Detection of Intrusions and Malware, and Vulnerability Assessment. 2020 Jun 11; 12223: 3–22.
Published online 2020 Jun 11. doi: 10.1007/978-3-030-52683-2_1

PMCID: PMC7338193

Automated CPE Labeling of CVE Summaries with Machine Learning

Guest Editor (s): Clémentine Maurice,[8] Leyla Bilge,[9] Gianluca Stringhini,[10] and Nuno Neves[11]
[8]Univ Rennes, CNRS, IRISA, Rennes, France
[9]NortonLifeLock Research Group, Biot, France
[10]Department of Electrical and Computer Engineering, Boston University, Boston, MA USA
[11]Department of Informatics, University of Lisboa, Lisbon, Portugal
Clémentine Maurice, Email: clementine.maurice@irisa.fr.
Contributor Information.

Emil Wåreus[12,13] and Martin Hell[13]

Abstract

Open Source Security and Dependency Vulnerability Management (DVM) has become a more vital part of the software security stack in recent years as modern software tend to be more dependent on open source libraries. The largest open source of vulnerabilities is the National Vulnerability Database (NVD), which supplies developers with machine-readable vulnerabilities. However, sometimes Common Vulnerabilities and Exposures (CVE) have not been labeled with a Common Platform Enumeration (CPE) -version, -product and -vendor. This makes it very hard to automatically discover these vulnerabilities from import statements in dependency files. We, therefore, propose an automatic process of matching CVE summaries with CPEs through the machine learning task called Named Entity Recognition (NER). Our proposed model achieves an F-measure of 0.86 with a precision of 0.857 and a recall of 0.865, outperforming previous research for automated CPE-labeling of CVEs.

Keywords: Machine learning, Open source, Vulnerabilities, CVE, CPE

Introduction

Back to Top

In almost all software development today, using open source and third-party components is crucial for its success. It is beneficial to the quality, security, functionality, and development efficiency. However, at the same time, it increases the exposure to vulnerabilities in code developed by third parties. To maintain control over the security of the developed software, the maintainers need to continuously monitor if vulnerabilities have been introduced or found in these third-party dependencies. This is commonly done with Dependency Vulnerability Management (DVM) tools that automate the process of Software Composition Analysis (SCA), and matches used software components with known vulnerabilities.

The main source of vulnerabilities is the National Vulnerability Database (NVD) [15]. These vulnerabilities have a unique Common Vulnerabilities and Exposures (CVE) identifier. The list of such identifiers is maintained by Mitre and includes a short summary of the vulnerability. In the last few years, around 30–50 new vulnerabilities have been given a CVE identifier and been recorded in NVD each day. Combining this with the fact that software projects can have thousands of dependencies, including transitive dependencies, it is clear that the process of identifying new vulnerabilities must be automated.

NIST security professionals take the CVEs as they are published by Mitre and link one or more Common Platform Enumerations (CPE) [14] to each CVE. These CPEs are used to specify which software and versions are vulnerable. NIST also adds other pieces of information, such as a CVSS score, and thus maintains a rich database of information about published vulnerabilities.

While the summary, as recorded in the original identifier provided by Mitre, often includes information regarding which product and versions are affected, the list of CPEs formalizes this information and provides it in a standardized, and machine-readable, format. Thus, the CPE is a crucial addition to the CVE information when vulnerability identification and assessment are being automated.

Unfortunately, far from all CVEs maintained in the NVD database are correctly linked to CPEs. Moreover, as reported in [4], there is a notable time-lag from the first CVE disclosure to the addition of CPEs to the vulnerability. In 2018, the median time to correctly assign the CPE metadata was 35 days. The manual effort performed in the analysis of CVEs is not limited to only assigning CPEs in this 35 day period, but we are only interested in the CPEs in our current experiments. As soon as the CVE is known (or even before), exploits are developed and attacks can be found in the wild. Thus, such a time-lag can leave a software system vulnerable to attacks since automated tools are not able to correctly inform developers and users of these vulnerabilities.

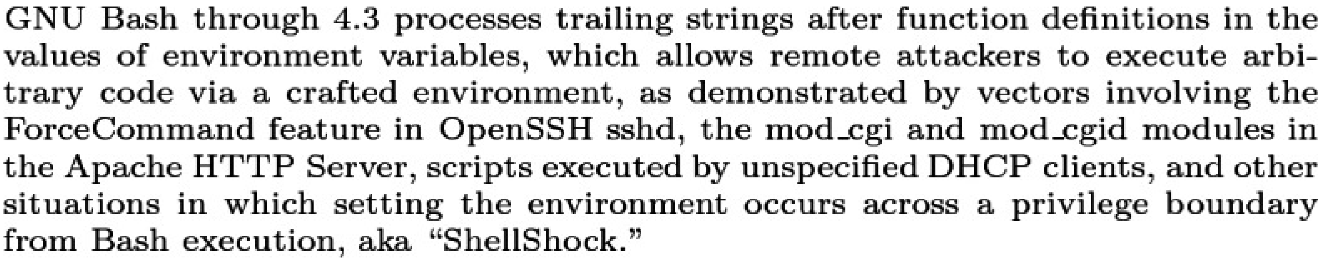
In this paper, we use Natural Language Processing (NLP), or more specifically, Named Entity Recognition (NER), to automatically build a CPE, or list of CPEs, from the summary text. We build a model inspired by [12]. As input to the model, we use word and character level embeddings, casing-features, and a security lexicon of common CPEs. The model itself consists of a Bidirectional Long-Short-Term Memory (BLSTM) network together with a Conditional Random Field (CRF) to determine the labels. Using such NLP algorithms, we achieve unprecedented performance, with an F-measure of 0.8604, recall of 0.8637, and precision of 0.8571.

The paper is organized as follows. In Sect. 2 we give a brief background on the vulnerability data of interest and Natural Language Processing. In Sect. 3 we present the dataset that we use and its corresponding labels. In Sect. 4 we frame the problem and determine how we evaluate our results. Section 5 presents our model, the features, and some theory for the model. Then, we present and discuss our results in Sect. 6. Related work is described in Sect. 7 and the paper is concluded in Sect. 8.

Background

Vulnerability Data

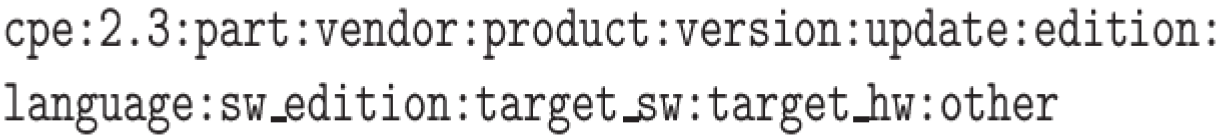
A new vulnerability is often reported as a CVE. The list of CVEs is maintained by Mitre and each entry contains a unique CVE number, a short summary, and at least one external reference [20]. The CVE summary typically includes the affected product and versions. An example of the ShellShock CVE-2014-6271 is given below.



