about the developed methodology and the conducted empirical study are provided in Sections III and IV.

III. Methodology

In order to extract software names and versions from the CVE descriptions, we utilize two major models: NER model [36], [37] and ES model [38]. NER model takes advantage of the machine learning techniques and is able to classify each word within the description as "software name" (SN), "software version" (SV), or "other" (O). The model develops its ability to classify description words by observing a large amount of labeled (ground truth) training data. The benefit of this model is that it does not require expertise in the software-security domain. However, it is not easy to create a formal verification of the model due to model complexity. ML models are also susceptible to various types of attacks, such as data backdoors or adversarial attacks, while the ES model leverages our knowledge and expertise in the cyber-security field. For this model, we create a set of rules that are able to find SN and SV within a CVE description. In tandem, these models are able to extract potential candidates of SN and SV. The overall architecture of the SN and SV extraction process is presented in Figure 1. Finally, we employ the CPE dictionary [9] provided by the National Institute of Standards and Technology to verify all predicted software product names and versions.

A. NER Model Architecture

The NER model used in this paper consists of three models. Figure 1 shows the overall process flow for training and classification, and Figure 2 provides the interaction and detailed architecture of the models used:

1) Word to vector (word-to-vector) model on a word level (green color in Figure 2);
2) Word to vector model on a character level (char-to-vector) (yellow color in Figure 2);
3) Bi-directional long short term memory (LSTM) model (NER model itself) (white color in Figure 2);

For word embedding, our NER model employs the standard approach presented by Mikolov *et al*. [28] and encodes each word into a one-dimensional vector. To train the word-to-vector model, we used training corpus data from [35] and the DeepLearning4J framework [39]. The corpus data contains 252,851 sentences with 11,221,324 words. The trained model size is 551,150,529 bytes and embeds each word into a vector of 250 real numbers from -1 to 1. This word embedding into the vector allows us to convert words of different lengths into a vector of fixed length and use it as input for the deep learning model. This model makes sure that words that often appear close to each other in the CVE descriptions are close to each other in the feature hyper-space as well. The model returns an empty vector if it does not have the word in its dictionary. In

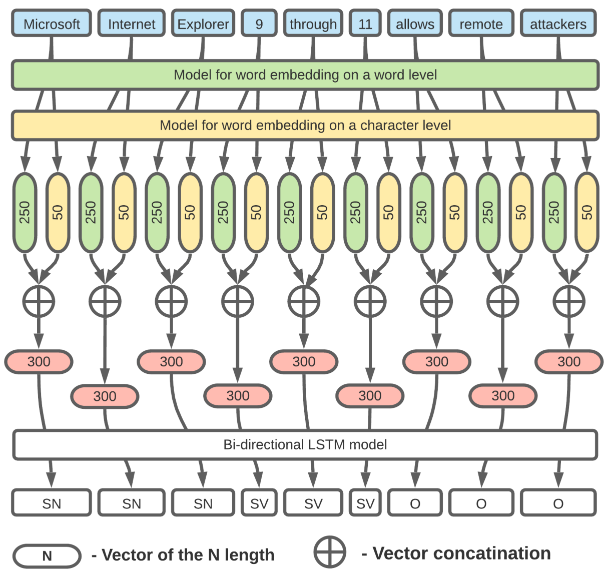
Microsoft
Internet
Explorer
9
through
11
allows
remote
attackers
Model for word embedding on a word level
Model for word embedding on a character level
250
50
300
Bi-directional LSTM model
SN
SV
O
N - Vector of the N length
- Vector concatination

Fig. 2. Detailed view of the developed NER model. Blue shaded rectangles represent the words in CVE description, green shaded ovals are vectors produced by word embedding, yellow shaded ovals are product of the word embedding on the character level, and white rectangles are labels assigned by the bi-directional LSTM model.

this case, the returned vector is filled with randomly generated values.

To solve this issue with "unknown" words, we employ the character to vector model, which embeds word into a vector of fixed length (50 elements in our case) on the character level [40]–[42]. This model does not rely on a dictionary but rather identifies semantic similarities between words. This allows converting even those words that were misspelled. The vectors of similarly spelled words are positioned close to each other in the hyper-space.

Word and character level vectors are concatenated and fed into the bi-directional LSTM. The input layer's size is equal to the size of concatenated vectors obtained from word-to-vector and char-to-vector models. The benefit of using the LSTM model [43] is that it can grasp the structure of the word itself as well as learn the relations between words within a sentence. Bi-directional LSTM [32], [33] scans sentences in two directions: forward and backward, which allows it to learn word relations even better. The trained model generates a vector of confidence levels for each word in the description that is fed into the model.

B. Expert System Model and Rules

To further improve product extraction performance from the CVE description, we have developed a set of rules that are also capable of extracting SN and SV. Some of the rules consider output from the NER model, and some use NLP techniques, such as taking into account part-of-speech (PoS).

