Automation of Vulnerability Information Extraction using Transformer-Based Models of Natural Language Processing [⋆]

Fateme Hashemi Chaleshtori, Indrajit Ray, and Indrakshi Ray

Colorado State University, Fort Collins CO 80523, USA
{fatemeh, indrajit.ray, indrakshi.ray}@colostate.edu

Abstract. A security attack typically exploits one or more vulnerabilities in the system. We need to recognize and mitigate vulnerabilities as quickly and as extensively as possible to prevent security breaches. Consequently, companies often store vulnerabilities information expressed in natural language and share them with other stakeholders. Dissemination of this information in a structured and unambiguous format in a timely manner is critical to prevent losses due to security attacks. Towards this end, researchers have worked in automated extraction of vulnerability information expressed in natural language. Proposed solutions utilize rule-based strategies like Pattern Matching and Part-of-Speech Tagging, Machine Learning models based on Conditional Random Fields (CRF), and hybrid models using the NLP techniques and pattern recognition to form a semi-automated method. We propose an alternative approach that uses Transformer-based models for vulnerability extraction. Such models are easy to use, fully automated, and can work with any existing or new databases where vulnerability information is expressed in natural language. We propose the use of the Transformer models such as BERT, which has outperformed other models in various NLP tasks, in an end-to-end neural architecture for Named Entity Recognition (NER) to extract security-related terms from the text. We demonstrate our results on other language representation models similar to BERT that are based on Transformers. We fine-tune these models on the labeled data from vulnerability databases, for the task of NER so that we can automatically extract security-related words and phrases from a vulnerability description, such as software name and version, file name, and operating system. Our method outperformed CRF-based models and eliminates the need for feature selection. Moreover, it can detect new information from vulnerabilities where the description text patterns do not match those specified by rule-based systems.

Keywords: Cybersecurity · Vulnerability · Information Extraction · Natural Language Processing · BERT.

[⋆] This work was supported in part by funds from NIST under award number 60NANB18D204, and from NSF under award number CNS 2027750, CNS 1822118 and from NIST, Statnett, Cyber Risk Research, AMI, and ARL.

1 Introduction

Cyber-attacks are now becoming more frequent and damaging, and forestalling them has become an uphill struggle. Learning about the potential vulnerabilities of a system and attempting to block all possible ways to exploit these vulnerabilities lessens the likelihood of experiencing cyber-attacks. Information about vulnerabilities, exploits, user actions, system specifications and settings, and the attacks are usually disclosed in the form of text sources like mailing lists initially, before being included in vulnerability databases. Security analysts must use this information to investigate whether their own systems are susceptible to comparable vulnerabilities. Such assessment must be done at the earliest to prevent security breaches.

The use of natural language for storing vulnerability information makes it time-consuming for security analysts to use this information. Towards this end, there are efforts in automated extraction of vulnerability information from natural language texts [31, 19, 9, 25, 18, 26].

Information extraction is the task of searching and analyzing unstructured data to discover structured information in it [17]. Rule-based techniques such as Part-of-Speech tagging and Pattern Matching are commonly used for vulnerability information extraction [9, 31, 26]. Such techniques determine linguistic patterns for different terms based on the text grammatical structure and look for them in the text to extract important information units. Although such models can be very accurate on a specific target domain, the downside is that they are labor-intensive and fail to extract key information when the text is in a form that does not adhere to the patterns specified in the rules [6].

One of the most important subtasks of information extraction is Named Entity Recognition (NER). NER is the task of scanning a text to automatically identify major entities in it and classify them into predetermined categories, such as location, person, and organization [13]. NER can help to extract useful and important information from a given text document. It has applications in text categorization, recommender systems, and text summarization. [23]. NER can be used in vulnerability information extraction because it can direct attention to important cyber-security entities in the text. However, the existing NER models [4, 8] do not support domain-specific entities and they need further modification and training.

Multi-class classifiers like Support Vector Machine (SVM) [23, 5] and Conditional Random Field (CRF) [19, 4] are models that are frequently used for addressing the task NER. CRF is a machine learning model used for data classification and is suitable for labeling sequential data like text, where the status of neighbors, such as words surrounding a target word, can affect the predicted class for the target instance [30]. Joshi et al. [19] used the same CRF-based architecture used by Stanford NER [4] and trained it on a manually annotated security dataset to build a domain-specific named entity tagger. Such a method necessitates feature selection which requires domain expertise.

Deep learning models such as Long Short-Term Memory (LSTM) [14, 24, 15] and pre-trained transformed-based language models like BERT have been utilized to improve the task of NER. LSTM processes a sentence word by word and its architecture causes a bias towards the latest words in a text sequence and cannot capture long dependencies [16]. Models like BERT, on the other hand, take the whole sentence as one input rather than word by word, and thus, they focus on all words equally and can better capture the connection between the words. Hence, BERT can achieve a better performance than LSTM on the task of NER [21]. Such language representation models have outperformed all previous models in common NLP tasks such as question answering. Labusch et al. [20] are leveraging the BERT model for solving NER in historical and contemporary German texts.