GNU Bash through 4.3 processes trailing strings after function definitions in the values of environment variables, which allows remote attackers to execute arbitrary code via a crafted environment, as demonstrated by vectors involving the ForceCommand feature in OpenSSH sshd, the mod_cgi and mod_cgid modules in the Apache HTTP Server, scripts executed by unspecified DHCP clients, and other situations in which setting the environment occurs across a privilege boundary from Bash execution, aka “ShellShock.”

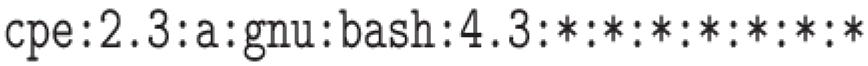
This information is then used by NVD, adding, among other things, a CVSS score, and a list of CPEs. The CVSS score provided by NIST is environment independent, but useful when assessing the severity of the vulnerability. The CPE provides a standardized string for defining which product and versions are affected by the vulnerability.

The current version of CPE is 2.3. The format is specified in [14], and is given by the string



```
cpe:2.3:part:vendor:product:version:update:edition:
language:sw_edition:target_sw:target_hw:other
```

The first part defines that it is a CPE and its version. Then, *part* can be one of h for hardware, a for application and o for operating system. The following fields are used to uniquely specify the component by defining *vendor*, the name of the *product*, the product *version* etc. It is common to use the fields up to and including version, even though, as can be seen, further details about the component can be defined. An example, as can be found in CVE-2014-6271, is given by



```
cpe:2.3:a:gnu:bash:4.3:*:*:*:*:*:*:*
```

NVD also provides a JSON feed with CVE data for each vulnerability. This feed supports additional fields for defining ranges of versions that are vulnerable. This feed provides a more efficient representation if there are many versions affected. This feed is further detailed in Sect. 3.

NVD consists of around 130 000 vulnerabilities (early 2020). The summary is given immediately when the CVE is published since it is required by Mitre, while the CPE is later added by NVD. The discrepancy differs between different CVEs, but an analysis in [4] reported that, in 2018, the median to correctly assign CPE data was 35 days.

Natural Language Processing and Named Entity Recognition

Natural Language Processing (NLP) is the task to make computers understand linguistics, usually with the support of machine learning. Within NLP, tasks such as machine translation, document classification, question answering systems, automatic summary generation, and speech recognition are common [10]. One of the main advantages of using machine learning for NLP is that the algorithms may gain a contextual semantic understanding of text where classifications are not dependent on a single word, but rather a complex sequence of words that can completely alter the meaning of the document. This may be beneficial to our system, as new CPEs that have not been seen before in the NVD database may be correctly classified from the CVE-summary through a contextual understanding of the document.

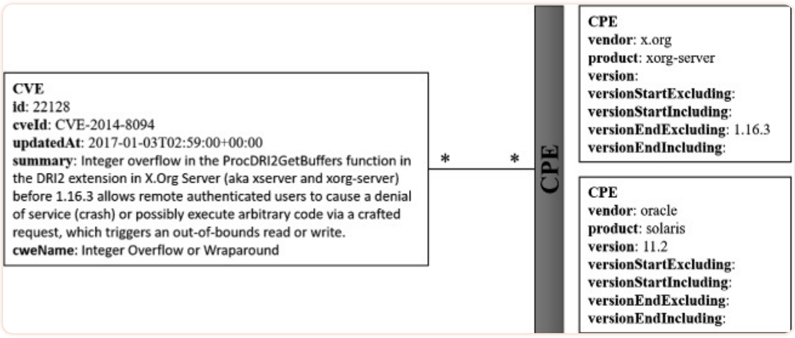
Named Entity Recognition (NER), or sequence labeling, is the NLP task of classifying each word in a sequence. One of the most common benchmarks in NER is the CoNLL-2003 dataset [21], where the task is to label words with ether person-, organization-, or location-names. NER is an important task within NLP, as a system needs to understand what category a word or sub-sequence belongs to truly understand the contextual meaning of the document.

Data and Labels

To successfully create machine learning models, it is necessary to collect data to train it. The goal for the model is to learn the general underlying structure of the problem through training on that data, which acts as a representation of that problem. This data is referred to as the *dataset*. Our dataset consists of historical vulnerabilities with already determined CPEs. These can be retrieved using the NVD data feed. Each entry in the dataset have the following features:

- **cveId:** The unique identifier and name for each CVE.
- **updatedAt:** The date and time of the last update from NVD for this particular CVE.
- **summary:** A text description of the CVE, often naming the vulnerable software, including product, vendor, and version.
- **cweName:** The Common Weakness Enumerator.
- **cpes:** A list of CPEs linked to this particular CVE. Each CPE contains:
  - *vendor:* The vendor of the product or software.
  - *product:* Name of the software.
  - *version:* An exact statement of a single vulnerable version.
  - *versionStartExcluding:* All versions are vulnerable after (excluding) this version.
  - *versionStartIncluding:* All versions are vulnerable after (including) this version.
  - *versionEndExcluding:* All versions are vulnerable before (excluding) this version.
  - *versionEndIncluding:* All versions are vulnerable before (including) this version.

Our analysis concludes that 81.9% of all CPEs from CVEs in NVD only specifies one of the following fields: *version*, *versionStartExcluding*, *versionStartIncluding*, *versionEndExcluding*, and *versionEndIncluding*. About 14.5% have no version range specified, and 3.6% have exactly two version ranges specified. Figure 1 illustrates how a CVE-CPE link can be structured.



CVE
id: 22128
cveId: CVE-2014-8094
updatedAt: 2017-01-03T02:59:00+00:00
summary: Integer overflow in the ProcDRI2GetBuffers function in the DRI2 extension in X.Org Server (aka xserver and xorg-server) before 1.16.3 allows remote authenticated users to cause a denial of service (crash) or possibly execute arbitrary code via a crafted request, which triggers an out-of-bounds read or write.
cweName: Integer Overflow or Wraparound

CPE
vendor: x.org
product: xorg-server
version:
versionStartExcluding:
versionStartIncluding:
versionEndExcluding: 1.16.3
versionEndIncluding:

CPE
vendor: oracle
product: solaris
version: 11.2
versionStartExcluding:
versionStartIncluding:
versionEndExcluding:
versionEndIncluding:

Fig. 1.

Overview of the relationship between a CVE and multiple CPEs.

As seen in Fig. 1, some of the product and vendor strings can be found in the summary. The version can also be found in the summary but is dependent on the context of the summary to determine if other versions are vulnerable (in this case all versions before version 1.16.3). In this paper, only the summary is regarded as input features, the CPE-list as the labels, and all other data is disregarded in the model. Naturally, all CPEs may not be possible to link to the summary through text models as there is no occurrence of the product or vendor in the paragraph. This is shown in Fig. 1, as Oracle Solaris is not mentioned in the paragraph, but is considered vulnerable from the context that X.Org xorg-server is vulnerable. In our analysis, we find that about 59% of CPEs can be mapped with regex methods to its CVE summary, and for 27% of the CVEs, all corresponding CPEs can be mapped to its summary. To map a CPE to a CVE-summary, each single word/version-string in the CPE must be matched to a word in the summary disregarding casing and special characters. Multi-word labels must be matched in the correct order, and up to 5 intermediate words are disregarded.