To identify PoS, we employ Apache OpenNLP library [44]. Our analysis of several random CVE descriptions indicates that SN belongs to proper nouns PoS.

TABLE I
SOFTWARE NAMES (SN) DISTRIBUTION WITHIN A SENTENCE

Distance in words	%, occurs	Distance in words	%, occurs
[1-3]	28.55	[31-50]	6.42
[4-10]	33.02	[51-75]	1.62
[11-20]	19.52	[76-100]	0.45
[21-30]	9.80	[101-200]	0.62

TABLE II
SOFTWARE VERSIONS (SV) DISTANCES FROM THE RELEVANT SOFTWARE NAME (SN)

Distance in words	%, occurs	Distance in words	%, occurs
[1-4]	72.85	[31-50]	0.33
[5-10]	16.93	[51-75]	0.15
[11-20]	6.63	[76-100]	0.03
[21-30]	1.09	[101-200]	0

To develop SN and SV extraction rules, we have examined the whole dataset provided by [35] and analyzed 25,978 SN instances and 36,292 SV instances. We have found that 61.5% of all SN in the dataset occurs within the first ten words in the sentence and almost 91% of all SN lies within the first 30 words of the sentence. More details of this analysis are presented in Table I.

A similar analysis shows that only 0.67% of all SV occurs before SN. One of the developed rules exploits the proximity of potential SV to the already mentioned SN. Our analysis revealed that almost 73% of all SV lies no further than five words from the related SN, and almost 90% within ten words proximity. We provide more details on this in the Table II. Figure 3 provides details on distances for SN and SV.

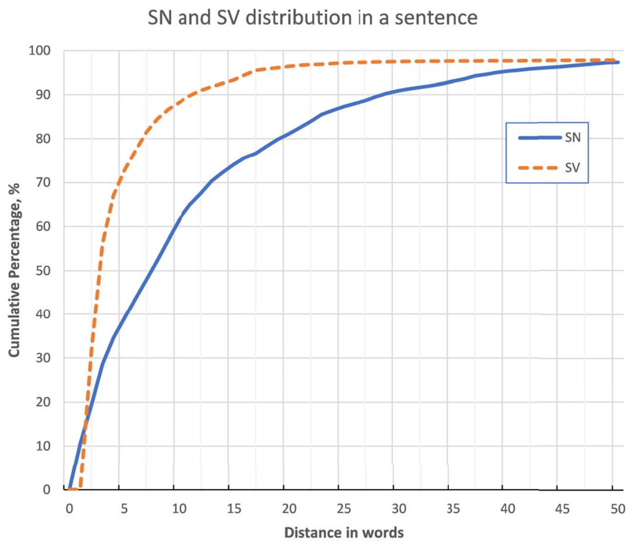

Fig. 3. Distribution of SN and SV in a sentence. The Blue solid line represents the distance (in words) of SN from the sentence's beginning, and the orange dashed line represents a distance (in words) of SV from the relevant SN instance.

To improve NER model performance, we developed a set of rules that are based on our expertise in the software security domain as well as on the analysis of only training datasets to avoid bias. We developed separate rules for the SN and SV extraction.

C. Software Name Extraction Rules

First of all, we classify each word of the sentence as PoS. Our random analysis of CVE descriptions analysis shows that SNs most likely are identified as a Proper Nouns (NNP) PoS. In addition, we take into account Noun (NN) PoS if it is just after the "in" preposition. We also take into account articles "a", "an", and "the", and consider distance (in words) of the potential SN from the sentence beginning. Finally, we check all potential SN instances in the CPE dictionary. Our ES classifies a word as an SN if at least one of the following rules is true:

1) The word is classified as NNP and within 40 words range from the sentence beginning and does not belong to an article and is in the CPE dictionary. This rule is based on the training dataset analysis (see Table I).
2) The word is between two SNs.

To improve the precision of SN extraction, we use verification in the CPE dictionary. The suspected word is fed into our CPE lookup algorithm, which checks if this word is mentioned in CPE and forms a list of most likely correct entries. The algorithm repeatedly filters down the list of products in the CPE dictionary based on whether it contains each word in the name extracted from the sentence. If there is a version found in the name, it will also filter the list based on that, which significantly narrows down the possible products that it matches with. Then, we check if the words before and after the suspected word are also in the CPE result list (see Figure 4). All words that match CPE records are marked as SN. This approach allows us to avoid marking as SN those words that can be in many product names but are not SN on their own (e.g., "service", "project", etc.). In addition, all words that were classified by the NER model as SN with a confidence level less than 0.9 are also checked in the CPE dictionary. This threshold is chosen based on the performed empirical study. A higher threshold level will trigger the CPE lookup dictionary more frequently than needed, and a lower threshold may lead to increased numbers of False-positive SN results.