Proposed Approach. We propose the use of Transformers-based NLP language representation models, such as BERT, for vulnerability information extraction. To fine-tune these models, we use a labeled dataset with over 1800 security-related sentences to fine-tune different Transformer-based models, for the NER task. This data has been gathered from various sources. The vulnerability descriptions are split into words and tagged with eleven different labels (ten security-related and one for non-security-related tokens), resulting in more than 50K tagged entities [19].

We use different variations of language representations that are based on Transformers [29], such as BERT [11], XLNet [32], RoBERTa [22], Electra [10], and DistilBERT [27]. and add a classification layer on top for NER. The above all fall in the class of pre-trained language models. A very important advantage of employing pre-trained language representation models is that they detect word-level and character-level features automatically and this obviates the need for feature engineering.

The advantage of our method is that the proposed model can be applied on any vulnerability dataset without any alteration or modifications. Moreover, in contrast to machine learning models, this one eliminates the need for feature engineering and feature selection because rather than extracting features from text and deciding which ones are appropriate to give to the model as input, the raw text data is given to the model. Subsequently, security analysts can benefit from this model with no assistant from computer science specialists. Moreover, our approach performs better when compared to rule-based methodologies where particular patterns and templates need to be characterized for different vulnerability descriptions.

Our Contributions. We fine-tune several Transformer-based machine learning models for the task of NER in the cyber-security field on a labeled dataset containing more than 1800 sentences and over 50K tokens. Ours is the first work that tests Transformer-based language representation models to address the problem of automatic vulnerability information extraction. The model that we present outperforms the CRF model on security-related NER task. We also introduce a mechanism to ensemble the results of different Transformer-based models by majority voting and this also increases the performance.

Paper Roadmap. In the rest of the paper, we will discuss the related research that has been conducted to solve this or similar problems in section 2; next, we describe the required background knowledge for this word in section 3, then, we describe our proposed approach in section 4 in detail to address with the issue. And finally, we conclude the paper and point out some interesting directions to continue this work in section 5.

2 Related Works

Some earlier works have been done to extract vulnerability information from text. Many of them are focused on a specific task. Schuppenies et al. [28] are trying to automatically extract vulnerability information useful for attack graphs construction. Attack graphs are used for modeling and analyzing complex network security. System information and vulnerability information are required to construct attack graphs. The authors use the extracted vulnerability information with an attack graph tool to build attack graphs.

Russo et al. [26] aim to summarize the vulnerability descriptions and categorize them according to a taxonomy modeled for the industry. The authors have manually inspected 130 common vulnerabilities and concluded that there are some recurrent linguistic patterns in the descriptions that can be used for automating the process of recognizing the relevant and key information and they are trying to detect such patterns in the text. Their approach has two main steps: information extraction and vulnerabilities classification. The authors develop a tool named CVErizer that automatically and accurately classifies software vulnerabilities with the help of rule-based pattern recognizing, and also helps vulnerability assessors understand software vulnerabilities by providing summaries of their descriptions.

Jones et al. [18] proposed a bootstrapping semi-supervised NLP model, which needs limited labeled training data, to extract security entities and their relationships from the text. The goal is to extract information in the form of relations having syntax ⟨ *subject entity*, *predicate relation*, *object entity* ⟩. An example of such a relation is ⟨ *software vendor*, *is a vendor of*, *software product* ⟩. They searched the text corpus for mentions of a few known seed instances, and when an instance is found, the model generates some patterns from the surrounding text automatically. This work focuses on relation extraction from text, whereas, our objective is NER.

Using data from the National Vulnerability Database and other text sources, Joshi et al. [19] proposed a model to provide a data representation of cybersecurity-related notions associated with vulnerability descriptions. The authors are using the data they collected and manually annotated to train a Conditional Random Field-based classifier with the Stanford Named Entity Recognizer [12, 4]. To evaluate the model, they used five-fold cross-validation. On average, they achieved a precision score of 83%, a recall score of 76%, and an F1-score of 80%. The major benefit of using machine learning models is that they automate the process of extracting important information and they can perform well on various input data with no change. We use the dataset provided by this work but instead of the CRF model, we are leveraging the Transformer-based language representation models to address the same problem.

Weerawardhana et al. [31] also try to extract key security-related terms from vulnerability databases. The authors are presenting two models for vulnerability information extraction: a machine learning module based on CRF and another module that benefits from Part-of-Speech tagging. For the machine learning module, they are following the same approach as in [19] for feature selection and training the model. The Part-of-Speech tagging model is rule-based and the goal is to detect grammatical patterns in the text. They are defining a set of patterns for software names and versions, file names, modifiers, vulnerability types, etc. and then, they try to find security terms that match these patterns. The latter model has outperformed the CRF-based model. The limitation of this approach is that this model may fail if a new style or pattern is introduced in a vulnerability description that does not match the predefined patterns.