A sequence word labeling model requires a label for each word in the sentence. There are eight labels to consider in the CPEs provided by NVD: *vendor, product, version, versionStartExcluding, versionStartIncluding, versionEndExcluding, verisonEndIncluding,* and *O* (which denotes the none-label). Some vendors or products consist of multiple words, which need to be accurately predicted by the model. To denote this, labels are split into B- and I-labels where B denotes a start of a label, and I denote the word following the previous B or I labeled words. A part of an example sentence, taken from the CVE summary in Fig. 1, can be seen in Table 1.

Table 1.

Example of labeled sentence.

Text:	extension	in	X.Org	Server	before	1.16.3	allows	remote	authenti
Label:	O	O	B-product	I-product	O	B-versionEndExcluding	O	O	O

Problem Statement and Evaluation

The high-level problem we try to solve is one of determining what software and what versions are described in a document. This could be limited to mapping each document to already existing CPEs in the available CPE-list [16]. We choose not to do this because the available CPE-list is deficient as it is lacking entries for many products. Analyzing all available CPEs mentioned in CVEs, about 60% of those are only mentioned once. Thus, the probability of a new CVE describing a new, none existent, CPE is high. Therefore, we decide to allow our system to create new CPEs, in terms of finding software that has not been mentioned in any existing CPE list yet. A completely successful NER-predicted CVE-summary from our test-data will let us reconstruct all corresponding CPEs correctly, while the model may create new CPEs on new CVE-summaries.

To determine success, we measure our system as conventional NER-model. Over each predicted sequence we calculated the *precision*

$$precision = \frac{\sum true\_positive}{\sum true\_positive + \sum false\_positive}, \tag{1}$$

recall

$$recall = \frac{\sum true\_positive}{\sum true\_positive + \sum false\_negative}, \tag{2}$$

and their harmonic mean *F1*

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}, \tag{3}$$

and remove every correctly predicted O-label from the measurements as it greatly inflates the result. We also measure the overall *accuracy* of the model as the number of completely correctly NER-predicted CVE-summaries divided by the total number of summaries in that particular dataset. A hold-out strategy is implemented to measure these metrics, with a training set to train the model on, a validation set to optimize the model during development, and a testing set to test the final result.

The final model is intended to be used in DVM-tools to provide an estimation of CPEs that are associated with a CVE. This estimated association is provided with no or very little time-lag, which may prevent "one day"-vulnerabilities. The time-lag was further discussed in Sect. 1. Furthermore, additional CPEs could be provided to older CVEs that have been incorrectly labeled with the wrong CPEs or are missing some CPE associations.

Modeling

In this section, the feature engineering and machine learning model is described. The model is inspired by the work of [2] and [12] in the context of generic NER, where the contribution was to feed the text-data into multiple input sequences to capture different levels of semantics in the text. In brief, words are converted to vector representations [13] to embed contextual semantic meaning in its vector values. Additional word level and character level features are engineered to capture variations, such as word level numerical saturation, casing, and commonly used CPE-products and -vendors. These features are fed into a recurrent Bidirectional Long Short-term Memory (BLSTM) network to perform sequence labeling. Dropout [19] is used to regularize the model after concatenated embeddings, after the recurrent layer, and within the case-feature layer. This model was chosen as it presented a superior performance on the specific task of CPE-labeling compared to other common architectures, such as BERT [3]. The model is also suitable, as domain knowledge can easily be embedded through feature engineering. An overview of the architecture is presented in Fig. 2.

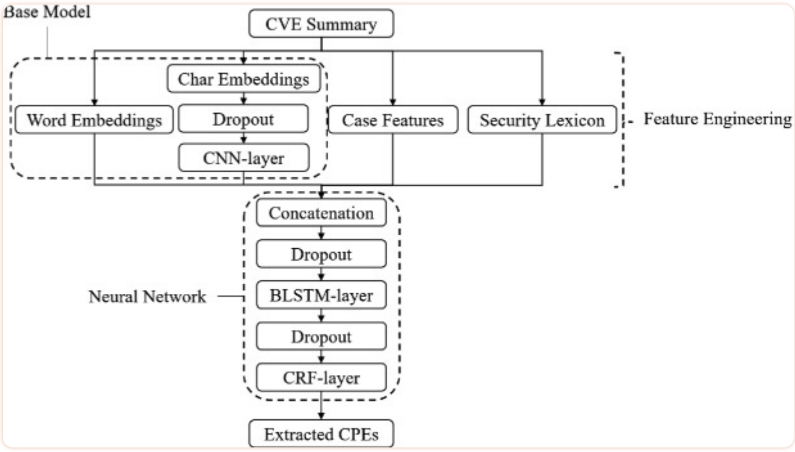


Fig. 2.

Overview of the model architecture and data pipeline.

Feature Engineering

This subsection will discuss the four parallel input layers used in the feature engineering part of our model as seen in Fig. 2. These are word level embeddings, character level embeddings, word level case-features, and a word level lexicon of known statements. The word and character level embeddings are regarded as part of the *base model*, and case and lexicon features are regarded as optional/experimental features to the model. The output features are concatenated into an information rich sequential matrix that is fed into a neural network described in Sect. 5.2.

Word Level Embeddings. Each word is transformed into a 50, 100, 200, or 300 dimensional numerical vector to represent the semantics of that word with Glove embeddings [18]. These embeddings are pre-trained on a large set of Wikipedia articles and consists of a vocabulary of 400 000 words. This language model serves as a good starting point for our experiments, as they are well documented and tested, which enables us to look into other variables in the modeling. These embeddings are not tuned during training and missing words from the vocabulary are assigned a default randomly generated vector.

Character Level Features. To extract character level features for each word, we employ a three-stage process of embedding on character level, applying a one-dimensional convolution (CNN-layer), and extracting the final word-features with a max-pooling layer. The embeddings are randomly initialized and tuned during training. Dropout is applied to prevent the model from overfitting. The employed CNN-layer has a filter-size of 30 and a kernel-size of 3. A max-operation is done over each filter, so each word outputs a character-feature vector of shape (1, 30), and for the whole word-sequence a shape of (word-sequence-length, 30). Character level features enable the model to learn new words through decoding of character-sequences, and can thereby give similar output-values to insignificant variations of very similar character sequences. As our text-domain (security) is quite different from the pre-trained word level embeddings (Wikipedia), the character level embeddings enable our model to learn security-related semantics.

Word Level Case Features. In the task to find versions, products, and vendors, casing and other word-properties may be important to determine the label of that particular word. For instance, it is common that products' and vendors' names are capitalized. The version label contains a high concentration of character level digits, but may also contain mid-word punctuation and special characters. Table 2 shows the different case-features, which are fed into random-uniformly initialized trainable embeddings with the same dimension as the number of cases.

