Towards end-to-end Cyberthreat Detection from Twitter using Multi-Task Learning

Nuno Dionísio, Fernando Alves, Pedro M. Ferreira and Alysson Bessani
LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal
Email: {ndionisio, falves}@lasige.di.fc.ul.pt, {pmf, anbessani}@ciencias.ulisboa.pt

***Abstract*—Continuously striving for cyberthreat awareness is an essential task to secure an IT infrastructure. Analysts must guarantee access to information on the most up-to-date cybersecurity events and threats. This monitoring process is often the job of a security information and event management system, which relies on the timeliness and relevance of its feeds. There has been growing interest in exploiting open source intelligence for this purpose, mainly due to its timeliness and volume. Social media sites such as Twitter, are capable of aggregating numerous cybersecurity-related sources and act as a stream of information that can be used to feed a cyberthreat intelligence platform. In this paper, we present a multi-task learning approach combining two Natural Language Processing tasks for cyberthreat intelligence. Our pipeline is capable of reading a stream of tweets from a set of Twitter accounts and, through a shared deep neural network architecture, simultaneously identify relevant cybersecurity-related content and extract indicators of compromise therein. We show that in comparison to the traditional independent tasks baseline, one of the tasks achieves a slight F1 score improvement, while the other task is able to maintain its performance. Thus, the proposed approach greatly simplifies the pipeline and the requirements for data and online model adaptation over time, without sacrificing functional performance.**

I. Introduction

Cyber Threat Intelligence (CTI) is a component of cybersecurity which has been receiving an increasing amount of attention, both from CTI researchers in academia as well as from CTI practitioners in Security Operation Centers (SOCs) and in security services providers.

Timely awareness is a challenging task for a SOC, given the numerous attack vectors that may threaten an IT infrastructure. Given a complex and diverse system incorporating all sorts of hardware, software, and different operating systems, the difficulty in acquiring, processing and selecting the relevant cybersecurity information increases dramatically. As an infrastructure scales and new tools are introduced, vulnerabilities and new attack vectors inevitably creep in, leading to an increasing trend of successful attacks [1]. Although there is the option of subscribing a paid service to receive curated feeds, research has shown that Open Source Intelligence (OSINT) provides useful information to create Indicators of Compromise (IoC) [2]–[4].

In previous work [5], we presented a CTI pipeline that relied on two Deep Neural Networks (DNN) laid out sequentially to process Twitter data streams. As these OSINT streams provide large amounts of heterogeneous data produced at a rapid rate, the pipeline goals are: (i) to select only the IT infrastructure-related content, and (ii) to extract relevant security-related information therein to fill an IoC.

Given that both DNNs are trained on the same source of data, they can be combined through a Multi-Task Learning (MTL) approach which would allow for a more generalized representation in the shared layers. MTL is an inductive transfer learning mechanism where a model is trained on multiple tasks, leveraging the knowledge acquired for one to boost the performance of the other [6], [7]. Recent work in NLP has shown that MTL can often boost the performance of state-of-the-art models [8]. MTL methodologies have shown not only to improve results on tasks that share a common domain, but also that by learning multiple related tasks the model improves its generalization capability, greatly reducing chances of overfitting [9].

In this paper, we present a MTL approach that merges the two previous models into an end-to-end pipeline for cybersecurity-centric Natural Language Understanding (NLU). The intended pipeline should be a complete end-to-end DNN architecture with no requirement for feature engineering or extra components in the processing pipeline.

Our MTL CTI pipeline uses Twitter as its OSINT data stream source. This social media platform was chosen due to its ability to act as a natural aggregator of multiple sources [10], and its big data characteristics: large volume of data, the highly diverse pool of users, high accessibility, and timely production of new content. These properties remain true in the cyber security domain, which resulted in a multitude of work being develop over the years [5], [11], [12].

Our tool receives tweets, through the Twitter API, from a predetermined set of accounts which have been selected based on their likelihood of outputting security-related content about a specified IT infrastructure. The tweets are filtered based on the mention of IT infrastructure assets and then normalized before they are fed to the DNN stage. For processing the text, a MTL DNN model forks into two output modules: a binary classifier and a Named Entity Recognizer (NER). Both share character-level and word-level representation layers which can either be a Convolutional Neural Network (CNN) [13] or a type of Recurrent Neural Network (RNN) such as the Long Short-Term Memory (LSTM) [14]. The combined result of the output modules produces a concise artifact reporting a security event, such as a vulnerability disclosure or security update, to issue an alert.

To train and evaluate our model, we used two sets of tweets, one previously collected for our baseline work [5] and another set collected over three months. We sought to analyse the different combinations of CNN and LSTM layers for the character-level and word-level representations for both our baselines and MTL models. The resulting MTL model provides an improvement on the binary classification task while being able to maintain the NER component performance on the testing set.