Ramnani et al. [25] target security advisories that may contain threat information or vulnerabilities. To extract useful information from text, they are using NLP, semi-supervised pattern identification, and matching methods. They use NLP to first split the text into sentences and then tokenize them. Subsequently, they apply their pattern identification model to that data. This work is not using NLP as a language representation tool. After extraction, they model the data based on the Structured Threat Security Expression model [7].

3 Background

Our work uses Transformer-based language representation models for vulnerability extraction. Transformer-based models have an architecture designed for sequence-to-sequence tasks where the goal is to transform a sequence of objects into another sequence, for example, the task of natural language text translation. Transformers are building blocks of language representation models that learn contextual relations between words by determining the weights assigned to different words in an input sentence depending on their importance for a specific task [11]. We point out the important features of transformer-based models that we utilize in our experiments and highlight their main differences.

BERT [11] is an open-source pre-trained deep learning model published by Google in October 2018. BERT benefits from transformer blocks to produce vector representations for the input sentence. BERT has been trained on a huge corpus of data and optimized on two unsupervised NLP tasks that do not need labeled data, Next Sentence Prediction, and Masked Language Modeling. These are relatively difficult tasks in NLP and the model should comprehend the text to solve them. After training, the model is ready for other tasks like NER. As training it for a domain-specific purpose needs a very large amount of data and is computationally expensive, researchers only fine-tune it on smaller datasets that are accessible more easily.

The RoBERTa model [22] was proposed by Facebook and the University of Washington in July 2019. RoBERTa is trained on a much larger dataset than BERT. Note that, BERT is trained on the English Wikipedia and BookCorpus [33] of size 16G, whereas the dataset used for training RoBERTa contains CC-NEWS data (76G), OpenWebText data (38G), and Stories (31G) as well as the English Wikipedia and BookCorpus (16G). Additionally, it has been optimized for the task of Masked Language Modeling (MLM) with a Dynamic Masking Pattern instead of a Static Masking Pattern. Masked Language Modeling is an NLP task in which some of the tokens in the sentence are masked and the model attempts to predict what those tokens are and reproduce them. This helps the model understand the connection between the tokens in a sentence. With a Static Masking Pattern, the tokens that are veiled are chosen randomly yet the sets of masked and unmasked tokens continue as before during the training phase. On the contrary, Dynamic Masking Pattern picks a new set of tokens to mask whenever a new sequence is fed to the model [22]. Another difference between the BERT model and the RoBERTa model is that RoBERTa does not train for the task of Next Sentence Prediction.

XLNet [32] is another transformer-based language representation model published by Carnegie Mellon University and Google AI Brain Team in Jan 2020. The XLNet model presents permutation language modeling to improve the training, in which all tokens are predicted but in random order. This assists the model with learning bidirectional connections and consequently, it better handles dependencies and relations between words. XLNet was trained on a very large data corpus with more than 130 GB of text data which is much bigger compared to the volume of data used for training the BERT model.

The DistilBERT model [27] is proposed by Hugging Face in March 2020. It is a 40% smaller model compared to BERT (BERT has 110M parameters and DistilBERT has 66M parameters.) but with the same general architecture. As it is smaller, it is 60% faster and has achieved almost the same performance as BERT in the language understanding tasks.

The Electra model [10] is published by Stanford University and Google Brain in March 2020. This model is proposing a task for training, similar to Masked Language Modeling but more efficient, named Replaced Token Detection. In comparison to the BERT model that is a generative model and tries to reconstruct the original token based on the unmasked tokens, the Electra model is using a discriminative module only. The input data to this model is corrupted instead of masked, meaning that some of the tokens are replaced with acceptable alternatives. The model's objective is to predict whether each token is original or replaced. The Masked Language Modeling technique only learns from the masked tokens – typically 15% of the data – while this discriminator module is using all tokens. The idea behind training the Electra model is similar to Generative Adversarial Network (GAN) in that both are trying to distinguish between the original tokens and those that have been replaced, and thus, the loss value is calculated over all tokens of the input rather than just masked tokens.

4 Our Approach

In this section, first, we talk about the dataset that we use for fine-tuning the models, its arrangement and content, and the steps we take to clean and set up the data to feed to the models. Afterward, we discuss the proposed approach for the vulnerability information extraction task and the experiments we carried out, and finally, we discuss the results and compare them to other approaches.

4.1 Data Preparation

We use a dataset of vulnerability descriptions, obtained from [19]. This dataset contains vulnerability information from the Common Vulnerabilities and Exposures (CVE) database [2], Microsoft Security Bulletins [3], Adobe Security Bulletins [1], and several security blog posts. There are individual files for each vulnerability description. In this dataset, sentences have been split into tokens (words and symbols), and the tokens are manually annotated with 10 labels by twelve Computer Science students. Annotation guidelines are not available in the paper. The dataset has nearly 50K tokens and 5K entities that have security-related labels. Each token is assigned one label. In case it isn't security-related, it gets the 'O' label, else, it gets a pertinent security label. In the following, we discuss these labels.