D. Software Version Extraction Rules

SV extraction happens after all SNs were identified since our algorithm takes into account that more than 99% of all SV occur after SN (based on the training dataset analysis). ES can classify the word as an SV only if SN is already in the sentence. We also analyzed CVE descriptions and developed a short yet effective trigger words list that contains the following words: "before","earlier","between","after", "through", "¿", "=", "¡", "version", "versions", "newer","older". Word is considered as an SV if at least one of the following rules is true:

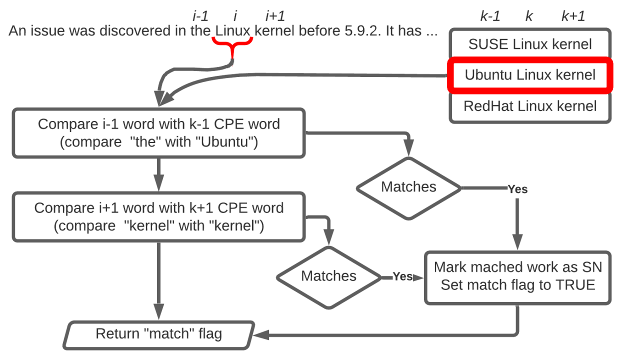
i-1
i
i+1
An issue was discovered in the Linux kernel before 5.9.2. It has ...
k-1
k
k+1
SUSE Linux kernel
Ubuntu Linux kernel
RedHat Linux kernel
Compare i-1 word with k-1 CPE word (compare "the" with "Ubuntu")
Matches
Yes
Compare i+1 word with k+1 CPE word (compare "kernel" with "kernel")
Matches
Yes
Mark mached work as SN Set match flag to TRUE
Return "match" flag

Fig. 4. Checking a suspected word in the CPE dictionary with reduced False-Positive results

1) It contains digits and is not further than 30 words from the last SN. This rule is based on the training dataset analysis (see Table II).
2) It is in the list of trigger words and is not further than 30 words from the last SN.
3) It contains digits, and the previous word is classified as an SV.
4) This word is "and" or "or" and is between two SVs.

In addition, all words that are marked as SV by the NER model with a confidence level of less than 0.8 are checked by the developed ES. The threshold maintains a balance between the SV extraction precision and recall. The lower threshold decreases the SV extraction precision, and the higher threshold level decreases the SV extraction recall without a significant improvement in the precision. Both SN and SV confidence thresholds were chosen using an empirical approach after a series of tests on the training dataset.

E. Matching SN with SV

There are various ways to match a software name with a software version. For example, Dong *et al*. [35] uses a neural network model similar to the NER model that is trained on a bunch of data. That model considers all pairs of SN and SV that were identified by the NER model. While this approach does not require expert knowledge in the security domain, it has several limitations. First of all, this approach allows matching only one SV word with one SN word. The authors solved this problem by connecting SV, and SN-related words with the underscore ("_") symbol. Here is the example from the training dataset: "Cobham_Sea_Tel 121_build_222701 devices allow remote ...". In this example, SN is represented by "Cobham", "Sea", and "Tel", while words "121", "build", and "222701" belong to SV class. As one can see, SN and SV words are connected with underscore symbols in such a way to make one SN and one SV words. However, this creates another issue: the NER model was trained on SN and SV words that were not connected. In addition, in real-life cases, these words would not be connected. Also, in the CVE description, very often, multiple SV are listed. Finally, since the relational model is a deep learning model, its employment is computationally expensive, which may be a problem if the system considers a dozen thousands of CVEs each day.

In our approach to matching the software name with the software version, we apply the ES. Our analysis of CVE description shows that SV words that follow an SN instance belong to that SN in almost 99.5% cases. The developed ES scans descriptions for SN and SV instances and connects all SVs with the SN until the new SN instance is met in the description. As a result, our system produces "Product" objects that contain software names and may contain multiple software versions related to that SN.

IV. Empirical Study

To evaluate experiment results, we measured the overall accuracy of each model as well as the precision, recall, and F1-score of each class separately. It is worth noticing that while both precision and recall metrics are important, recall metric plays a more important role in cybersecurity. Precision metric in classification tasks presents the number of correctly predicted class instances out of all predicted instances of that particular class, in other words, it reflects a false-positive rate, as a lower precision metric implies a larger number of false positives. Recall, on the other hand, presents the number of correctly predicted class instances out of the number of actual instances of that particular class, in other words, it reflects the true-positive rate. A lower recall metric indicates that the model misses more instances of a class. In the case of SN and SV extraction from the CVE descriptions, it is impossible to reach 100% in all performance metrics such as accuracy, recall, precision, etc., and the model tuning very often results in improving one metric over others. In the SN and SV extraction, it is important to miss as less SN and SV as possible, which means we have to improve the *recall* metric without a significant decrease of *precision* metric. According to our experiment results, we achieved the goal, and for SN extraction, we improved both recall and precision, while for SV extraction, we improved recall by sacrificing 2% of SV extraction precision. In addition, the application of the developed ES allowed us to reveal massive errors in the test dataset. More details about NER model are provided in section IV-A, ways to improve NER performance are provided in section IV-B, SN and SV matching algorithm is presented in section IV-C, current state of CVE analysis is presented in section IV-D, and experiment results analysis is presented in section IV-E. We also present confusion matrices for the result of our experiments in section IV-A.