The remainder of the paper is organized as follows. Section II presents an overview of related work. Details on the baseline work to which we compare the proposed MTL approach, described in Section IV, are provided in Section III. The experimental set up is described in Section V and the results are shown and discussed in Section VI. Section VII presents a brief analysis and examples of IoCs that were successfully built using the proposed approach. Finally, Section VIII presents the conclusions taken from the experimental work.

II. Related Work

A. Multi-Task Learning

MTL is an approach where multiple tasks are learned in parallel using a shared domain knowledge representation [7]. This methodology is used so that learning from the training data of one task improves the learning of other tasks, thereby improving generalization. Collobert et al. [6] proposed a model with a weight-sharing capable of learning multiple tasks. The proposed model follows a common approach in MTL where earlier layers of the network are shared among tasks, leaving the final layers to be task specific. Another common component is the training schedule, where the network is trained iteratively by one task at the time. Liu et al. [8] proposed a multi-task methodology that leveraged a pre-trained language model [15] to obtain an improvement in ten Natural Language Understanding (NLU) tasks. Ruder et al. [16] presented a meta-architecture for sequence tagging problems. The architecture presented is capable of learning what layers to share between the networks, which parts of those layers to share and how much.

B. Machine Learning and Cyber Threat Intelligence in OSINT

There has been a considerable interest in previous works to use Twitter as an information source for CTI. Sabottke et al. [11] used Twitter to conduct a quantitative and qualitative exploration of vulnerability-related information and proposed a exploit detector using a Support Vector Machine (SVM) classifier. The detector is capable of extracting vulnerability-related information from Twitter, augment it with additional sources and predict if the vulnerability is exploitable in a real-world scenario. Subsequently, the authors also considered adversarial scenarios to their pipeline. Le Sceller et al. [17] proposed SONAR, an automatic keyword-centric self-learned framework that can detect, geolocate and categorize cybersecurity on a Twitter stream. Liao et al. [18] developed iACE for automatic IoC extraction from structured technical articles using context terms and grammatical relations. Alves et al. [12] presented the design of a classifier model, a SVM and a Multi-Layer Perceptron (MLP), for a Twitter-based threat monitor for generating a summary of the threat landscape related to a given monitored IT infrastructure. Zhong et al. [19] deployed a Logistic Regression to analyze the severity of cybersecurity threats based on the language that is used to describe them online.

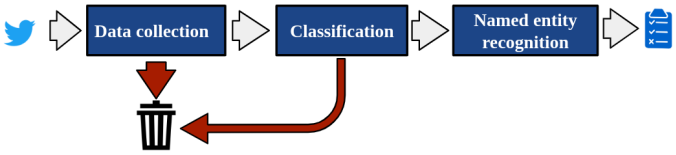

Fig. 1. Previous work Twitter-based threat detection pipeline architecture with single-task DNN [5].

III. Baseline DNN approach

In our previous work [5], we proposed a CTI tool that employed two sequential single-task DNN architectures to process a data stream, one identifying relevant security-related information and the other extracting relevant entities. The CTI pipeline architecture used in Alves work is shown in Figure 1

The first stage collects tweets, filters them based on a set of keywords and normalizes tweets to a specific format. In the second stage, a binary classifier labels tweets as either relevant, meaning they are likely to contain valuable information about an asset of interest, or irrelevant. Finally, in the information extraction stage, relevant tweets are processed by a NER network. The information extracted can be used to issue a security alert or to enrich an existing IoC in a threat intelligence platform such as MISP [20].

As the DNN architecture components used in these models form the basis of the MTL model proposed, we briefly describe them in the next subsections. A more formal specification will be given in section IV. The complete description of these DNNs design experiments can be found in the paper [5].

A. Classification DNN

The classification stage aims to detect tweets containing security-related information, so that only these proceed to the NER stage. We implemented a binary classifier using a CNN [13] whose architecture can be described by five layers: input, embedding, convolution, max-over-time-pooling, and output, as shown in Figure 2.

1) Input layer: The CNN receives sentences represented by a sequence of integers, each representing a word token, just as illustrated in Figure 2.

2) Embedding layer: Each integer corresponds to a numeric vector accounting for the semantic value of the corresponding word. These *word vectors* can be randomly initialized or extracted from trained language models (e.g., GloVE [21] or Word2Vec [22]). In both cases, the learning algorithm may further adjust the word vector.