OPERATINGSYSTEM Tags the name of the operating systems. Examples include Ubuntu 16.0, Windows 10, and Macintosh.

CONSEQUENCES Refers to the final results of the attack. Examples include arbitrary SQL commands, device crashes, and sensitive information leak.

SOFTWARE Refers to the name of a software. Examples include System.Net, Outlook, and Visual Studio.

MODIFIER Tags the version of a software or operating system that follows the name of the software or operating system. Examples include 8.0.3 or before 2.4.1.

NETWORK Refers to network related terms. Examples include SSL, TCP, and HTTPS.

ATTACK It marks the terms that are closely related to the attack. Examples include man-in-the-middle, brute-force, and phishing.

MEANS Tags the way to attack. Examples include crafted website, unspecified vectors, and SQL injection.

HARDWARE Hardware devices involved in the attack. Examples include CPU, processor cores, and server.

FILE Files involved in the attack. Examples include restorer.php, index.php, and EOSDataServer.exe.

OTHER Other technical terms that cannot be put in other security-related categories. Examples include webpage and administrative user rights.

O Non-security-related tokens.

The following steps are done to pre-process the dataset.

Step 1: Fix the inconsistencies between the labels. Some examples in the dataset had incomplete labels or had typographical errors. For example, in some cases *OPERATING* was used instead of *OPERATINGSYSTEM*; the two labels *OTHER* and *OTHER_TECHNICAL_TERMS* refer to the same category. Sometimes extra characters are at the end of some of the labels, such as in *SOFTWAREr*. We fixed such irregularities by partial term matching, using some of the initial characters of the labels that make them unique. We looked for the primary characters and supplanted the whole label with the right label.

Step 2: Remove the labels and concatenate the tokens to form a full text and split the text into sentences. The Transformer-based models in our experiments require data in a format that has two columns, *token* and *label*. It should be clear where a sentence starts and where it ends. In the dataset from [19], there was no separation between sentences. We used the Python Natural Language Toolkit (NLTK) [8] to break the data into sentences instead of applying fixed rules like considering punctuation symbols only or doing it manually because the NLTK library does this job more accurately. We first remove the labels and connect the tokens to one another by inserting a space in between. Then, we employ the *tokenize.sent_tokenize* function from the NLTK library and it returns a list of sentences. This function is not sensitive to additional spaces before punctuation marks, and it can deal with those, so, the concatenating spaces that appear before punctuation symbols will not cause any trouble.

Step 3: Identify the end of sentences and insert a new line. Based on the list that the *tokenize.sent_tokenize* function gives out, we separate the sentences by adding one new line at the end of each sentence.

Step 4: Reinsert the labels. Once the sentences are separated, we add the labels to the corresponding tokens and transfer the data into the desired format of two columns, namely, *token* and *label*.

Step 5: Accumulate the data from different files into one file. Finally, we gather all the information from the various polished documents in the dataset into one record.

Now, the format is suitable and the data is ready to be utilized for the fine-tuning phase.

4.2 Application of NER

Regular NER tools have a restricted and general set of labels to search for in a given text. For example, the Stanford NER [4] supports only three labels: *Person*, *Location*, and *Organization*. NLTK NER tool [8] supports six more labels, which are *Date*, *Time*, *Money*, *Percent*, *Facility*, and *GPE*. Note that, none of these labels are security-related. To address this shortcoming, we can consider a new set of labels, as mentioned in Section 4.1, for our domain-specific NER.

We are treating the NER task as a classification task. The important point to note is that each word will get a tag in a sentence; this implies that the relation

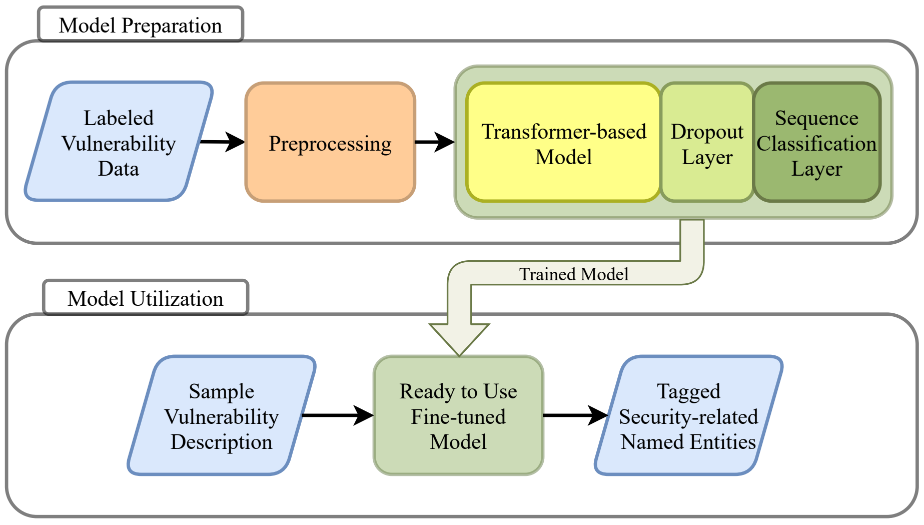
Model Preparation
Labeled Vulnerability Data
Preprocessing
Transformer-based Model
Dropout Layer
Sequence Classification Layer
Trained Model
Model Utilization
Sample Vulnerability Description
Ready to Use Fine-tuned Model
Tagged Security-related Named Entities

