VASE: A Twitter-based Vulnerability Analysis and Score Engine

Haipeng Chen[1,*], Jing Liu[2,**], Rui Liu[1,*], Noseong Park[2,**], and V.S. Subrahmanian[1,*]

[1]Dartmouth College
[2]George Mason University
[*]{haipeng.chen, rui.liu.gr, vs}@dartmouth.edu;
[**]{jliu30, npark9}@gmu.edu

***Abstract*—When a new vulnerability is discovered, a Common Vulnerability and Exposure (CVE) number is publicly assigned to it. The vulnerability is then analyzed by the US National Institute of Standards and Technology (NIST) whose Common Vulnerability Scoring System (CVSS) evaluates a severity score that ranges from 0 to 10 for the vulnerability[1]. On average, NIST takes 132.7 days for this — but early knowledge of the CVSS score is critical for enterprise security managers to take defensive actions (e.g. patch prioritization). We present VASE (Vulnerability Analysis and Scoring Engine) that uses Twitter discussions about CVEs to predict CVSS scores before the official assessments from NIST. In order to leverage the intrinsic correlations between different vulnerabilities, VASE adopts a graph convolutional network (GCN) model in which nodes correspond to CVEs. In addition, we propose a novel attention-based input embedding method to extract useful latent features for each CVE node. We show on real-world data that VASE obtains a mean absolute error (MAE) of 1.255 for predicting the CVSS score using only *three days* of Twitter discussion data after the date a vulnerability is first mentioned on Twitter. VASE can provide predictions for the CVSS scores for 37.85% of the CVEs at least one week earlier than the official assessments by NIST.**

***Index Terms*—Vulnerability Severity Prediction; Social Media Data Mining; Graph Convolution Networks; Input Embedding**

1 Introduction

In the face of an ever-increasing barrage of attacks, alerts, malware, patches, and more, IT security personnel in most organizations are overworked and overwhelmed [1]. To help ameliorate this situation, when a new vulnerability is discovered, a Common Vulnerability and Exposure (CVE) numbering authority[2] assigns a CVE number to that vulnerability, along with a brief description. Subsequently, NIST releases a severity score via the Common Vulnerability Scoring System (CVSS) for that CVE. The CVSS score of a CVE serves as a key input for critical decisions (e.g., severity score guided patching [2, 3]) by IT security personnel. However, as pointed out in [4], it takes an average of 132.7 days for NIST to evaluate the CVSS score after the public assignment of a CVE number to a vulnerability. During this period, enterprises may be vulnerable to malicious cyberattacks which utilize the information provided at the time of disclosure. According to [5], "half of the exploits were published within two weeks after the vulnerability being discovered", and "only one percent of vulnerabilities have exploits developed more that a year after its discovery". Therefore, it is critical to provide an estimation of the CVSS score as early as possible.

Social media has been used to predict certain cybersecurity events [6, 7, 8, 9, 4, 10]. Our data indicates that there is active discussion on Twitter after some CVE numbers are published. In this paper, we leverage the Twitter discussion about a CVE in order to predict that CVE's CVSS score as early as possible.

A straightforward way is to train a feature engineering-based machine learning model (with historical data) to independently predict the CVSS score for each CVE. However, this is sub-optimal because i) it neglects some potential correlations between different CVEs and ii) it requires manual effort to filter out useless tweets and extract important features. For instance, Twitter discussion analysis shows that CVE-2017-0143 and CVE-2017-0146 are highly correlated vulnerabilities — both were used by the infamous WannaCry ransomware. To better leverage correlations among CVEs, we make our first technical contribution by building a graph convolutional network (GCN) based prediction model, where each node of the graph is a CVE, and the edges are constructed using semantic similarities between the sets of tweets discussing each pair of CVEs. In general, each tweet contains only partial information of a CVE, and is not likely to cover all aspects about a CVE. Hence, it is important to consider the whole set of tweets that mention a given CVE.

Features are usually fed directly into GCNs after preprocessing — but this can involve expensive feature extraction. Some GCN models adopt the concept of a (usually linear) embedding layer before graph convolution layers in order to reduce dimensionality. However, in our problem, the input to the GCN is a pool of tweets which cannot be efficiently encoded by a simple linear embedding layer. The number and content of tweets that mention different CVEs can vary dramatically, making it difficult to uniformly represent them as a single-sized input feature vector. For some popular CVEs, the number of associated tweets is much larger (e.g., several thousands) than for others. Hence it is challenging to sort out the tweets with the most critical information within a large pool of related tweets. Moreover, not all tweet sentences can be

Authors listed in alphabetic order.
[1]We use vulnerability and CVE interchangeably for the same concept.
[2]MITRE Corporation is one such CVE numbering authority in the US.