Table 2.

Number of entries in security lexicon

Case	Property
Numeric	Integer fraction = 1.0
Mostly numeric	Integer fraction > 0.5
Contains numeric	Integer fraction $\leq$ 0.5
All lower	All lower case
All upper	All upper case
Initial upper	First character upper case
Default	All other cases

Security Lexicon. To embed domain knowledge into the system, a security-lexicon is built. The labels *product* and *vendor* are included in the lexicon features. The lexicon is constructed from the complete set of CVEs from the NVD database consisting of about 130 000 vulnerabilities describing about 50 000 different products, excluding all CVEs in the validation and test dataset. Each entry into the lexicon can describe one of three entities, *product*, *vendor*, and *product and vendor*. Some product/vendor names exist both as products and vendors, which explains this separate feature. The total number of entries in the lexicon can be seen in Table 3.

Table 3.

Number of entries in security lexicon

Feature	Number of entries
Product	8513
Vendor	3097
Product and vendor	1005

When constructing the security lexicon, only common CPE-products and -vendors are added to the lexicon. The cutoff was set to the top 80% of the most common products and vendors to avoid CPEs with very few mentions. As seen in Fig. 3, the accumulated product mentions are heavily skewed towards products with very few mentions. This distribution may discourage the use of a lexicon-feature and increase the importance of case-features and contextual understanding of the model, as the probability of new CVE-summaries containing already existing CPEs has historically been low.

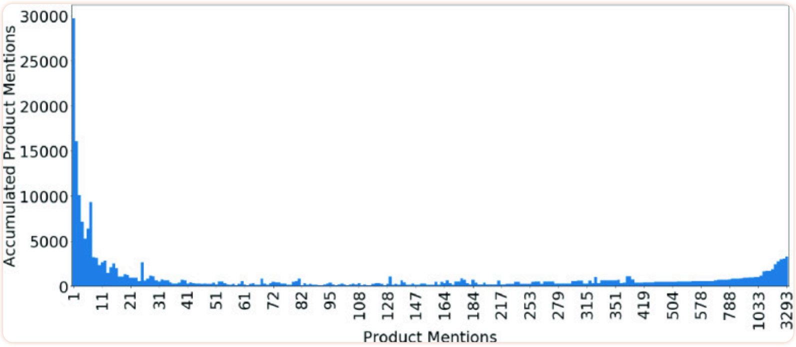


Fig. 3.

Accumulated mentions of product over the number of mentions of a product. The X-axis denotes the number of mentions of individual CPE-product and the Y-axis denotes the number of accumulated mentions of products with X-mentions. The mean of the distribution is 4.69 mentions per product and the median is 1 mention per product.

Neural Network

The input layer of the model consists of some or all features described in Sect. 5.1. The outputs of these features are all considered as embeddings that can be concatenated into a high-dimensional feature-map considering multiple characteristics of the input sequence. These concatenated embeddings are then fed into a neural network for sequence classification. The network architecture is inspired by [12], where the embeddings are fed into a Bidirectional Long Short-term Memory (BLSTM) layer and labels are decoded in a Conditional Random Field (CRF).

BLSTM-Layer. The LSTM [7] neural network unit is a type of recurrent layer that has theoretically strong capabilities to capture long-distance dependencies in sequential data. In text-data, recurrent models are capable of capturing contextual semantics in the data, and correctly model the sequential variations and dependencies of that text data. Conventional recurrent units suffer from problems such as the vanishing and exploding gradient [1, 17] which disables these networks to be effective on some tasks. The LSTM unit handles these complications by an internal architecture consisting of an *input gate*, *output gate*, *forget gate*, and a *cell state*. An overview of the LSTM cell can be seen in Fig. 4.

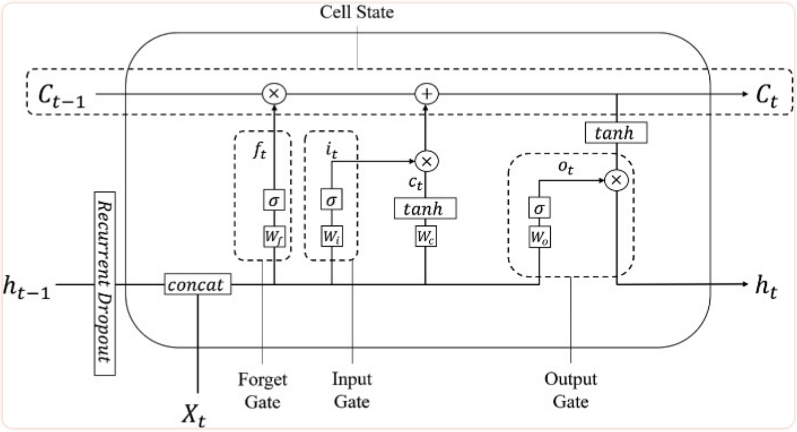


Fig. 4.

Long Short Term Memory cell. The input gate, output gate, forget gate, and cell state are marked in dotted lines.

In the figure, X_t denotes the *t*'th embedded input word to the LSTM cell and *h* represents the hidden state. The variable h_{t-1} is the output from the previous LSTM cell and h_t serves as the output prediction from this LSTM cell for the *t*'th word in the sequence. *C* denotes the cell state, which passes the memories of the already processed sequence to the LSTM cell. The forget gate is a nested neural network with a sigmoid activation function that scales the previous hidden state sequence between 0 and 1, where a low output value for a particular part of the sequence denotes that word should be forgotten. The output from the forget gate f_t is derived through

$$f_t = \sigma(W_f \times concat(h_{t-1}, X_t) + b_f), \tag{4}$$

where W_f and b_f are the trainable weights. The activation function σ is derived through

$$\sigma(x) = \frac{1}{1+e^{-x}}. \tag{5}$$

The input gate values are derived similar to Eq. (4),

$$i_t = \sigma(W_i \times concat(h_{t-1}, X_t) + b_i), \tag{6}$$

where W_i and b_i are trainable weights as well. Similarly to Eq. (4), the sigmoid in Eq. (6) normalizes the input values and previous hidden state between 0 and 1, which corresponds to their relative importance in this particular time step *t*. This layer is responsible to decide what new data should be added to the cell state. To calculate the cell state, the input and previous hidden state is passed through the following equation

$$\hat{c}_t = tanh(W_c \times concat(h_{t-1}, X_t) + b_c), \quad 7$$

to calculate the actual information that the input at step *t* brings. W_c and b_c are trainable weights. The *tanh* function normalizes the input between -1.0 and 1.0 through the following equation

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \quad 8$$

The relative importance is calculated for *X* and *h* and applied to the output from Eq. (7), which together with the forget gate forms the cell state through

$$C_t = f_t \times C_{t-1} + i_t \times \hat{c}_t, \quad 9$$

where C_{t-1} is the previous cell state. To calculate the output from a particular part of the sequence, which corresponds to the hidden state h_t, the input X_t and h_{t-1} are passed through an output gate. This gate decides what information should be passed to the next hidden state and output as a sequence prediction. The output gate is derived through

$$\hat{o}_t = \sigma(W_o \times concat(h_{t-1}, X_t) + b_o), \quad 10$$

where W_o and b_o are trainable weights and the current hidden state is calculated through

$$h_t = \hat{o}_t \times tanh(C_t). \quad 11$$

The output is passed to the next layer of the model, and is a matrix of shape $[batch\_size,\ sequence\_length,\ weight\_shape]$, where the $batch\_size$ is the number of parallel input examples fed to the model, $sequence\_length$ is length of the sentence, and $weight\_shape$ is a user set parameter that decides the number of weights used in the four nested neural networks.