A. Training NER model

To train NER model we tested two approaches: at first we have trained the developed NER model only at the "Memory Corruption" CVE type data, similarly to [35]; then we trained model on all thirteen types of CVEs: "HTTP Response Split", "CSRF", "File Inclusion", "Directory Traversal", "Gain Privileges", "Memory Corruption", "Bypass", "Gain Information", "SQL Injection", "XSS", "Overflow", "Denial of Service", and "Code Execution". All datasets were divided into two parts: training and testing. "Memory corruption" data has 6,922 sentences (226,296 words) for training purpose and 1,002 sentences (33,518 words) for testing. For all data types, we used 13,947 sentences (465,873) words for the training process, and 7,698 sentences (214,019 words) for the testing. On average, we have used 2/3 of data for training and rest 1/3 for the testing. This is the conservative approach in the case when there is a sufficient amount of data.

TABLE III
Confusion matrix for the model that is trained on "Memory Corruption" data only

		Predicted		
		SN	SV	O
Actual	SN	1816	5	546
	SV	6	2498	175
	O	26	66	28357

TABLE IV
Confusion matrix for the model that is trained on all training data (all CVE types)

		Predicted		
		SN	SV	O
Actual	SN	1836	13	518
	SV	12	2513	154
	O	44	49	28356

To test both models, we used the "Memory Corruption" CVE type test dataset and proved that using all data improves results for SN and SV detection in the "Memory Corruption" CVE descriptions. Confusion matrix for both models are presented in Table III and Table IV, performance metrics are in the Table V and Table VI.

As one can see from the presented results, the NER model trained on all available training data has a higher recall for both SN and SV extraction, while the precision of SN extraction is lower. This lower precision is explained by the fact that the NER model trained only on "memory corruption" data simply detects less SN in total, which can be fixed by adding ES into the model. Therefore, for the real-life application NER model

TABLE V
Performance metrics for the model that is trained on "Memory Corruption" data only. Overall accuracy = 0.975

	Precision	Recall	F1-score
SN	0.983	0.767	0.862
SV	0.972	0.932	0.952
O	0.975	0.997	0.986
Average	0.977	0.9899	0.933

TABLE VI
Performance metrics for the model that is trained on all training data (all CVE types). Overall accuracy = 0.976

	Precision	Recall	F1-score
SN	0.970	0.776	0.862
SV	0.976	0.938	0.957
O	0.977	0.997	0.987
Average	0.974	0.903	0.935

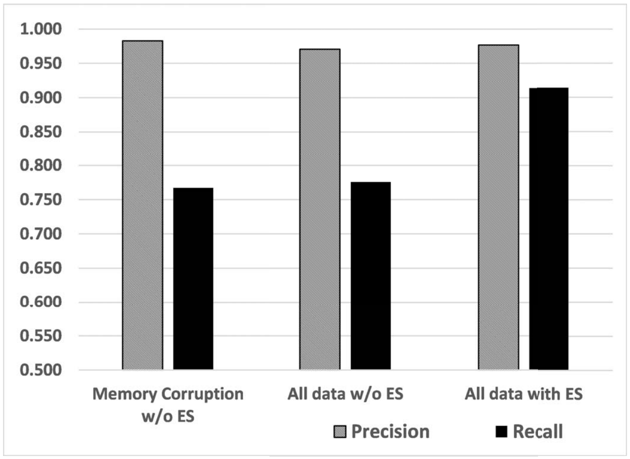

Fig. 5. Performance metrics of the SN extraction for tested models

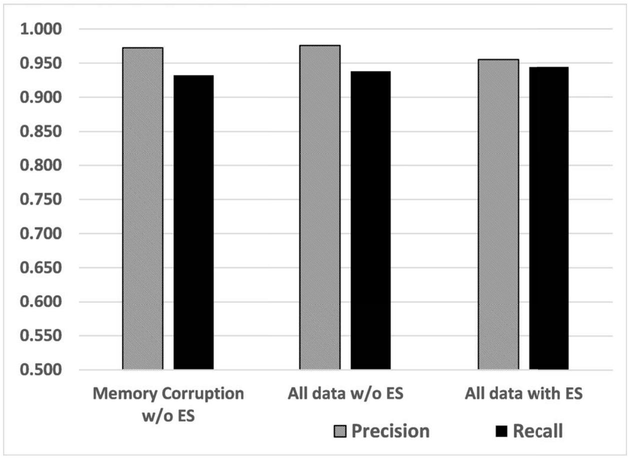

Fig. 6. Performance metrics of the SV extraction for tested models

trained on all training data was chosen.