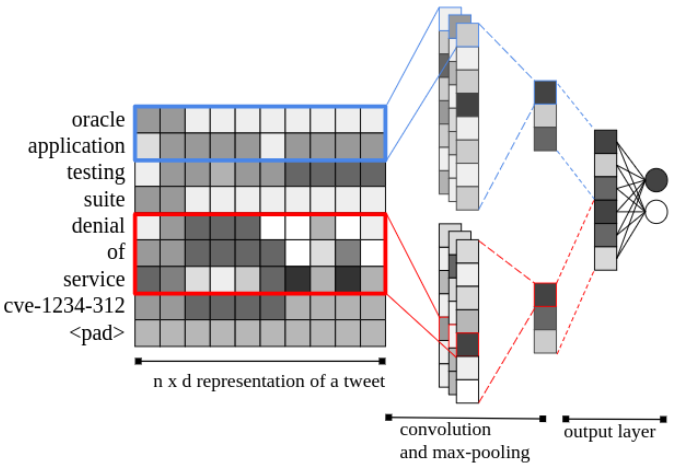

Fig. 2. Convolutional neural network architecture for sentence classification.

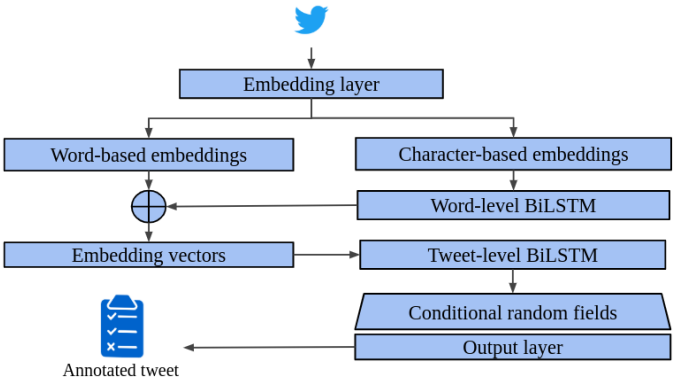

Fig. 3. Bidirectional long short-term memory architecture for named entity recognition.

3) Convolution layer: With a set of learnable kernels, the convolution operation will slide down these kernels producing feature maps. The height and width of the filters correspond to the number of words covered and to the dimension of the word vectors, respectively. Given that this operation computes one feature for each stride of the filter over the embedded matrix, the convolutional layer outputs a set of feature map vectors.

4) Max-over-time-pooling layer: The feature values composing a map denote how strong the feature is within a specific input window. We reduce the feature map into a single value providing information on the presence or absence of a feature, through the max-pooling operation [23].

5) Output layer: Before using the selected features in the output layer of the network, dropout [24] is applied to the feature vector. Dropout acts as a form of regularization [25], preventing overfitting and promoting generalization. Finally, feature nodes are used by a fully-connected softmax layer which outputs the probability of a tweet to contain relevant information. If relevant, the tweet moves to the next stage of the pipeline, otherwise it is dropped.

B. Named entity recognition DNN

In the NER phase, we aim to extract information from tweets that have been considered relevant by the classifier. Our model is based on a BiLSTM neural network [26], illustrated in Figure 3.

This network locates and labels valuable security-related entities such as monitored infrastructure assets, vulnerabilities, attacks, and vulnerability repository IDs mentioned in tweets. To the extent possible, we defined security-related entities according to descriptions from the ENISA risk management glossary [27]. Next, we briefly describe the network layers: input, embedding, word-level BiLSTM, tweet-level BiLSTM, and output.

1) Input layer: In addition to the integers representing the word tokens applied to the input layer of the classifier, this network receives secondary sequences of integers, one for each word at the input, representing the characters that form the word.

2) Embedding layer: This layer functionality is similar to the classifier's regarding the sequence of integers that correspond to the words in tweets. Additionally, the integers in each secondary sequence are also converted to numeric vectors, providing a character-level representation of the tweet. Therefore, besides the word-level embedded matrix, each word has a corresponding character-level matrix representation.

3) Word-level BiLSTM layer: The embedded character matrix corresponding to each word is fed to a BiLSTM network, containing two cells that read the sequence of character vectors in opposite directions. Both cells possess a hidden state vector that is updated at every time step (i.e., at every character read). After reading all the characters the cell states are extracted and concatenated, forming vectors that hold a character-level representation from both left-to-right and right-to-left readings.

4) Tweet-level BiLSTM layer: Similar to the process described for the word-level BiLSTM, we feed this tweet representation to another BiLSTM layer, word by word. However, while the previous layer only retrieves the final hidden state, in this layer we read it at every time-step (every word representation read).

5) Output layer: In the final layer of the NER model, we have a fully-connected neural network and a Conditional Random Field (CRF) module [28].