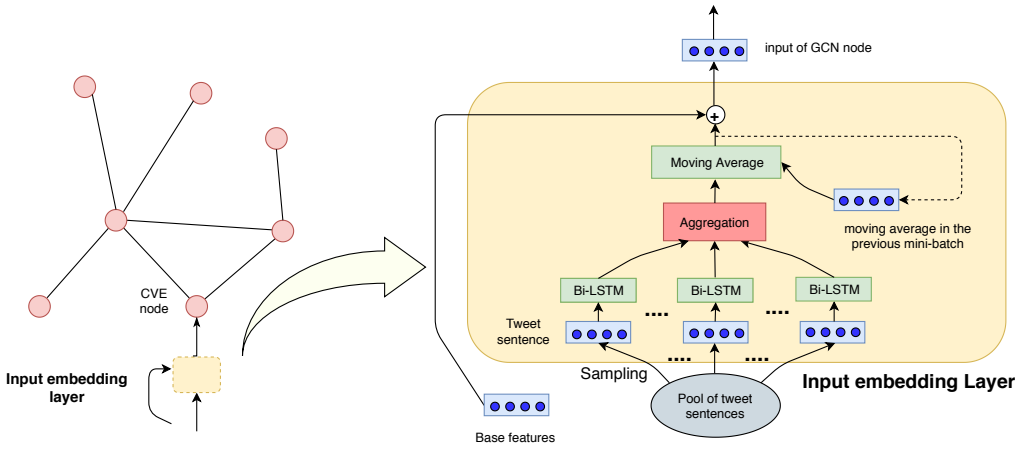

Fig. 1: The proposed GCN-AE architecture with a sample embedding layer for one CVE.

fed into one training mini-batch due to limited GPU memory.

To tackle these two challenges, our second technical contribution is the design of a novel *embedding layer* (cf. Figure 1) on the raw input (i.e., a pool of tweet content), which consists of three sub-layers: i) a bi-directional LSTM which is used to extract a latent feature vector for each tweet, ii) an attention layer which is to efficiently sort useful tweets' feature vectors out, and iii) a moving average layer which is used to handle the GPU memory limitation. Our proposed Graph Convolutional Network with Attention-based input Embedding (GCN-AE) provides an end-to-end framework that automatically extracts useful features from the pool of tweets of a CVE.

We conduct extensive experiments to evaluate VASE using 8-months of CVE data from 2017. The results show that using only 3 days of Twitter discussion data related to a CVE (after the date that the CVE is first mentioned on Twitter), our approach predicts CVSS scores with a mean absolute error (MAE) of 1.255. VASE can provide predictions for 37.85% of the CVEs at least one week earlier than the official vulnerability assessments by NIST. As a comparison, note that only 42.19% of CVEs are ever discussed on Twitter after the assignment of a CVE number. The effectiveness of different components of GCN-AE is shown via an ablation study.

2 Related Work & GCN Preliminaries

Social Media Data Mining for Cybersecurity [6, 11, 4, 10] use Twitter to predict if and when a CVE will be exploited in the future. [7, 8, 9] use Stack Exchange, Reddit and Twitter to find potential cyberattack events. However, none of these works try to predict the severity score of a CVE. The one work most closely related to ours is [12] which uses Twitter data to estimate their own notion of severity (different from the CVSS severity) for a vulnerability. However, they first predict the severity of a CVE independently from *each individual tweet*, and then choose the largest severity value among all the estimated values from a small pool of tweets that mention the CVE, which is much simpler than our GCN-based processing. They use a small corpus of $6,000$ tweets for this purpose. In contrast with their relatively simple model, we develop a neural network-based model which predicts the widely-accepted CVSS score from a pool of tweets, and the size of our dataset (with 204,423 tweets) is more than 30 times larger than theirs.

Graph Convolutional Networks GCNs have been widely adopted to deal with graphical data. Traditionally, GCNs are designed on top of the graph spectrum (or Laplacian matrix) [13, 14, 15]. Given a graph G with n nodes and an adjacency matrix A, a layer-wise forward propagation rule is usually defined as:

$$X^{l+1} = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{\frac{1}{2}}X^{l}W^{l}), \tag{1}$$

where X^l and X^{l+1} are respectively the input and output matrices of the l^{th} GCN layer. $\hat{A} = A + I$ aggregates the feature vectors of the node and its adjacent nodes. Here, I is the identity matrix whose dimension is the same as A. $\hat{D}$ is the diagonal node degree matrix which is used to normalize $\hat{A}$. W^l is a trainable weight matrix of layer l. σ represents an activation function.