B. Improving NER performance with ES rules

To improve the performance of the chosen NER model (that is trained on training datasets from all CVE types), we applied the developed ES. The confusion matrix for NER model augmented with the ES is presented in the Table VII and the performance metrics are presented in the Table VIII. Our NER model augmented with ES significantly outperforms the bare NER model in the SN extraction recall by almost 4% and improves the precision by 0.7%. While 4% recall improvement may not seem very impressive, in fact, the augmented NER model found additional 323 SN instances out of a total of 2367 SN instances in the dataset. In other words, the bare NER model missed 13.6% of all present SN instances.

In terms of SV extraction, the application of ES does not provide that significant improvement as for the SN extraction. The augmented NER model found 19 additional SV instances out of 2679 present SV instances, which is 0.7%. However,

TABLE VII
CONFUSION MATRIX FOR THE MODEL THAT IS TRAINED ON ALL TRAINING DATA (ALL CVE TYPES) AND APPLIES ES RULES

		Predicted		
		SN	SV	O
Actual	SN	2164	8	195
	SV	14	2530	135
	O	38	110	28301

TABLE VIII
PERFORMANCE METRICS FOR THE MODEL THAT IS TRAINED ON ALL TRAINING DATA (ALL CVE TYPES) AND APPLIES ES RULES. OVERALL ACCURACY = 0.985

	Precision	Recall	F1-score
SN	0.970	0.914	0.944
SV	0.955	0.944	0.950
O	0.988	0.995	0.992
Average	0.973	0.951	0.962

SV extraction rules are very cheap in a computational sense and improve overall performance.

The overall accuracy of the augmented NER model is better by 0.9%. Figure 5 and Figure 6 present performance comparison of all three tested models.

C. Matching SN with SV

To test the SN and SV matching capability of the developed system, we have randomly selected 60 real CVE description from the NVD database [45] from the 2015 year up to the 2020 year with ten CVE per each year. The difficulty level of the selected CVE drastically differs. For example, “CVE-2015-7489” has a simple one-sentence description that contains one SN instance of three words and seven words that belong to SV and represent two SV. On the other hand, the “CVE-2020-2050” description consists of four long sentences, has seven mentions of the same SN (“PAN-OS”) and eight associated versions. This second example of the CVE descriptions may be confusing even for the human expert.

We compared matching algorithm performance when it uses the bare NER model and augmented NER model. To compare the model performance in this task, we calculated such metrics as how many descriptions where the system did not find the affected products at all, how many SN instances were not found, and how many SV instances were not found.

During the evaluation, we inspected CVE descriptions and identified how many SN and SV in each description. Then we inspected the SN and SV extraction and matching results and calculated how many correct instances of affected product were found. In some cases, if the system found a part of the SN or SV, which is still understandable for a human expert, a partial score was assigned. For example, in the case of the “CVE-2019-5702”, the whole SN has to be “NVIDIA GeForce Experience”, while the system was able to extract only “NVIDIA GeForce”. Considering that the system extracted the SV version and matched it to the found SN, it is still understandable that SN belongs to “NVIDIA GeForce Experience”, therefore, we assigned that system had found 0.8

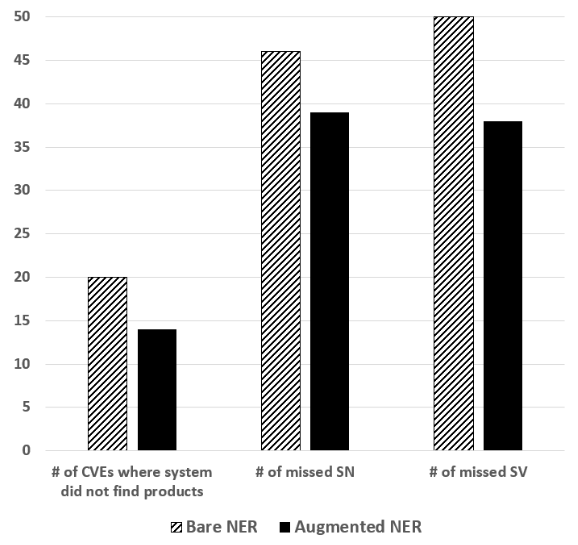

Fig. 7. Results of SN and SV matching for "bare" NER models and "augmented" NER model. Less is better.

SN out of 1 SN. However, if the system found a product that is not understandable without looking into the description, it got zero points for that SN. Similarly were scored the SV extraction performance. For example, we would assign 0.5 points to the extracted SV if the original SV consisted of two version numbers and the system found only one.