Fig. 1. Proposed model diagram

of a word with the other words in the same sentence is captured in our model. The training data that we use has a consistent format; each token is assigned a label and the model knows the sentence that each word belongs to in light of the fact that the sentences are separated in the dataset.

Figure 1 illustrates the overall structure of our proposed model. Transformer-based models for NLP are general-purpose language representation models incorporating special types of deep neural networks, known as Transformers, and are trained on a huge volume of unlabeled data to comprehend natural language text. These models take text as their input, tokenize it into words and symbols, and then produce vector representations for them.

To boost the performance of these models and make them produce more meaningful representations for domain-specific words (security-related words and labels in this case), we fine-tune our network on the security-related dataset described earlier, for the particular task of NER. Consequently, a *Dropout Layer* and a *Sequence Classification Layer* are added on top of the Transformer-based model. The use of dropout layers is a common practice in designing neural network architectures to prevent overfitting. This layer randomly shuts down some of the neurons in the network to prevent the network from memorizing the input/output relations. The sequence classification is a linear neural network layer that transforms the vector representation of each token into 11 scores, one for each of the security-related NER labels. Then, the label corresponding to the maximum score is considered to be the predicted NER tag for that token.

We try different Transformer-based models, in addition to the BERT model, to discover which one is performing better. Models with different training data and different architectures, namely, BERT [11], RoBERTa [22], XLNet [32], DistilBERT [27], and Electra [10] have been fine-tuned in our experiments.

4.3 Experiments

Fine-tuning the Transformer-based models with cross-validation. This experiment aims to figure out if the results obtained by the experiments that have been done in [19] and [31] can be improved. We use the same dataset as in [19] to fine-tune different models and to have a fair comparison of the models. Similar to [19], we apply five-fold cross-validation for fine-tuning the models. Now, we need to adjust the model configuration parameters. We tried different learning rates and various numbers of epochs to learn the best setting for each model. A specific set of hyperparameters leads to the best performance for different models even with the same architecture. Table 1 shows the most suitable hyperparameters for fine-tuning each model. We utilized a linear learning rate schedule, a maximum sequence length of 512, and the Adam optimizer algorithm for all models.

Table 1. Best hyperparameters for each model

	LearningRate	**#Epochs**	**BatchSize**
RoBERTa-Base	1.00e-5	8	8
RoBERTa-Large	2.00e-5	8	8
BERT-Base-Cased	1.00e-4	8	8
BERT-Large-Cased	9.00e-6	12	8
Electra-Small-Discriminator	5.00e-5	8	8
DistilBERT-Base-Cased	5.00e-5	8	8
XLNet-Base-Cased	1.00e-4	8	8
XLNet-Large-Cased	5.00e-6	10	4

Table 2 presents the evaluation metrics achieved by five-fold cross-validation training. For each fold, 20% of the data is saved for testing and the rest is utilized for training. Since the data is unbalanced and the number of instances in each class is different, to drop the adverse consequence of the model's low performance on classes with fewer samples and not to overvalue the high performance of the model for classes with more samples, we report the micro average (weighted average), rather than the macro average (equally treating all classes), for precision, recall, and F1 score to have a fair demonstration of the models' performance. As table 2 reveals, the XLNet-Base-Cased model has the best performance among the seven models that we tested.

Merging the results of different models by majority voting. Another experiment that we carried out is fine-tuning the models with the same configurations and hyperparameters but on 90% of the data and we keep only 10% for testing. The results for this experiment are shown in table 3. The XLNet-Base-Cased model is outperforming the other models regarding the recall and F1 score. It obtains scores of 81.94%, 87.5%, and 84.63% for precision, recall, and F1 score respectively.