One limitation of the above graph convolution is the high computational complexity of calculating $\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{\frac{1}{2}}$, especially with a large number of nodes in the graph (which is the case in VASE). Various models have been proposed to improve the run-time of GCNs [16, 17, 18, 19, 20, 21], which can be generally formulated in the following form:

$$X^0 = e(F) \tag{2}$$

$$Z^{l+1} = g(X^l, A) \tag{3}$$

$$X^{l+1} = c(Z^{l+1}, X^l) \tag{4}$$

Here $e(\cdot)$ is an *embedding* operator which transforms the raw input feature matrix $F \in \mathbb{R}^{n \times m'}$ into the first hidden matrix $X^0 \in \mathbb{R}^{n \times m}$, where m' and m are respectively the dimensions of the raw and embedded input vectors, and n is the number of nodes. $g(\cdot)$ is an *aggregation* operator, which selectively aggregates each node's neighbors' hidden vectors from the l^{th} GCN layer based on the adjacency matrix A. $c(\cdot)$ is a *combination* operator which combines Z^{l+1} and X^l. Note that for all the hidden matrices $X^l, l = 0, 1, \ldots$, their row

dimension is always equal to n, while their column dimension depends on the combination operator $c(\cdot)$.

Existing models differ in the definition of the three operators. For the aggregation operator, LGCN [18] uses k-max pooling, where k is the average degree of nodes, while GraphSAGE [16] uses max/mean pooling or LSTM. For the combination operator, LGCN and GraphSAGE use column-wise 1D convolutional and fully connected layers respectively. PinSAGE [19] is a modified version of GraphSAGE to handle web-scale graphs. Aggregation and combination are integrated with an attention mechanism in GAT [17] and AGNN [21].

While aggregation and combination have been well studied, the embedding operator has received less attention: in particular, the embedding operator is usually a fully connected (linear) layer for dimensionality reduction. As pointed out earlier, a key challenge in applying existing approaches to our problem is that the raw inputs are pools of tweet sentences which cannot be adequately encoded using simple linear embedding layers. We therefore propose a novel embedding operator by integrating a multi-layered bi-directional LSTM and an attention mechanism to process raw tweets.

3 Problem Description

The CVSS [22], which is maintained by the CVSS Special Interest Group (SIG), provides a globally recognized standard for CVEs. A CVSS score is a numerical score between 0 and 10 that reflects the severity of a CVE.

The goal of this paper is to predict the CVSS score of a given CVE by using 3 days of tweets about the CVE after its first mention on Twitter. This paper uses a dataset containing approximately 8 months of tweets from January to August 2017. It contains the following 3 components:

CVE: The CVE [23] dataset contains a list of vulnerabilities that are discussed on the Twitter platform *before* their CVSS scores are released by NIST. Each CVE entry contains a CVE number and the date on which it is released by MITRE. In total, there are 6,270 CVEs in our dataset, which takes up around 42.69% of all the CVEs within the targeted time period.

NVD: We obtain ground truth about CVSS scores from the National Vulnerability Database (NVD) [24] at NIST which publishes and maintains CVSS scores. We only collect CVE entries which are discussed on Twitter before NIST releases the CVSS score, so the size of the NVD dataset is the same as that of the CVE dataset.

Twitter: We extract all tweets that have the "CVE" in them during the targeted time frame from the *Twitter firehose*. From our Twitter data, we observe that 58% of the tweets about a given CVE are posted within 3 days after its first-mention date (i.e., the date it is first mentioned on Twitter), and the marginal increase of tweet volume becomes quickly smaller after 3 days. To obtain a prediction as early as possible while obtaining enough Twitter discussion data about the CVEs, we use the 3-day Twitter discussion data for each CVE as the raw input to GCN-AE. Our Twitter dataset contains a total of 204,423 tweets and 28,893 users.

4 Methodology

VASE consists of three main components: i) graph construction based on basic natural language processing methods, ii) attention-based input embedding, and iii) transductive inference with GCN-AE.

4.1 Basic Features

We first extract two types of base features for each CVE:

1) The numbers of: tweets mentioning the CVE, retweets, replies, tweets favorited, hashtags/URLs/user mentions per tweet, verified accounts; the average age of accounts; and the average number of tweets per account. Note that these features are proposed by [6] for vulnerability exploit prediction.