To make this LSTM layer bidirectional [6], one simply use two separate, but identical, LSTM layers that pass over the input sequence in one direction each. The output is then concatenated. The output is regularized with dropout [19].

The reason for using a BLSTM is that an LSTM cell does not know anything about the future sequence $t+1, t+2, \ldots$, which may be contextually valuable. For instance, when classifying a version, a part of the sequence may be "[..] vulnerable version 1.3.4 and earlier". A BLSTM can capture the semantic meaning of "and earlier", and correctly classifies this as *versionEndIncluding*.

CRF-Layer. As shown in the architectural overview in Fig. 2, the output from the BLSTM is fed to a Conditional Random Field (CRF) [8] layer. The benefits of a CRF layer is statistically correlated label determination when assigning a class to a word in a sequence. For instance, the probability of a word being labeled with *I-product* increases if the previous word has been labeled with *B-product*. With CRF, labels are assigned jointly to reflect a final prediction for all entities in the sequence that make sense together. This is done through conditional probabilities and global normalization of a random field model.

Consider the output sequence of the BLSTM-layer $\mathbf{h} = \{\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_i, \dots, \mathbf{h}_N\}$, where $\mathbf{h}_i$ denotes the numerical vector output from the BLSTM-layer corresponding to the *i*'th word from the CVE-summary word sequences of length *N*. The label sequence $\mathbf{y} = \{y_1, y_2, y_i, \dots, y_N\}$ denotes each corresponding labels to the CVE-summary word sequence, where y_i denotes the predicted label for the *i*'th word. $Y(\mathbf{h})$ denotes the universe of all possible labels for $\mathbf{h}$. The conditional random field describes the conditional probability of label y_i in respect to input h_i and surrounding labels $y_{v \neq i} = y_{v \sim i}$, where $\sim$ denotes *v* as close to *i*, as $p(y_i | h_i, y_v, v \sim i)$ over all possible label sequences.

To determine the probability, a layer of weights $\mathbf{W}$ and biases $\mathbf{b}$ are used as

$$p(\mathbf{y}|\mathbf{h};\mathbf{W},\mathbf{b}) = \frac{\prod_{i=1}^{n} \gamma_i(y_{i-1}, y_i, \mathbf{h}_i)}{\sum_{y^* \in Y(\mathbf{h})} \prod_{i=1}^{n} \gamma_i(y^*_{i-1}, y^*_i, \mathbf{h}_i)}, \tag{12}$$

where

$$\gamma_i(y', y, \mathbf{h}_i) = exp(\mathbf{W}^T_{y',y} \mathbf{h}_i + \mathbf{b}_{y',y}). \tag{13}$$

The weights are trained through gradient descent and the Adam optimizer [11], as the rest of the model. The output of the CRF-layer is decoded from the highest conditional probability over the full sequence and serves as the output of the model.

Results and Discussion

Training

To train the model a dataset of 15190 CVEs from NVD was used, with an evaluation set of 3798 entries and a test set of 4748 entries. The test and evaluation split was done randomly. Experiments were conducted on whether to do a time-split instead of the dataset to prevent look-ahead bias but resulted in an insignificant performance change. The model was optimized with Bayesian hyperparameter optimization [9] over the following hyperparameters:

- The *learning rate* is a parameter that scales how much each weight should be updated in each gradient descent step [11].
- The *number of cells in the LSTM-layers* determines the size of the weight matrices W_f, W_i, W_{of}, and W_c, and their corresponding biases.
- Whether the *casing features* should be used.
- Whether the *lexicon features* should be used.
- The dimension of *word level embeddings* of pre-trained vectors.
- The dimension of *character level embeddings* of randomly initialized vectors.
- The *Dropout*-rate before and after the BLSTM-layer, and inside the char-features.
- The *Recurrent dropout*-rate in the LSTM-cells which determines the dropout rate of previous hidden state h_{t-1}.

The training was performed on NVIDIA TESLA K80 GPU and it took about 4–6 h to train the model once. In total, it took about 30 h to do the full training sweep on 16 K80s for 80 training iterations with different hyperparameter settings. This amounts to about 20 GPU-days. The parameter search space can be seen in Table 4. The Adam optimizer [11] was used to update the trainable parameters in the model and early stopping to reduce the risk of overfitting.

Table 4.

Hyperparameters search space and parameters used for best result.

Hyperparameter	Optimal value	Search space
Learning rate	0.00113	0.0001 to 0.01
LSTM cells	305	100 to 400
Use casing	True	True or False
Use lexicon	False	True or False
Word embedding dimension	100	50, 100, 200, or 300
Character embedding dimension	119	10 to 120
Dropout	0.2106	0.2 to 0.8
Recurrent dropout	0.2486	0.2 to 0.8

Main Results

In Table 5 the results are presented for the different model configurations. It is clear that the security lexicon did not provide any significant signal to improve the model. The word level casing feature proved beneficial to the performance with a significant improvement over the base model. The best performance on the test set was attained without the lexicon features and with casing features with an F-measure of 0.8604, a precision of 0.8571, and a recall of 0.8637. It is also clear that the same model had the best performance on the validation set, but we can see some indications of overfitting to the training-set as the F-measure, recall, and precision are much higher. This may indicate that additional performance could be gained with more aggressive regularization techniques. The fully combined model had much worse performance on the training set and similar performance on the test and validation set. This may indicate that further training and hyperparameter optimization could increase the performance of this model and enable it to surpass the other options.

Table 5.

Results of the four training cases

Model	Test set			Validation set			Training set		
	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall
Base model	0.8499	0.8437	0.8562	0.8435	0.8379	0.8491	0.9498	0.9404	0.9595
+ lexicon	0.8554	**0.8604**	0.8505	0.8493	0.8570	0.8418	0.9986	0.9983	0.9989
+ case	**0.8604**	0.8571	0.8637	0.8533	0.8536	0.8530	0.9963	0.9952	0.9973
+ lexicon + case	0.8574	0.8505	**0.8645**	0.8527	0.8482	0.8572	0.9422	0.9336	0.9510

An example of the output can be seen in CVE-2018-11761, where *"In Apache Tika 0.1 to 1.18, the XML parsers were not configured to limit entity expansion. They were therefore vulnerable to an entity expansion vulnerability which can lead to a denial of service attack."* is correctly parsed to ***vendor****: apache*, ***product****: tika*, ***versionStartIncluding****: 0.1*, ***versionEndIncluding****: 1.18* by the model. This example was in the test set, and is therefore never seen by the algorithm during training.