IV. Multi-task Learning Approach

The MTL CTI tool architecture is presented in Figure 4, which depicts a high-level representation of the three main stages. The first, collects tweets through the Twitter API, filters them based on a set of keywords, which are defined according to a specific IT infrastructure, and normalizes the tweets representation. In the second stage, a Multi-Task DNN model performs two tasks: (i) it classifies (relevant or not) tweets according to their relevance to the cybersecurity of the given infrastructure; and (ii) it processes relevant tweets to extract useful entities that can be used to issue a security alert or fill an IoC. In a final stage, these artifacts can feed a CTI

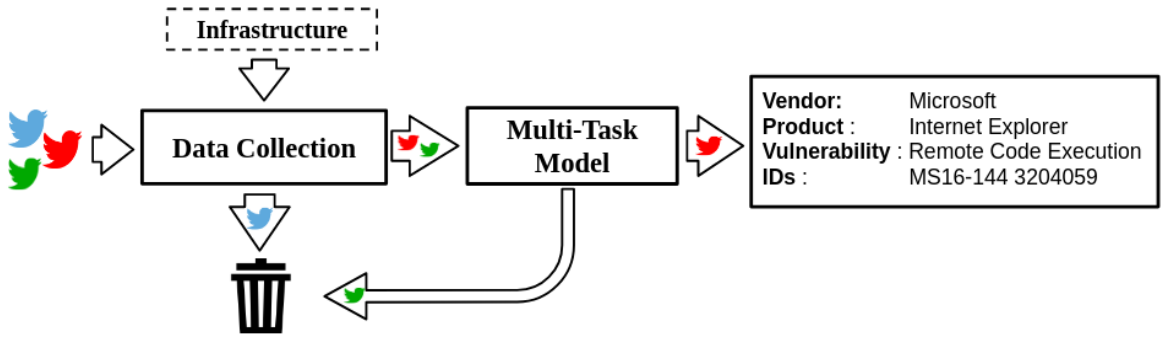

Fig. 4. MTL-based cyberthreat intelligence pipeline.

platform such as MISP [20]. In the following we give details about the two first stages.

A. Data collection

The first stage focuses on data processes, including collection, filtering and pre-processing.

A manually curated set of Twitter accounts, selected on the basis of their likelihood to share security-related information, are used to query the Twitter's streaming API.

By assuming that a tweet about an infrastructure asset has to mention its properties and components, the filter module employs a set of user-defined keywords describing the infrastructure being monitored to drop irrelevant tweets, further decreasing the amount of information that flows through the pipeline. For instance, an analyst wants to be informed about potential threats to a web service hosted on a cloud platform, the set of keywords used for system's filter should include operating systems, web server software (including version numbers), the cloud platform being used and all other components supporting the asset in question.

The final component in this stage is a simple pre-processing function that standardizes the representation of tweets by converting every character into lower-case, removing hyperlinks and special characters, except for useful characters such as '.', '-', '_', and ':', as they are often used in IDs, version numbers, or component names.

B. Multi-Task Learning Model

Figure 5 provides a high-level representation of our MTL architecture and its eight components, described below.

1) Input Layer: As briefly explained before, the model receives two representations of a text: a word-level sequence of n tokens where each token represents a word, and n_c character-level tokens, where each token represents a character. Each word token is converted to a randomly initialized d_w-dimensional vector, resulting in a $n \times d_w$ matrix. Similarly, each character token is also converted into a d_c-dimensional vector, leaving each word to be represented at the character-level by a $c \times d_c$ matrix where c is the number of characters in a word.

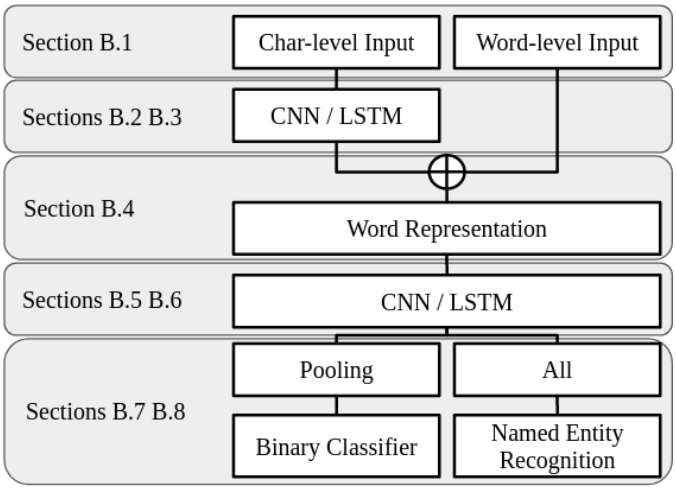

Fig. 5. Multi-Task Learning model architecture.