Table 2. Results of the five-fold cross-validation experiment

Model	**Metric**	**1st Fold**	**2nd Fold**	**3rd Fold**	**4th Fold**	**5th Fold**	**Average**
RoBERTa-Large	Precision	83.94	82.69	85.23	66.53	80.52	82.29
	Recall	73.48	83.08	83.87	68.02	85.27	80.91
	F1 score	78.36	82.88	84.54	67.27	82.83	81.51
BERT-Base	Precision	75.48	79.99	85.33	64.45	82.16	80.07
	Recall	80.42	82.95	82.84	66.8	85.5	81.99
	F1 score	77.87	81.44	84.07	65.61	83.8	80.99
BERT-Large-Cased	Precision	74.36	79.52	82.25	60.67	79.58	78.14
	Recall	81.22	84.01	85.72	69.64	84.03	83.08
	F1 score	77.64	81.7	83.95	64.84	81.74	80.52
DistilBERT-Base-Cased	Precision	75.16	81.8	85.83	60.44	82.69	80.5
	Recall	80.59	81.21	83.87	66.19	84.15	81.53
	F1 score	77.78	81.5	84.84	63.19	83.41	80.98
Electra-Base-Discriminator	Precision	73.44	77.15	85.05	58.61	79.02	77.83
	Recall	79.8	77.21	84.43	63.36	81.67	79.77
	F1 score	76.49	77.18	84.74	60.89	80.32	78.76
XLNet-Large-Cased	Precision	80.3	83.08	85.06	69.35	83.93	82.43
	Recall	80.48	82.91	86.41	62.75	82.15	82.07
	F1 score	80.39	83	85.73	65.89	83.03	82.24
XLNet-Base-Cased	Precision	79.46	82.47	86.59	65.77	84.87	**82.49**
	Recall	82.36	85.2	86.62	69.23	85.92	**84.24**
	F1 score	80.88	83.81	86.61	67.46	85.39	**83.35**

To further improve the performance, we stack three best-performing models considering the F1 score, which are XLNet-Base-Cased, XLNet-Large-Case, and RoBERTa-Large, and apply the majority voting between the output labels. Figure 2 depicts this technique. For each sentence that is given to the three models as input, we gather the labels that they produce for every token. Then we check if at least two out of the three models agree on the label allocated to a token. In that case, the majority voting module outputs that label for that particular token. On the off chance that each of the three models is predicting different labels for a token, we assume that this token is non-security-related and assign the label 'O' to it. The class-wise performance of this model is represented in table 4.

4.4 Discussion of Results

We select the best-performing model based on the recall metric on the ground that in this context, it may be desirable to find all of the security terms in a vulnerability description even though this means the output will be noisy and the model will incorporate non-security-related terms in the output. In other words, it is smarter to discover and gather all of the key terms from a given vulnerability description than to miss some of them while the output is more precise.

Table 3. Results of fine-tuning the models on 90% of data without cross-validation

Model	Precision	Recall	F1 score
RoBERTa-Large	81.25	85.35	83.25
BERT-Base	80.48	85.35	82.84
BERT-Large-Cased	77.38	86.92	81.87
DistilBERT-Base-Cased	80.45	85.84	83.06
Electra-Base-Discriminator	77.66	82.28	79.9
XLNet-Large-Cased	**82.14**	87.17	84.58
XLNet-Base-Cased	81.94	**87.5**	**84.63**

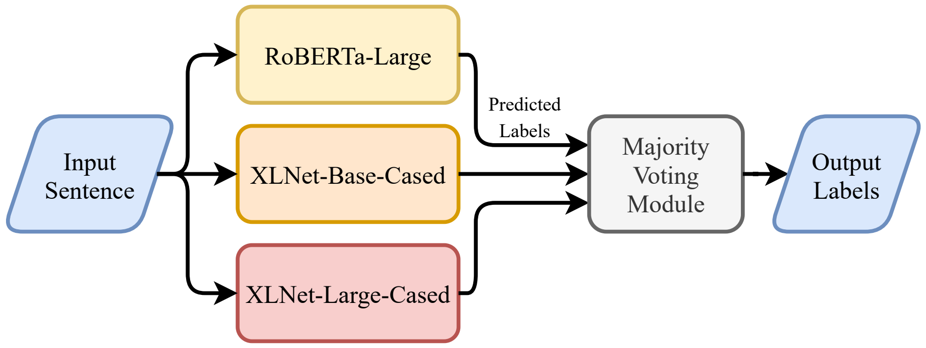

Fig. 2. Structure of the majority voting method to combine the results of three best-performing models

Considering the average result of five runs in the cross-validation training, the XLNet-Base-Cased model is achieving better results compared to other models in view of all three metrics.

The results reported in [19] show that the CRF classifier can achieve the values of 83%, 76%, and 80% for precision, recall, and F1 score respectively for the task of NER, adjusted for vulnerability information extraction. The XLNet-Base-Case model, fine-tuned on the same data with cross-validation technique, outperforms this model by obtaining a score of 82.49% for precision, 84.24% for recall, and 83.35% for F1 score. The improvement in the recall metric is significant. The recall score that the XLNet-Base-Cased model achieves is 8.24% higher than the CRF model.

The method presented in the second experiment that merges the outputs of three models achieves a precision score of 85.5%, a recall score of 87.83%, and an F1 score of 86.65%. This indicates that compared to the best-performing model fine-tuned on 90% of the data, the XLNet-Base-Case model, the ensemble model with majority voting increased the average precision score by 3.56%, the average recall score of 0.33%, and the average F1 score by 2.02%.