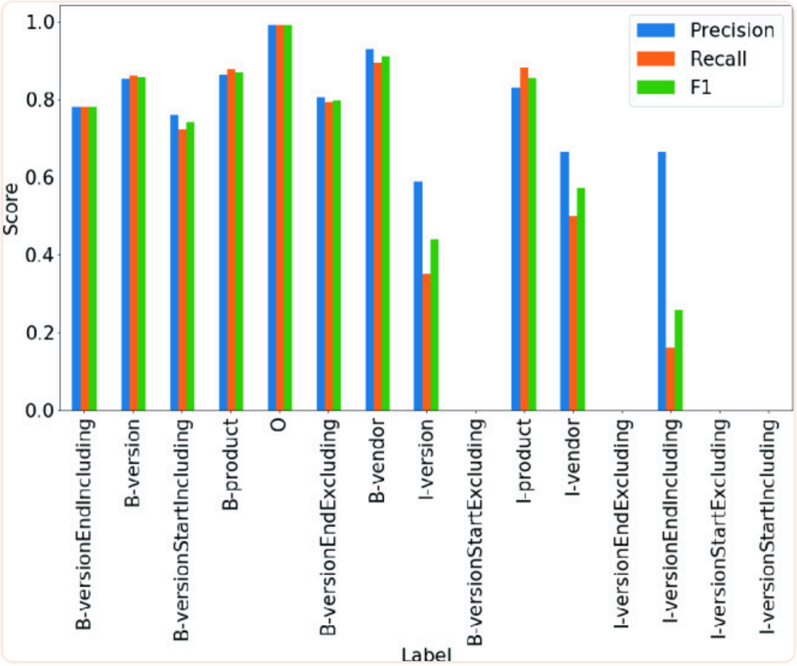
Performance over CPE-product, -vendor, and -version

At a more granular level shown in Table 6, we can observe the performance of each label on the test set, as well as the number of instances of each label in the test set *Label Count* and the number of predicted instances *Prediction Count*. We can see that some classes perform better than others. The F-measure is high for B-vendor, B-product, and B-version, as well as I-product. It is clear that there is a correlation between Label Count and all performance scores, which makes sense for this type of model as neural networks tend to be very data-hungry. In Fig. 6 labels with more examples in the dataset clearly have higher performance than less common labels. There seems to be a cutoff at approximately 300 examples to have an F-measure above 0.8. We can also observe that the performance for multi-word labels are worse, as the scores for I-labels are lower. To further increase the performance on I-labeled entries, it may be beneficial to create n-grams features in the lexicon or collect additional data for those particular cases. Figure 5 visualizes the results from Table 6. The model achieves a similar distribution over each label, which is visualized in Fig. 7.

Table 6.

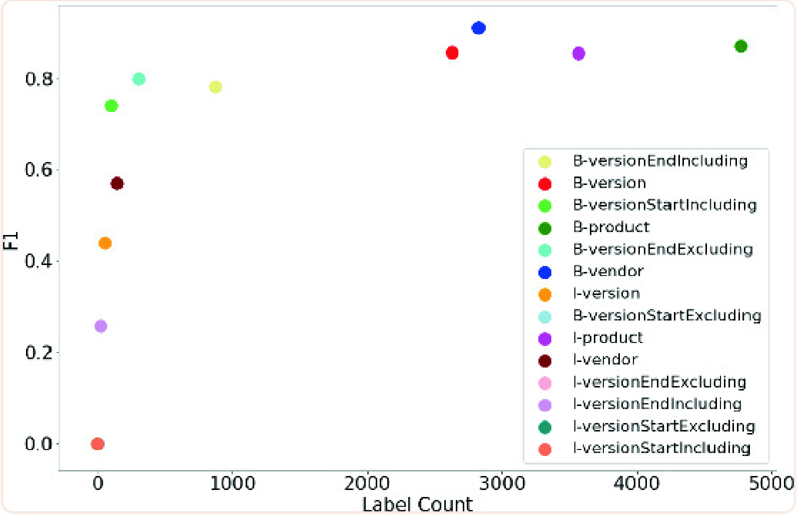
Granular test results from model with case features and without lexicon. Scores are over each possible label for the model. Label Count describes how many instances of that particular label is present in the test set, and Prediction Count describes how many predictions the model produces for a particular label.

Label	F1	Recall	Precision	Prediction count	Label count
B-versionEndIncluding	0.7817	0.7817	0.7817	875	875
B-version	0.8573	0.8618	0.8527	2655	2627
B-versionStartIncluding	0.7415	0.7238	0.76	100	105
B-product	0.8711	0.8774	0.8649	4840	4771
O	0.9935	0.9931	0.9938	184649	184768
B-versionEndExcluding	0.7987	0.7922	0.8053	303	308
B-vendor	0.9126	0.8951	0.9308	2715	2823
I-version	0.4396	0.3509	0.5882	34	57
B-versionStartExcluding	0	0	0	2	1
I-product	0.8549	0.8812	0.8302	3787	3568
I-vendor	0.5714	0.5	0.6667	111	148
I-versionEndExcluding	0	0	0	0	1
I-versionEndIncluding	0.2581	0.16	0.6667	6	25
I-versionStartExcluding	0	0	0	0	0
I-versionStartIncluding	0	0	0	0	0



[Fig. 5.](#)

Precision, F-measure, and Recall over each possible class for the model with case-features and without lexicon-features.



[Fig. 6.](#)

Scatter plot over Label Count and F1-score for each class (excluding ‘O’). This plot indicates that there seems to be a minimum amount of examples in each class to achieve an F1-score above .8 at approximately 300.

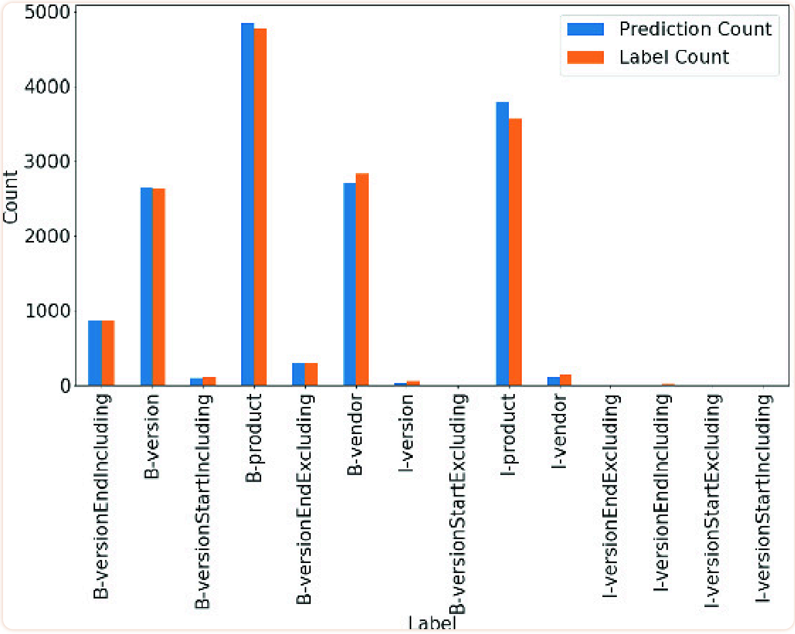


Fig. 7.