2) We extract a bag of keywords (BoW) for each CVE using the mutual information metric [25], and thus obtain a BoW feature matrix for the entire set of CVEs. We then apply Principal Component Analysis (PCA) on the BoW feature matrix to reduce dimensionality. We call it "BoW features".

4.2 CVE Graph Construction

Traditional machine learning models which make predictions independently may be sub-optimal since they neglect the correlations between similar CVEs. They also require manual effort to filter out useless tweets. We therefore build a GCN-based prediction model, which considers correlations between CVEs. These are encoded as a CVE graph where each CVE is represented as one node. As in [14], we have an edge linking two CVEs if their BoW feature vectors have a cosine similarity of κ or more and treat κ as a hyper-parameter.

4.3 Attention-based Tweet Content Embedding

BoW features are coarse-grained and do not fully capture the content of the entire pool of tweets for the CVEs. We therefore use the entire pool of tweets as the raw input feature matrix denoted as $F \in \mathbb{R}^{t \times m'}$ where t is the total number of input tweets, and m' is the maximum sentence length. As mentioned earlier, most existing GCNs use one linear layer for the initial embedding $e(\cdot)$: $X^0 = FW^0$, where W^0 is a trainable weight matrix of the linear layer. This linear layer is essentially performing a linear transformation of the original input feature matrix F^0 into X^0. However, a simple linear layer cannot directly extract proper latent features from raw texts in our problem. We therefore propose a novel architecture to learn an embedding of the pool of tweets.

Figure 1 shows the overall VASE architecture. For each CVE node, there is one embedding layer to extract a hidden vector from the pool of tweets (sentences). The detailed description *for each CVE node* is as follows. Note that for each CVE one tweet sentence is denoted as a sequence of word embedding vectors.

1) First, we use 2-layer bi-directional LSTM networks to process raw input tweets and produce a hidden matrix $H^{lstm} \in \mathbb{R}^{t \times d}$ for the set of all tweets given a CVE, where t is the number of input tweets for the CVE and d is the dimension of the LSTM hidden vector. Note that t is smaller than the size of the entire pool in many cases. For

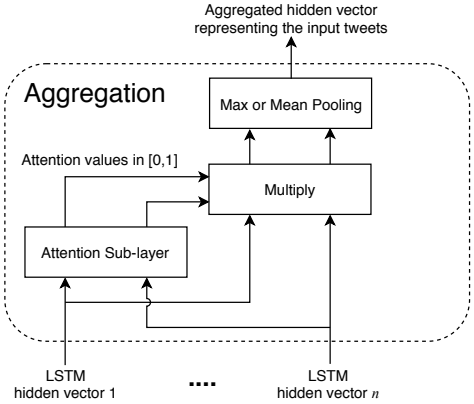

Fig. 2: Attention-based input aggregation layer. This attention mechanism is trained to extract sentences with useful information.

space reasons, we omit the description of bi-directional LSTM networks – see [26].

2) We then apply an attention layer to get an attentive matrix $H^{attn} \in \mathbb{R}^{t \times d}$. Intuitively, if a tweet is not useful in prediction, its corresponding vector in H^{attn} will be a zero vector. We then do a max or mean pooling over H^{attn} to get an aggregated hidden vector for each CVE.
3) Finally, the CVE-level hidden vector is concatenated with the basic feature vector H^{base} to get the final input feature vector for the GCN.

Figure 2 illustrates the attention-based input aggregation layer. Note that this input aggregation in VASE's embedding layer is different from the aggregation function of the GCN in Eq. (4).

4.3.1 Attention based Aggregation with Slope Annealing: Formally, for each CVE, we use the following attention mechanism to extract features from the important tweets:

$$\hat{h} = \sigma(\tau H^{lstm} W^{attn}), \tag{5}$$

$$H^{attn} = \hat{h} \cdot H^{lstm}, \tag{6}$$

where $\hat{h} \in \mathbb{R}^t$ is the attention value column vector. Each element in $\hat{h}$ represents a scalar attention value for each tweet sentence. Intuitively, higher attention values are assigned to tweet sentences which are more useful for prediction. $W^{attn} \in \mathbb{R}^{d \times 1}$ is a trainable weight matrix, and σ is the Sigmoid activation function. In order to amplify the differences between attention values for different tweets, we apply slope annealing [27] by multiplying $H^{lstm}W^{attn}$ with a scalar slope parameter $\tau > 1$. We start with $\tau = 1$ (i.e., the original Sigmoid function) and increase it by 0.004 in every epoch until a maximum value $\tau_{max} = 5$ is reached. Thus, the slope is τ times more steep than that of the original Sigmoid function and the attention values in $\hat{h}$ converge to either 0 or 1 quickly. Here ' $\cdot$ ' denotes a row-wise product of $\hat{h}$ and H^{lstm}. The row-wise product yields the attentive matrix $H^{attn} \in \mathbb{R}^{t \times d}$. Intuitively, if a tweet is not useful in improving prediction, the attention weight matrix W^{attn} will be trained to assign low values to its corresponding row-vector in H^{attn}.[3]

[3]We omit the subscript for CVE nodes for ease of representation.