Weerawardhana et al. [31] analyzed the performance of the CRF-based NER framework introduced in [19] compared with their machine learning model which

Table 4. Class-wise performance of the majority voting model

CLass	Precision	Recall	F1 score	Support
SOFTWARE	91.45	93.09	92.26	666
OPERATINGSYSTEM	89.83	95.78	92.71	166
CONSEQUENCES	80.32	91.89	85.71	111
OTHERS	65.14	66.36	65.74	107
MEANS	69.23	66.32	67.74	95
MODIFIER	90.24	82.22	86.05	45
NETWORK	38.46	41.67	40	12
FILES	100	100	100	4
ATTACK	0	0	0	1
HARDWARE	0	0	0	1
Micro Average	85.5	87.83	86.65	
Macro Average	62.47	63.73	63.02	

has the same architecture and is based on CRF as well, on the labels that are common between these two models, which are *FILE*, *MODIFIER*, *OPERATING SYSTEM*, *SOFTWARE*, and *CONSEQUENCES* (referred as *IMPACT* in [31]) in spite of the fact that the data corpus used for training the models in these two works are different. Their ML model has similar performance to the model proposed in [19]. Then, they compared their machine learning model with the Part-of-Speech (PoS) tagging approach which emphasizes the grammatical structure of the text, both trained on the same dataset. Their results indicate that PoS tagging performs better than the machine learning approach, considering precision and F1 score. Table 5 compares the PoS tagging model with our method that ensembles best-performing models by applying majority voting on the predicted labels. In this comparison, we are only referring to the labels that both methods support. For three out of the five labels, our model is obtaining better scores and the macro average of all metrics in five classes shows that our model is outperforming the PoS tagging approach.

Other than the performance, another viewpoint to think about for comparing these models is that in Part-of-Speech tagging, a specialist ought to define befitting guidelines, rules, and linguistic patterns to extract information in each security category while our approach is ready to use. This implies that when the number of categories grows, new patterns and structures should be set, and furthermore, the vulnerability description text style or format is liable to change and old patterns may not succeed in finding correct terms in an unfamiliar text. As such, when a set of patterns functions admirably for a particular vulnerability dataset, it probably will not perform well on another. This is the place where machine learning models in general, and our model in particular, can do much better as the building block of it, Transformer-based language representation models, have already been trained on huge corpora of text data and no change is required to adopt the model to the new inputs.

Table 5. Comparing the performance of our approach and POS tagging approach proposed in [31]

	Ensemble Transformer-based Models with Majority Voting			Part-of-Speech Tagging		
Class	**Precision**	**Recall**	**F1 score**	**Precision**	**Recall**	**F1 score**
FILE	100	100	100	72	96	81
CONSEQUENCES	80.32	91.89	85.71	94	80	86
OPERATINGSYSTEM	89.83	95.78	92.71	26	90	40
SOFTWARE	91.45	93.09	92.26	71	77	74
MODIFIRE	90.24	82.22	86	99	97	98
Macro Average	90.37	92.6	91.34	72.4	88	75.8

5 Conclusion and Future Directions

We propose a deep learning model to automatically extract vulnerability information from the natural language vulnerability description text. This model is a NER system that uses transformer-based language representation models and extracts security-related words and phrases from the text. We train this system on domain-specific data which enables it to recognize security-related entities.

We use a real-world labeled vulnerability dataset containing 50K tokens for both training and evaluation. Our results demonstrate 8.24% and 3.35% improvements in the average recall and F1 score by our best-performing model, XLNet-base-Cased, over the older approaches in the literature that are based on Conditional Random Fields NER systems. Other than the progress in performance, this method removes the need for feature engineering because the input data is the raw text. Also, unlike the rule-based method such as Pattern Matching and Part-of-Speech tagging, new text styles are not a challenge for this model and it can perform well on any vulnerability dataset.

We also propose an ensemble model based on our three top-performing models. To evaluate this method, in another experiment we fine-tuned all models on 90% of the dataset and tested them on the remaining 10%. This ensemble model applies a majority voting mechanism to the predictions of XLNet-Base-Cased, XLNet-Large-Cased, and RoBERTa-Large. It introduced a 2.02% improvement in F1 score over the best single model.

In addition to NER, Co-reference Resolution also can be incorporated to extract more information from a vulnerability description. Co-reference Resolution intends to find all terms that are referring to a specific entity and normalize them (use the same term for all the references)[13]. For example, discovering what entity each pronoun is referring to and replacing it with that entity. We consider adding such a component to our proposed model to have a more thorough information extraction system.