2) Character-level CNN Layer: Following the input layer, the CNN working at the character-level uses a set of k_c learnable kernels, each containing f_c filters with a height h_c, which are variable from kernel to kernel, and a width of d_c, equal to the d_c-dimensional vectors. During the convolutional operation these filters stride down the vectors producing $k_c \times f_c$ feature maps. These features are then passed by a max-pooling function, resulting in a single value from each feature map, leading to a total of $f_k c = k_c \times f_c$ features. The resulting character-level representation is a vector of $f_k c$ features for each word, outputting a numerical matrix of $n \times f_k c$.

3) Character-level LSTM Layer: Alternatively, the character-level representation can be produced by a bidirectional LSTM layer that uses two h_c-dimensional vectors, called hidden states, which are modified as the input is read. These vectors receive the input character-level vectors in opposite directions and once the input is processed, the resulting h_c-dimensional vectors are concatenated. Thus, this layer outputs a $h_c \times 2$ numerical vector for each word.

4) Word Representation: Once the character-level representation is ready, it is concatenated with the d_w-dimensional word-level representation. This leads to each word being represented by a vector of length $d_w + k_c \times f_c$ when the

character-level CNN is used, or of length $d_w + 2 \times h_c$ when using the character-level LSTM. We denote this new vector length as d.

5) Word-level CNN Layer: The word-level CNN layer is similar to the character-level CNN layer, except that it receives a composed word representation from the previous layer. It uses k_w kernels, each with f_w filters of height h_w and width d (dimension of the word representation vector). The process of computing the feature maps is equivalent to the character-level CNN layer. However, in this case we pad the input before the convolution to keep the original dimensions. As so, we do not perform any pooling operation at this stage and thus output a $n \times k_w \times f_w$ matrix.

6) Word-level LSTM Layer: The word-level LSTM layer requires less modifications to its character-level counterpart. It receives the $n \times d$ input, initializes its h_w-dimensional vectors for both directions, and processes the input at each time-step. The only modification necessary to the workflow described before consists in keeping the state of the hidden vectors at each time-step as opposed to keeping only the last vectors. This leads to an output of a $n \times h_w \times 2$ matrix.

7) Binary Classification: The output layer of the first task produces a binary classification on the basis of the shared layers described above. Since we intend to classify the sentence as a whole, we reduce the dimension of the received input by performing a max-pooling operation. The resulting features are flattened into a final feature vector and passed to a fully-connected sigmoid layer.

8) Named Entity Recognition: For our second task, instead of classifying the sentence globally, we intend to label each of the words as one of t possible entities. The output from the shared layers is passed through a fully-connected neural network containing t neurons, providing a $n \times t$ matrix. This acts as a score matrix, where the activation of the t neurons produces a score for each label. These could be passed through a *softmax* function, leaving the prediction to be made independently, or a CRF module which is beneficial in cases of dependency between the target values. Based on previous experimental works [29], we chose to use a CRF layer to produce the NER stage predictions.

V. Experimental Set-up

This section describes the experimental work designed to establish a baseline among the single-task models and subsequently the evaluation of the multi-task model on both tasks.

A. Datasets

For model training and evaluation, we collected a total of 31281 tweets across two time periods, **T1** and **T2**, as displayed in Table I. These tweets were collected through a selection of manually curated accounts and passed through a filter, as previously described in Section IV-A. Once the data was gathered, we manually labeled the tweets as relevant or not relevant, depending on the security-related content found. Afterwards, the subset of tweets labeled as relevant were selected for the NER task. The words in these tweets where then labeled using the labels shown in Table II. The ENISA risk management glossary [27] was used to define the entities, although we adopted broad definitions so that the labels become more distinguishable by the network. For example, the label `VUL` includes both vulnerabilities and threats.

TABLE I
Datasets used for training and evaluation.

Dataset	Time Interval	Positives	Negatives	Entities
T1	21/11/2016 to 27/03/2017	8093	12954	42793
T2	01/06/2018 to 01/09/2018	2980	7254	13694

TABLE II
Named entities to be extracted from a tweet.

Label	Description
O	Does not contain useful information.
ORG	Company or organization.
PRO	A product or asset.
VER	A version number, possibly from the identified asset or product.
VUL	May be referencing the existence of a threat or a vulnerability.
ID	An identifier, either from a public repository such as the National Vulnerability Database (NVD) [30], or from an update or patch.

B. Training and Evaluation

For the training stage we used dataset **T1** and the first month of **T2**. The remaining samples of **T2** were used for testing. During training, we randomly sampled 20% of the data to form a validation set for early stopping purposes. For single-task models, the early stopping criteria focused on the F1-score not improving for a predetermined amount of steps. Regarding multi-task models, we used the F1-score of the two tasks F1-scores, as we weight both tasks equally.

One of the objectives of our evaluation was to analyze the impact that each module has in the model performance. Thus, for both tasks, we evaluated each combination of the character-level and word-level modules, including the absence of one of these representations. This resulted in eight variations as displayed in Table III.