The performance results are presented in Table IX. The matching algorithm based on the NER model augmented with the developed ES outperformed the matching algorithm based on the bare NER model in all performance metrics. Figure 7 presents the results of SN and SV matching tests for both "bare" and "augmented" NER models. The augmented NER model did not find an affected product in 14 CVE descriptions, while the bare NER model-based method did not find an affected product in 20 descriptions. The augmented NER model missed 39 SN instances out of 101, while the bare NER model missed 46 out of 101 present SN instances. The augmented NER model also missed fewer SV instances in comparison to bare NER: 38 missed SVs vs. 56 (out of 136). Finally, the augmented NER-based method generated 56 correct matches, while the bare NER-based model generated only 46 correct matches out of 101.

One of the most difficult cases for both the bare NER model and augmented NER model was the description of the "CVE-2020-28373", which has 14 SN instances and 13 SV instances. Both models failed to identify even one SN or SV. This model behavior is explained by the very specific names of the affected products.

1) Matching algorithm results analysis.: In the case of "CVE-2017-17465", the bare NER based method did not detect any SN or SV instances, while the NER-ES based system successfully extracted the correct SN and related version. In the case of the "CVE-2015-6404" description, the bare NER

TABLE IX
PERFORMANCE OF THE SN AND SV MATCHING ALGORITHMS THAT ARE BASED ON BARE NER MODEL AND NER MODEL AUGMENT BY ES.

	Bare NER	Augmented NER
Did not find affected product at all (out of 60)	20	14
Missed SN instances (out of 101) / total	46	39
Missed SV instances (out of 136) / total	50	38
Found correct matches (out of 101) (more is better)	46	56

model detected only the SV instances and did not detect any SN. Therefore, the system did not extract any affected product from the description because it could not connect identified SV with SN. NER-ES model partially picked up the affected SN ("Hosted Collaboration"), and the system connected identified SV with this SN. While the extracted product name is not full, a networking specialist will likely identify the affected product.

Finally, the "CVE-2015-6416" description is interesting because it contains a long SN. The bare NER model successfully identified only one word that belongs to SN ("Interaction") and did not find any SV. Unfortunately, this one word is not enough for a specialist to identify the affected product. Our NER-ES model successfully identified the whole affected software name as well as its version. Based on that data, our system extracted the correct product: SN: Cisco Unified Email Interaction Manager and Unified Web Interaction Manager. SV: 11.0(1).

These examples demonstrate the benefits of combining ES with the deep learning approach, rather than just using a pure data science approach without applying knowledge in the cybersecurity domain.

D. Current state of CVE analysis and its improvement

To analyze how many CVEs do not have a proper field with affected products and their versions, we employed a special framework developed in our lab, which automatically scrapes CVEs from thousands of open sources. The system is able to acquire CVE from various CNAs and parse results into our database. We inspected how many CVEs have an empty "platform" field. Our analysis showed that 19,465 CVE out of 77,648 have an empty "platform" field, which is 25% of all stored CVEs in our database. Additional analysis of CVEs with the filled "platform" field revealed that the field's content does not correspond to the affected product(s) mentioned in the CVE description. To estimate the percentage of the CVEs with the relevant "platform" field, we randomly chose 100 CVEs with the filled "platform" field from our database and manually inspected them. In fact, only 21 CVEs had a relevant "platform" field. This means that only approximately 12,000 CVEs have a relevant affected product in their "platform" field. In other words, 84.2% of all stored CVEs in the database have either an empty or incorrect "platform" field.

We tested our NER-ES-based product name extraction method on these 100 CVEs as well. Our model successfully extracted correct products from 43 CVE descriptions, which increases the number of correct product names more than two times. In addition, some CVEs refer to more than one product, but, unfortunately, CVE sources provided only one product. Our method is capable of extracting multiple affected products from the description. In fact, among the inspected 100 CVEs, there are six cases when our method extracted more than one affected product mentioned in a CVE description, while the CVE source provided only one affected product.

E. Experiment result discussion

From the experiment results, one can clearly see the benefits of augmenting machine learning techniques with ES to extract affected products from CVE descriptions. However, this approach requires additional expertise in the cybersecurity domain in addition to the expertise in the artificial intelligence and machine learning domain. The integration of two approaches to the data analysis allowed us to obtain a better result compared to the pure data science approach.

In addition, the application of the ES allowed us to reveal numerous errors in the testing data set [35]. While we were analyzing the root cause of the high rate of false-positive identification of SN instances, even though the developed algorithm checks each potential SN in the CPE dictionary, we discovered many inconsistencies in the testing data set. These dataset inconsistencies include labeling the same software in one part of the dataset as SN while in other places of the same dataset, it was labeled as "other". For example, this was the case for the "Linux kernel" words. In some places, these words were labeled as SN ("Linux SN", "kernel SN"), while in another place, both words had labels "O" ("Linux O", "kernel O"). Clearly, the second case was incorrect.

Further investigation led us to discover that many words that belong to various software names were labeled as "Other". After a detailed review of the dataset, we fixed it and relabeled 496 SN instances and 43 SV instances. Further are examples of our dataset fixes: Linux kernel marked as SN, "Xen" marked as SN and its versions "4.1 and earlier" marked as SV, "Xen" SN and "4.9.x" marked as SV, "iOS" marked as SN and "9" as SV, "ChakraCore" marked as SN, etc. An improved dataset will be published for public use in the immediate future.