References

1. Adobe security bulletins and advisories. https://helpx.adobe.com/security/security-bulletin.html, accessed: 2021-07-26
2. Cve database. https://cve.mitre.org, accessed: 2021-07-26
3. Microsoft security bulletins. https://docs.microsoft.com/en-us/security-updates/securitybulletins/securitybulletins, accessed: 2021-07-26
4. Stanford named entity recognizer (ner). https://nlp.stanford.edu/software/CRF-NER.html, accessed: 2021-07-25
5. AbdelRahman, S., Elarnaoty, M., Magdy, M., Fahmy, A.: Integrated machine learning techniques for arabic named entity recognition. IJCSI **7**(4), 27–36 (2010)
6. Aggarwal, C.C., Zhai, C.: Mining text data. Springer Science & Business Media (2012)
7. Barnum, S.: Standardizing cyber threat intelligence information with the structured threat information expression (stix). Mitre Corporation **11**, 1–22 (2012)
8. Bird, S., Loper, E., Klein, E.: Natural Language Processing with Python. O'Reilly Media Inc., Sebastopol, CA, USA (2009), https://www.nltk.org/
9. Bridges, R.A., Jones, C.L., Iannacone, M.D., Testa, K.M., Goodall, J.R.: Automatic labeling for entity extraction in cyber security. arXiv preprint arXiv:1308.4941 (2013)
10. Clark, K., Luong, M.T., Le, Q.V., Manning, C.D.: Electra: Pre-training text encoders as discriminators rather than generators. arXiv preprint arXiv:2003.10555 (2020)
11. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
12. Finkel, J.R., Grenager, T., Manning, C.D.: Incorporating non-local information into information extraction systems by gibbs sampling. In: Annual Meeting of the Association for Computational Linguistics. pp. 363–370 (2005)
13. Golshan, P.N., Dashti, H.R., Azizi, S., Safari, L.: A study of recent contributions on information extraction. arXiv preprint arXiv:1803.05667 (2018)
14. Gunawan, W., Suhartono, D., Purnomo, F., Ongko, A.: Named-entity recognition for indonesian language using bidirectional lstm-cnns. Procedia Computer Science **135**, 425–432 (2018)
15. Hammerton, J.: Named entity recognition with long short-term memory. In: Natural language learning conference at HLT-NAACL 2003. pp. 172–175 (2003)
16. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
17. Jiang, J.: Information extraction from text. In: Mining text data, pp. 11–41. Springer (2012)
18. Jones, C.L., Bridges, R.A., Huffer, K.M., Goodall, J.R.: Towards a relation extraction framework for cyber-security concepts. In: Annual Cyber and Information Security Research Conference. pp. 1–4 (2015)
19. Joshi, A., Lal, R., Finin, T., Joshi, A.: Extracting cybersecurity related linked data from text. In: 2013 IEEE International Conference on Semantic Computing. pp. 252–259. IEEE (2013)
20. Labusch, K., Kulturbesitz, P., Neudecker, C., Zellhöfer, D.: Bert for named entity recognition in contemporary and historical german. In: Conference on Natural Language Processing, Erlangen, Germany. pp. 8–11 (2019)

21. Li, J., Sun, A., Han, J., Li, C.: A survey on deep learning for named entity recognition. IEEE Transactions on Knowledge and Data Engineering (2020)
22. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692 (2019)
23. Mansouri, A., Affendey, L.S., Mamat, A.: Named entity recognition approaches. International Journal of Computer Science and Network Security **8**(2), 339–344 (2008)
24. Na, S.H., Kim, H., Min, J., Kim, K.: Improving lstm crfs using character-based compositions for korean named entity recognition. Computer Speech & Language **54**, 106–121 (2019)
25. Ramnani, R.R., Shivaram, K., Sengupta, S.: Semi-automated information extraction from unstructured threat advisories. In: Innovations in Software Engineering Conference. pp. 181–187 (2017)
26. Russo, E.R., Di Sorbo, A., Visaggio, C.A., Canfora, G.: Summarizing vulnerabilities' descriptions to support experts during vulnerability assessment activities. Journal of Systems and Software **156**, 84–99 (2019)
27. Sanh, V., Debut, L., Chaumond, J., Wolf, T.: Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108 (2019)
28. Schuppenies, R., Meinel, C., Cheng, F.: Automatic extraction of vulnerability information for attack graphs. Hasso-Plattner-Institute for IT Systems Engineering, University of Potsdam (2009)
29. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. arXiv preprint arXiv:1706.03762 (2017)
30. Wallach, H.M.: Conditional random fields: An introduction. Technical Reports (CIS) p. 22 (2004)
31. Weerawardhana, S., Mukherjee, S., Ray, I., Howe, A.: Automated extraction of vulnerability information for home computer security. In: International Symposium on Foundations and Practice of Security. pp. 356–366. Springer (2014)
32. Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., Le, Q.V.: Xlnet: Generalized autoregressive pretraining for language understanding. arXiv preprint arXiv:1906.08237 (2019)
33. Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., Fidler, S.: Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In: IEEE international conference on computer vision. pp. 19–27 (2015)