Label and Prediction count for each class in the test dataset. Note that the ‘O’-label is removed for this visualization.

Feature Analysis

The lexicon features did not provide any significant performance gains together with or compared to the case-features. It is possible that the case features better captured characteristics of the vendor and product labels since those are commonly capitalized in some manner, rather than over-relying on a fairly static memory of common labels. This result is in line with the distribution of products shown in Fig. 3, as 60% of all products NVD are mentioned only once. Other papers, such as [4] and [5] use keyword-based systems or features targeting narrow properties of the vendor- and product label. These systems are not taking the context of the sequence into consideration when performing classification, which we believe is the main reason why we achieve significantly better results. With a contextually aware classification, our system is able to find new CPEs that have never been seen before by NVD in any CVEs. This is highly desirable in a system to automatically extract CPEs from CVEs due to the distribution in Fig. 3.

Increasing Performance on Rare Labels

Our dataset consists of about roughly 20% of all available CVEs NVD, which may limit our results. This particular subset was chosen as over 90% of all CPE-version, -product, and -vendor strings for all CPEs paired with a CVE could be found in the summary through regular expression. Stronger regular expressions could increase the number of training examples, and further increase the performance of the system. To increase the performance in the more challenging task of classifying multi-word labels, an overweight to these cases could be provided to the training data, or the model could be pre-trained on a larger high-quality data set such as CoNLL 2003 [21].

Error Analysis

The overall accuracy of the system of correctly labeling every CPE in each CVE is 0.6744, measured as the full CVE-summary being correctly NER-annotated by the system. If the system only regards pursuing vendor and product classification, the accuracy would increase to 0.7772, which is more comparable to earlier research as they do not always search for version ranges in the summary. The distribution of the number of errors in all sequences that were incorrect is visualized in Fig. 8, where the accumulated error for sequences with up to 3 errors stands for about 80% of the miss-classified summaries. Looking further into what types of errors the model makes, Fig. 9 visualizes in total about 90% of all miss-classifications. In the top four spots, regarding about 40% of the error are bad predictions on the product label, with the I-label scoring higher than the B-label. This strengthens the hypothesis that the system needs improvement to better find multi-word labels. The top two mistakes contribute to a lower precision, as it incorrectly finds a CPE-product where there is none, and error three and four contributes to a lower recall as products are miss-classified as a O-words.

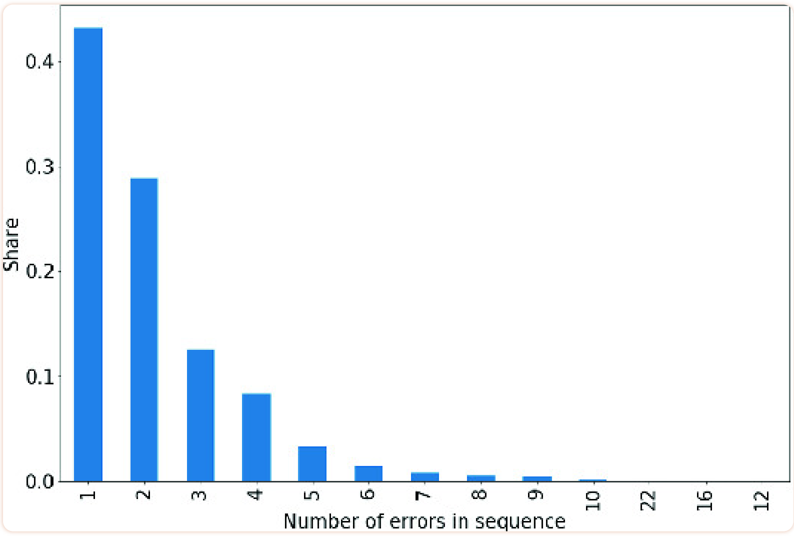


Fig. 8.

Distribution of number of miss-classifications in a sequence over all miss-classifications.

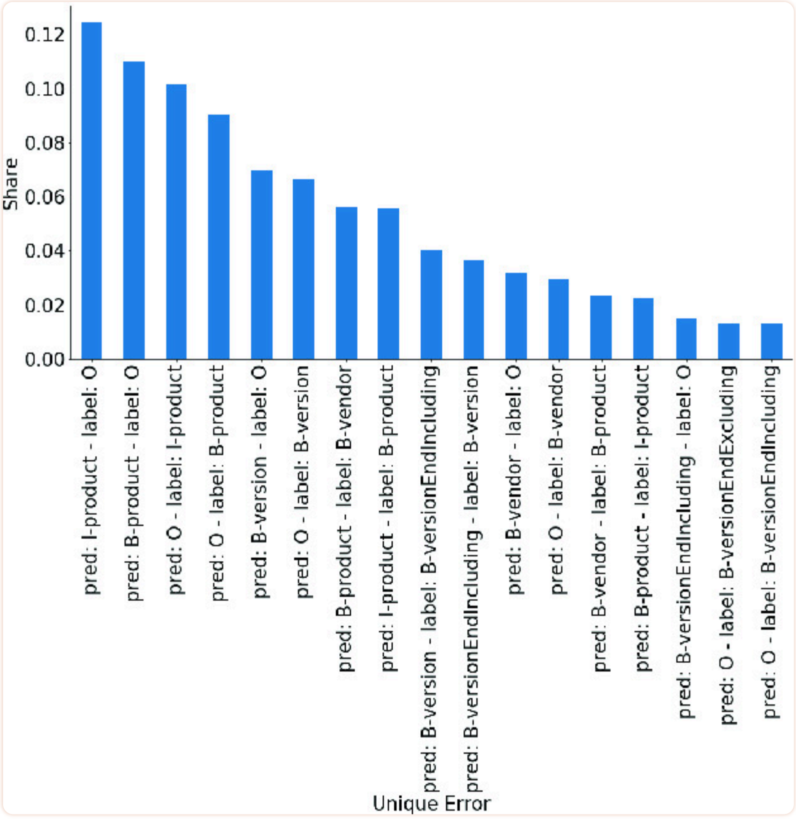


Fig. 9.

Common miss-classifications made by the system. This explains about 90% of the error.