The models were implemented in PyTorch [31] using the Adam optimizer [32], and trained with batches of 256 data points, using a learning rate of 0.001 for both the binary and the NER training steps.

A grid-search was conducted to tune hyper-parameters and model design variables. The following alternatives were considered in the search:

- Character vector dimension: 50, 100, 200.
- Word vector dimension: 100, 200, 300.
- Character-level CNN used a single kernel with height 2, 3, or 4, and number of filters within {64, 128, 256}.
- Word-level CNN used 1 to 3 kernels with height 2, 3, or 4, and number of filters within {64, 128, 256}.

TABLE III
MODEL ARCHITECTURE VARIATIONS.

Model	Description
WordCNN + CharCNN	Uses a CNN for the word-level representation and character-level representation
WordCNN + CharRNN	Uses a CNN for the word-level representation and RNN for the character-level representation
WordRNN + CharCNN	Uses a RNN for the word-level representation and CNN for the character-level representation
WordRNN + CharRNN	Uses a RNN for the word-level representation and character-level representation
Only WordCNN	Uses only a CNN for the word-level representation
Only WordRNN	Uses only a RNN for the word-level representation
Only CharCNN	Uses only a CNN for the character-level representation
Only CharRNN	Uses only a RNN for the character-level representation

- Charecter-level RNN used a single bidirectional LSTM cell with hidden vector dimensions varying within {100, 200, 300}.
- Word-level RNN used a single bidirectional LSTM cell with hidden vector dimensions varying within {100, 200, 300}.
- Dropout probability before the output layer: 0.0, 0.3, 0.5.

For the Multi-task cases we opted to only evaluate the architectural variations that used both types of input representations (word and character). The multi-task training schedule is presented in Algorithm 1.

Further details on the parameters and design of the models can be seen in the JSON files available with the source code[1].

Algorithm 1: Multi-Task Learning schedule.

```
Initialize model parameters θ randomly.
Set the stopping criteria : stopTrain
Set mini-batches for each task
while not stopTrain do
    for t in Tasks do
        Get batch for task t: b_t
        Compute loss for task t: L_t(θ, b_t)
        Compute gradient : ∇_t(θ)
        Update model weights : θ = θ − ε∇_t(θ)
    end
end
```

VI. RESULTS

For evaluation of the binary classification task, we chose to measure the F1-score between the True Positive Rate (TPR) and the True Negative Rate (TNR). Similarly, for the NER task we calculated the F1-score between Precision and Recall. Each result presented corresponds to the best case found in the grid-search conducted for each modular architectural variation.

A. Single-Task Models

Tables IV and V show the single-task models results obtained with the validation and testing sets (separated by a /), for the binary classification and NER tasks, respectively.

[1]https://github.com/ndionysus/multitask-cyberthreat-detection

TABLE IV
SINGLE TASK MODEL BINARY CLASSIFICATION RESULTS (VALIDATION / TEST).

	CharCNN	CharRNN	No Char
WordCNN	**0.947 / 0.916**	0.944 / 0.901	0.909 / 0.883
WordRNN	**0.946 / 0.917**	0.939 / 0.902	0.902 / 0.887
No Word	0.932 / 0.900	0.938 / 0.903	

TABLE V
SINGLE TASK MODEL NAMED ENTITY RECOGNITION RESULTS (VALIDATION / TEST).

	CharCNN	CharRNN	None
WordCNN	0.971 / 0.939	**0.979 / 0.940**	0.915 / 0.883
WordRNN	**0.979 / 0.940**	0.977 / 0.937	0.880 / 0.867
None	0.971 / 0.938	0.959 / 0.931	

TABLE VI
MULTI-TASK MODEL RESULTS (VALIDATION / TEST).

	Binary	**NER**
WordCNN + CharCNN	0.945 / 0.915	0.967 / 0.936
WordRNN + CharCNN	0.951 / 0.918	0.973 / 0.938
WordCNN + CharRNN	0.949 / 0.921	0.968 / 0.932
WordRNN + CharRNN	**0.951 / 0.922**	**0.973 / 0.940**

In the binary classification model, the combination of character-level CNN and word-level RNN or CNN, presented the best results (comparable for word-level RNN or CNN). Regarding the NER model, two different variations achieved the best results, indicating that the best solution is to use a combination of CNN and RNN, regardless of the type of each specific layer. The analysis of the results presented in both tables also leads to the conclusion that the usage of both representation levels, character and word, is beneficial for the model. This is likely due to the fact that by additionally using the character representation, the model becomes less dependent on the training set vocabulary.

B. Multi-Task Models

Table VI shows the results obtained by the four Multi-Task DNN models in both tasks. As previously observed, using

TABLE VII
SAMPLE OF THREE INDICATORS OF COMPROMISE PRODUCED BY THE TEST SET.