During the manual dataset verification, we also were verifying software names in the NVD CPE [9]. We have found that some software names are only partially in the CPE. The first software name is "Quest Privilege Manager". CPE contains only "Privilege Manager" and "Quest" separately. The second case is "Ichitaro Pro", where CPE has only "Ichitaro". However, Pro is clearly a part of JustSystems's product name. Finally, in the case of "Dameng DM Database Server", the CPE contains only "Database Server". However, there exists a product with this name. This double verification, allowed us to confirm that we have fixed and updated the entire dataset and that we avoided any possible bias towards any machine learning model.

Before we fixed the test dataset, the bare NER model was outperforming the precision of our augmented NER model in the SN extraction task (while recall of our model still was higher). That better performance is explained by the fact that our model finds those SNs that were wrongfully labeled as "Other" in the older version of the dataset.

V. Conclusion

Software vulnerability discovery and their publishing play an important role in mitigating various cyber-security threats and developing more secure software with enhanced confidentiality, integrity, and availability characteristics. There exist numerous authorities that publish CVE. Unfortunately, delayed processing of the discovered CVE report, such as affected product name extraction, may delay CVE publication and therefore create additional threats of cyber-attacks or some other unintentional cyber-security issues, due to delay in the patch developing and publishing processes. To address affected product name extraction from the reported CVE descriptions, we developed a system that is capable of extracting software name (SN) and associated software versions (SV) from the human-readable descriptions.

The developed system takes advantage of machine learning techniques such as deep learning, natural language processing (NLP) such as part-of-speech (PoS) classification, and ES. We have developed and trained a NER model that is based on bi-directional long-short term memory (LSTM) neural network model that is capable of classifying each word into SN, SV, or other (O).

To improve the NER model performance, we developed rules that rely on NLP techniques and our analysis of CVE descriptions. This analysis allowed us to develop rules for additional SN and SV extraction and rules to match them. The developed rules improved SN extraction recall by 8%. In addition, we employed a common product enumeration catalog (CPE) that allowed us to improve the SN extraction precision as well.

We employed the developed automatic system for the CVE acquiring and revealed that 84% of the stored CVEs in its database have either empty or incorrect associated products due to their absence in CVE report sources. Our empirical study demonstrated that the developed and presented method for the affected product name extraction is capable of reducing the number of CVEs with empty/incorrect "platform" field by more than two times. It is also, in contrast to the CVE sources, extracts more than one affected product.

In addition, the developed system allowed us to reveal more than 500 errors in the testing dataset and fix them. Finally, we have developed an additional dataset for testing product name extraction systems containing CVE from 2015 to 2020. These datasets are going to be published in the immediate future.

References

[1] Tripwire Security, "Unpatched Vulnerabilities Caused Breaches in 27% of Orgs," https://www.tripwire.com/state-of-security/vulnerability-management/unpatched-vulnerabilities-breaches/, 2019, (Accessed on 12/05/2020).

[2] National Vulnerability Database, "Nvd dashboard," https://nvd.nist.gov/general/nvd-dashboard#, 2018, (Accessed on 05/04/2018).

[3] "Vulndb," https://vulndb.cyberriskanalytics.com/, (Accessed on 09/01/2020).

[4] IBM, "IBM X-Force Exchange," https://exchange.xforce.ibmcloud.com/, (Accessed on 09/01/2020).

[5] R. Habalov and T. Schmid, "Vulncode-db," https://www.vulncode-db.com/, (Accessed on 09/02/2020).

[6] "Exploit database - exploits for penetration testers, researchers, and ethical hackers," https://www.exploit-db.com/, (Accessed on 09/01/2020).

[7] "Securityfocus," https://www.securityfocus.com/, (Accessed on 09/01/2020).

[8] Y. Dong, W. Guo, Y. Chen, X. Xing, Y. Zhang, and G. Wang, "Towards the detection of inconsistencies in public security vulnerability reports," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 869–885. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/dong

[9] "Official Common Platform Enumeration (CPE) Dictionary ," https://nvd.nist.gov/products/cpe, (Accessed on 10/26/2020).

[10] Y. Nan, M. Yang, Z. Yang, S. Zhou, G. Gu, and X. Wang, "Uipicker: User-input privacy identification in mobile applications," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 993–1008.

[11] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, "Autocog: Measuring the description-to-permission fidelity in android applications," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 1354–1365.

[12] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "{WHYPER}: Towards automating risk assessment of mobile applications," in *22nd {USENIX} Security Symposium ({USENIX} Security 13)*, 2013, pp. 527–542.

[13] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proceedings of the 36th international conference on software engineering*, 2014, pp. 1025–1035.