Related Research

Other research has tried more extensive engineering of text-features to extract CPEs from the CVE-summary published by NVD. In [5] the authors mine the target product, vendor, and version from the summary by tokenization of each word much like our case-feature and lexicon-feature to discover punctuation, capitalization, and commonly used vendors/products. They also generate snippets (sequence of tokens) to cover multi-word labels through engineered rules based on the feature vector. Multiple token-sequences can then be grouped into a CPE (vendor-product-version link) based on rules, such as that all version tokens that are within 6 tokens of a product token are assigned to that product token. The context of each version is analyzed to determine the version type (before/after, including/excluding). The authors achieve an F-measure of 0.70 (precision: 0.71, recall: 0.69), which we significantly outperforms as we attain an F-measure of 0.86 (precision: 0.857, recall: 0.864).

A similar system of finding CPEs to “one day”-vulnerabilities was proposed in [4], where the authors use a key-word based technique with TF-IDF to find the probability of each word being assigned to a certain sub-class within a CPE. The output of the model is an ordered list of words with a high probability of being a relevant word in a CPE. The authors results may not be comparable to our research, as their system is not intended for automated use and needs explainability. Although, to make a fair comparison we compare their results of the precision of the top predicted word in each ordered list, which is just below 0.6. Our system achieves a higher precision of 0.857. Still, their research indicates that a TF-IDF implementation of a lexicon feature could provide additional performance to our system in terms of finding already mentioned products and vendors.

The model is largely based upon [12], which combined engineered features, a BLSTM-network and a CRF-layer to perform NER on the CoNLL 2003 [21] dataset. They achieve an F-measure score as high as 0.9121, which to our knowledge held the state-of-the-art during some time in 2016. Results from different datasets are not comparable, as the quality and the general challenge of each dataset may be different. Other, more recent, implementations of state-of-the-art NLP-models were implemented in our research such as BERT [3], but with a significant decrease in performance compared to our model.

Conclusion

Our research concludes that it is possible to automate CPE-labeling of CVEs with machine learning with high precision and recall, in regards to the CPEs that are actually mentioned in the CVE-summary. Our model is able to find CPE-products, -vendors, and -versions with an F-measure 0.8604 (precision: 0.8571, recall: 0.8637) through NER-tagging, and completely reconstruct all corresponding CPEs in 67.44% of CVE-summaries. This system enables DVM-tools to automatically and without time-lag get an estimate of some CPEs a particular CVE describes and thereby reduce the risk of becoming a victim of a “one day”-vulnerability. Additionally, CPEs may also be found in incorrectly labeled CVEs or from vulnerabilities from other sources, such as forums, email-threads, or RSS-feeds. To our knowledge, our results establish a new state-of-the-art in extracting CPEs from CVE-summaries.

The system could be further developed by embedding knowledge of the available universe of CPEs into the results of the prediction so that each estimated CPE could be pared to one or multiple existing CPEs. A TF-IDF or n-grams implementation of the security lexicon feature, as in [4], could also improve the performance of the system, possibly also taking advantage of a security-lexicon, which in our case brings no noteworthy additional performance.

Acknowledgements

This work was financially supported by the Swedish Foundation for Strategic Research, grant RIT17-0035.

Contributor Information

Clémentine Maurice, Email: clementine.maurice@irisa.fr.

Leyla Bilge, Email: Leyla.bilge@nortonlifelock.com.

Gianluca Stringhini, Email: gian@bu.edu.

Nuno Neves, Email: nuno@di.fc.ul.pt.

Emil Wåreus, Email: emil.wareus@eit.lth.se.

Martin Hell, Email: martin.hell@eit.lth.se.

References

1. Bengio Y, Simard P, Frasconi P. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks.* 1994;5(2):157–166. doi: 10.1109/72.279181. [PubMed] [CrossRef] [Google Scholar]

2. Chiu JP, Nichols E. Named entity recognition with bidirectional LSTM-CNNs. *Trans. Assoc. Comput. Linguist.* 2016;4:357–370. doi: 10.1162/tacl_a_00104. [CrossRef] [Google Scholar]

3. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. CoRR abs/1810.04805 (2018). http://arxiv.org/abs/1810.04805

4. Elbaz, C., Rilling, L., Morin, C.: Automated keyword extraction from "one-day" vulnerabilities at disclosure. Research Report RR-9299, Inria Rennes - Bretagne Atlantique, November 2019. https://hal.inria.fr/hal-02362062

5. Glanz, L., Schmidt, S., Wollny, S., Hermann, B.: A vulnerability's lifetime: enhancing version information in CVE databases. In: Proceedings of the 15th International Conference on Knowledge Technologies and Data-Driven Business, i-KNOW 2015. Association for Computing Machinery, New York (2015)

6. Graves, A., Schmidhuber, J.: Framewise phoneme classification with bidirectional LSTM networks. In: Proceedings of the 2005 IEEE International Joint Conference on Neural Networks, vol. 4, pp. 2047–2052, July 2005 [PubMed]

7. Hochreiter S, Schmidhuber J. Long short-term memory. *Neural Comput.* 1997;9:1735–1780. doi: 10.1162/neco.1997.9.8.1735. [PubMed] [CrossRef] [Google Scholar]

8. John Lafferty, A.M., Pereira, F.C.: Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data, pp. 282–289, June 2001

9. Kaul, P., Golovin, D., Kochanski, G.: Google Cloud, August 2017. https://cloud.google.com/blog/products/gcp/hyperparameter-tuning-cloud-machine-learning-engine-using-bayesian-optimization

10. Khurana, D., Koli, A., Khatter, K., Singh, S.: Natural Language Processing: State of The Art, Current Trends and Challenges (2017). http://arxiv.org/abs/1708.05148 [PMC free article] [PubMed]

11. Kingma, D., Ba, J.: Adam: a method for stochastic optimization. In: International Conference on Learning Representations, December 2014

12. Ma, X., Hovy, E.: End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Berlin, Germany, pp. 1064–1074. Association for Computational Linguistics, August 2016

13. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. In: Proceedings of Workshop at ICLR 2013, January 2013

14. NIST, National Institute of Standards and Technology: Common Platform Enumeration: Naming Specification, Version 2.3, NIST Interagency Report 7695 (2011)

15. NIST, National Institute of Standards and Technology: National Vulnerability Database (2019). https://nvd.nist.gov

16. NIST, National Institute of Standards and Technology: Official Common Platform Enumeration (CPE) Dictionary (2020). https://nvd.nist.gov/products/cpe

17. Pascanu, R., Mikolov, T., Bengio, Y.: Understanding the exploding gradient problem. CoRR abs/1211.5063 (2012). http://arxiv.org/abs/1211.5063

18. Pennington, J., Socher, R., Manning, C.: Glove: global vectors for word representation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, pp. 1532–1543. Association for Computational Linguistics, October 2014. https://www.aclweb.org/anthology/D14-1162

19. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 2014;15:1929–1958. [Google Scholar]

20. The MITRE Corporation: Common Vulnerabilities and Exposures (2019). https://cve.mitre.org

21. Tjong Kim Sang, E.F., De Meulder, F.: Introduction to the CoNLL-2003 shared task: language-independent named entity recognition. In: Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4, CONLL 2003, USA, pp. 142–147. Association for Computational Linguistics (2003)