Tweet	NVD date	Tweet date	CVSS
Microsoft Windows - Advanced Local Procedure Call (ALPC) Local Privilege Escalation Exp (...)	09/12/2018	28/08/2018	7.9
Adobe Flash - AVC Processing Out-of-Bounds Read Exploit CVE-2018-12827 (...)	29/08/2018	28/08/2018	7.5
Vuln: Adobe Flash Player CVE-2018-12828 Unspecified Privilege Escalation Vulnerability (...)	29/08/2018	16/08/2018	9.8

both character-level and word-level representations provides better results when compared to the absence of any of these. Therefore, we focused on obtaining results only for these four variations of the model architecture. The configuration of the best models obtained from the grid-search is the following:

- WordCNN + CharCNN: character embedding dimensions are set to 50. The CharCNN component uses 1 kernel with height of 3 and 128 filters. The resulting features are concatenated with a word embedding with dimension of 300. These features are then sent to the WordCNN component with 3 kernels, with heights of 2, 3, and 4, all having 128 filters.
- WordRNN + CharCNN: character embedding dimensions are set to 100. The CharCNN component uses 1 kernel with height of 3 and 128 filters. The resulting features are concatenated with a word embedding vector with dimension of 200. These features are then sent to the WordRNN component with 1 bidirectional LSTM cell with hidden vectors dimension of 300.
- WordCNN + CharRNN: character embedding dimensions are set to 100. The CharRNN component uses 1 bidirectional LSTM cell with hidden vectors dimension of 200. The resulting features are concatenated with a word embedding with a dimension of 200. These features are then sent to the WordCNN component using 1 kernel with height of 2 and 128 filters.
- WordRNN + CharRNN: character embedding dimensions are set to 100. The CharRNN component uses 1 bidirectional LSTM cell with hidden vectors dimension of 100. The resulting features are concatenated with a word embedding with a dimension of 200. These features are then sent to the WordRNN component with 1 bidirectional LSTM cell with hidden vectors dimension of 300.

For the binary task, a slight improvement of 0.5% was observed on both validation and test results when compared to the best single-task results. The MTL model which obtained the best results, uses a combination of character and word level layers that previously displayed worse results when compared to the other three single task variations.

The NER task results did not achieve an equivalent improvement. Nonetheless, the best model variant obtained comparable performance to the best single-task model variations. We argue that the reason why only the binary task achieved a slight improvement is twofold. First, as the problem consists of predicting a label for each separate word, the NER task is more detailed and specific. This leads the model to exploit local relationships and build representations that favor this specificity, which can also be helpful to predict if a tweet is mentioning security-related information or not. However, the opposite may not be true. The binary task is more general, leading the model to learn representations of the overall semantic value of a whole sentence. As the NER task focuses more on local relations than global ones, this global information is likely to be less useful. Second, given that neural networks are data-hungry models, more importantly than the network architecture or the training process, the current size of the dataset may be a limitation. For the NER task, its size is only a fraction of the whole dataset used for the binary task, since only the positive entries are used to train.

VII. EXAMPLES OF INDICATORS OF COMPROMISE

To demonstrate the usefulness of Twitter as a source for CTI, we simulated a real-world scenario by passing the test set tweets through the deployed pipeline (Figure 4). Table VI-B presents a sample of tweets found in the testing set that were available on Twitter prior to their official disclosure in the NVD [30]. Each row displays a tweet, the NVD disclosure date, the earliest reference found in the testing set to the vulnerability, and the severity score according to the Common Vulnerability Scoring System (CVSS) [33]. The first tweet displayed in the Table VI-B refers to a zero-day vulnerability in Microsoft Windows' task scheduler. This was a case where a Twitter user published a vulnerability, paired with a proof-of-concept exploit, before the official disclosure by the product owner. The other two cases mention Adobe products vulnerabilities with high or critical severity. These three examples demonstrate the timeliness and relevance of the information extracted by the pipeline.

VIII. CONCLUSIONS

In this paper we report the merging of two separate deep neural network architectures developed in previous work, used for two distinct tasks of a cyberthreat intelligence tool. The resulting (upgraded) tool receives a stream of tweets which are processed by a single multi-task deep neural network model. We have shown that the multi-task model could achieve similar results regarding named entity recognition and binary classification tasks. Consequently, the complexity of the information pipeline of the cyberthreat intelligence tool can be greatly reduced, and the procedures required for online update of the dataset and model parameters are also simplified. In future work, the pipeline can be further enhanced by the usage of recent pre-trained language models that can be further fine-tuned with cybersecurity domain-specific text.