[14] J. Huang, Z. Li, X. Xiao, Z. Wu, K. Lu, X. Zhang, and G. Jiang, "{SUPOR}: Precise and scalable sensitive user input detection for android apps," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 977–992.

[15] W. You, P. Zong, K. Chen, X. Wang, X. Liao, P. Bian, and B. Liang, "Semfuzz: Semantics-based automatic generation of proof-of-concept exploits," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 2139–2154.

[16] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2013, pp. 345–355.

[17] X. Ye, R. Bunescu, and C. Liu, "Learning to rank relevant files for bug reports using domain knowledge," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 689–699.

[18] C.-P. Wong, Y. Xiong, H. Zhang, D. Hao, L. Zhang, and H. Mei, "Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis," in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 181–190.

[19] L. Tan, D. Yuan, G. Krishna, and Y. Zhou, "/* icomment: Bugs or bad comments?*," in *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, 2007, pp. 145–158.

[20] 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*, ser. KDD '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 105–114. [Online]. Available: https://doi.org/10.1145/1835804.1835821

[21] A. Khazaei, M. Ghasemzadeh, and V. Derhami, "An automatic method for cvss score prediction using vulnerabilities description," *Journal of Intelligent & Fuzzy Systems*, vol. 30, pp. 89–96, 08 2015.

[22] Y. Shen and G. Stringhini, "Attack2vec: Leveraging temporal word embeddings to understand the evolution of cyberattacks," 05 2019.

[23] L. Bilge and T. Dumitraş, "Before we knew it: An empirical study of zero-day attacks in the real world," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 833–844. [Online]. Available: https://doi.org/10.1145/2382196.2382284

[24] K. Nayak, D. Marino, P. Efstathopoulos, and T. Dumitras, "Some vulnerabilities are different than others - studying vulnerabilities and attack surfaces in the wild," in *RAID*, 2014.

[25] C. Sabottke, O. Suciu, and T. Dumitras, "Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits," in *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 1041–1056. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/sabottke

[26] E. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," 2010.

[27] N. Provos, M. Friedl, and P. Honeyman, "Preventing privilege escalation," in *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, ser. SSYM'03. USA: USENIX Association, 2003, p. 16.

[28] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, pp. 3111–3119, 2013.

[29] B. Jang, I. Kim, and J. W. Kim, "Word2vec convolutional neural networks for classification of news articles and tweets," *PLOS ONE*, vol. 14, no. 8, pp. 1–20, 08 2019. [Online]. Available: https://doi.org/10.1371/journal.pone.0220976

[30] 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 *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, 2016, pp. 90–101.

[31] S. Wu, Z. Fang, and J. Tang, "Accurate product name recognition from user generated content," in *2012 IEEE 12th International Conference on Data Mining Workshops*. IEEE, 2012, pp. 874–877.

[32] Z. Huang, W. Xu, and K. Yu, "Bidirectional lstm-crf models for sequence tagging," *arXiv preprint arXiv:1508.01991*, 2015.

[33] A. Graves, N. Jaitly, and A.-r. Mohamed, "Hybrid speech recognition with deep bidirectional lstm," in *2013 IEEE workshop on automatic speech recognition and understanding*. IEEE, 2013, pp. 273–278.

[34] E. Wåreus and M. Hell, "Automated cpe labeling of cve summaries with machine learning," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2020, pp. 3–22.

[35] Y. Dong, W. Guo, Y. Chen, X. Xing, Y. Zhang, and G. Wang, "Towards the detection of inconsistencies in public security vulnerability reports," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 869–885.

[36] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, "Neural architectures for named entity recognition," *arXiv preprint arXiv:1603.01360*, 2016.

[37] Z. Yang, R. Salakhutdinov, and W. W. Cohen, "Transfer learning for sequence tagging with hierarchical recurrent networks," *arXiv preprint arXiv:1703.06345*, 2017.

[38] S.-H. Liao, "Expert system methodologies and applications—a decade review from 1995 to 2004," *Expert systems with applications*, vol. 28, no. 1, pp. 93–103, 2005.

[39] D. Deeplearning4j, "Deeplearning4j: Open-source distributed deep learning for the jvm, apache software foundation license 2.0," 2016.

[40] C. Dos Santos and B. Zadrozny, "Learning character-level representations for part-of-speech tagging," in *International Conference on Machine Learning*. PMLR, 2014, pp. 1818–1826.

[41] J. Wieting, M. Bansal, K. Gimpel, and K. Livescu, "Charagram: Embedding words and sentences via character n-grams," *arXiv preprint arXiv:1607.02789*, 2016.

[42] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, 2015, pp. 649–657.

[43] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[44] A. OpenNLP, "Apache software foundation," *URL http://opennlp.apache. org*, 2011.

[45] National Vulnerability Database, "Nvd - vulnerabilities," https://nvd.nist.gov/vuln/, 2020, (Accessed on 12/14/2020).