ACKNOWLEGMENT

This work was supported by EC funding through the H2020 DiSIEM project (H2020-700692) and by FCT through the LASIGE Research Unit, ref. UIDB/00408/2020 and ref. UIDP/00408/2020.

REFERENCES

[1] "2019 Cyberthreat Defense Report," https://www.imperva.com/resources/resource-library/reports/2019-cyberthreat-defense-report/, accessed: 2019-12-10.

[2] Q. Le Sceller, E. B. Karbab, M. Debbabi, and F. Iqbal, "SONAR: Automatic Detection of Cyber Security Events over the Twitter Stream," in *Proc. of the 12th International Conference on Availability, Reliability and Security (ARES)*. Association for Computing Machinery, 2017.

[3] X. Liao, K. Yuan, X. Wang, Z. Li, L. Xing, and R. Beyah, "Acing the IOC Game: Toward Automatic Discovery and Analysis of Open-Source Cyber Threat Intelligence," in *Proc. of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Association for Computing Machinery, 2016.

[4] C. Sabottke, O. Suciu, and T. Dumitras, "Vulnerability Disclosure in the Age of Social Media: Exploiting Twitter for Predicting Real-World Exploits," in *Proc. of the 24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, 2015.

[5] N. Dionísio, F. Alves, P. M. Ferreira, and A. Bessani, "Cyberthreat detection from twitter using deep neural networks," in *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019.

[6] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th International Conference on Machine Learning*. Association for Computing Machinery, 2008.

[7] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[8] X. Liu, P. He, W. Chen, and J. Gao, "Multi-task deep neural networks for natural language understanding," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[9] J. Baxter, "A bayesian/information theoretic model of learning to learn via multiple task sampling," in *Machine Learning*, 1997, pp. 7–39.

[10] A. Attarwala, S. Dimitrov, and A. Obeidi, "How efficient is Twitter: Predicting 2012 U.S. presidential elections using Support Vector Machine via Twitter and comparing against Iowa Electronic Markets," in *Intelligent Systems Conference*, 2017.

[11] C. Sabottke, O. Suciu, and T. Dumitras, "Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015.

[12] F. Alves, P. M. Ferreira, and A. Bessani, "Design of a classification model for a twitter-based streaming threat monitor," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2019.

[13] Y. Kim, "Convolutional Neural Networks for Sentence Classification," *arXiv e-prints*, 2014.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, 1997.

[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019.

[16] S. Ruder, J. Bingel, I. Augenstein, and A. Søgaard, "Latent Multi-task Architecture Learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

[17] Q. Le Sceller, E. B. Karbab, M. Debbabi, and F. Iqbal, "Sonar: Automatic detection of cyber security events over the twitter stream," in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, ser. ARES '17, 2017.

[18] X. Liao, K. Yuan, X. Wang, Z. Li, L. Xing, and R. Beyah, "Acing the ioc game: Toward automatic discovery and analysis of open-source cyber threat intelligence," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[19] S. Zong, A. Ritter, G. Mueller, and E. Wright, "Analyzing the Perceived Severity of Cybersecurity Threats Reported on Social Media," *arXiv e-prints*, 2019.

[20] C. Wagner, A. Dulaunoy, G. Wagener, and A. Iklody, "MISP: The Design and Implementation of a Collaborative Threat Intelligence Sharing Platform," in *Proc. of the 2016 ACM on Workshop on Information Sharing and Collaborative Security (WISCS)*. Association for Computing Machinery, 2016.

[21] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation," in *Proc. of the Empirical Methods in Natural Language Processing*, 2014.

[22] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[23] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural Language Processing (Almost) from Scratch," *Journal of Machine Learning Research*, 2011.

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, 2014.

[25] S. Wager, S. Wang, and P. S. Liang, "Dropout Training as Adaptive Regularization," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013.

[26] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, "Neural architectures for named entity recognition," in *Proc. of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2016.

[27] ENISA, "Risk Management - Glossary," https://www.enisa.europa.eu/topics/threat-risk-management/risk-management/current-risk/risk-management-inventory/glossary, accessed: Sept. 2018.

[28] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data," in *Proc. of the 18th International Conference on Machine Learning (ICML)*. Morgan Kaufmann Publishers Inc., 2001.

[29] J. Yang, S. Liang, and Y. Zhang, "Design challenges and misconceptions in neural sequence labeling," in *Proceedings of the 27th International Conference on Computational Linguistics*, Aug. 2018.

[30] Information Technology Laboratory, "National Vulnerability Database," https://nvd.nist.gov/, accessed: Jan. 2020.

[31] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *arXiv e-prints*, 2019.

[32] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv e-prints*, 2014.

[33] Forum of Incident Response and Security Teams, "Common Vulnerability Scoring System," https://www.first.org/cvss/, accessed: Jan. 2019.