We then use mean or max pooling over the attention matrix H^{attn} to get a single vector representation $h^{pool} \in \mathbb{R}^d$: h^{pool} is the vector representation of the input t tweets.

4.3.2 Moving Average: Ideally, all tweets in the pool should be fed into H^{lstm}. However, this poses 2 problems. First, the numbers of tweets in the tweet pools vary from CVE to CVE. Moreover, when the number of tweets is large, GPU memory becomes an issue. To handle these two issues, we sample a fixed number of t tweet sentences from the entire pool of tweets for each mini-batch, while maintaining a *moving average* of the aggregated hidden vector h^{pool} over mini-batches (the dotted line in Figure 1) :

$$\hat{h}_i^{pool} = \gamma \hat{h}_{i-1}^{pool} + (1-\gamma) h_i^{pool}, \tag{7}$$

where h_i^{pool} is the aggregated hidden vector at i-th mini-batch and $\gamma \in [0, 1]$ is a parameter indicating the update rate of the moving average. Hence, $\hat{h}_i^{pool}$ captures the overall information of the entire pool of tweets when i is large.

4.3.3 Concatenating with Base Features: Suppose $\hat{H}^{pool} \in \mathbb{R}^{n \times d}$ denotes the moving average matrix of all CVEs – the row dimension is the number of CVEs. To further augment the input features, we also concatenate $\hat{H}^{pool}$ with the base features H^{base}. Formally, the input to GCN layers is:

$$X^0 = (\hat{H}^{pool} \oplus H^{base}) W^{cat} \tag{8}$$

where $\oplus$ denotes concatenation of two matrices, and W^{cat} performs a linear transformation on the concatenated vector. The row dimension of X^0 is still equal to the number of nodes.

4.4 Graph Convolutional Networks

For one graph convolutional layer, we adopt the generalized aggregation and combination operators as in Eqs. (3) and (4), where one graph convolutional layer is defined as an aggregation layer followed by a combination layer. Recall from Eq. (3) that the aggregation here is used to aggregate the feature vectors of neighbors for a target node, and is different from the attention-based input aggregation in our proposed embedding layer.

The GCN-AE model is trained in an end-to-end manner starting with the raw input tweets F, followed by the attention-based embedding layer to get X^0. X^0 is then fed into the GCN layers. We use 2 GCN layers followed by a linear layer with the Sigmoid activation multiplied by 10 to get the final prediction of each node (CVE).

4.5 GCN-based Transductive Inference

There are two common ways of using GCNs: transductive learning and inductive inference. In our problem, whenever a new CVE is released by MITRE, we need to predict its CVSS score based on tweets. Instead of performing inductive learning where a generalized predictive model is learned to infer the score of a new CVE, we perform transductive inference where we use historical data to directly infer the score of a new CVE from its neighborhood in the CVE graph.

For transductive inference, our CVE graph is constructed with *both* training and test CVEs using their BoW features. As

a result, all CVEs are mutually related in the graph. During training, only training CVEs with known CVSS scores are utilized to define the loss function. In this paper, we use the mean squared error (MSE) as the loss function in the training process: $\mathcal{L} = -\frac{1}{|T|}\sum_{i \in T}(\hat{y}_i - y_i)^2$, where T is a training set of CVEs whose CVSS scores are *already known*, y_i is the ground-truth score of CVE $i \in T$, and $\hat{y}_i$ is the predicted score. It is easy to see that while the scores of the test CVEs are not included in the loss function, they still influence the training of the GCN-AE architecture since these test CVEs are still interrelated with the training CVEs with known scores. GCN-AE is optimized to minimize the training loss. At the same time, scores of the training CVEs are inferred as side products of the loss minimization process.

The GCN-AE architecture is trained end-to-end, with the following learnable parameters: i) the word vectors in each tweet sentence, which is used as the raw input to the bi-directional LSTM, ii) the neural network weights in the bi-directional LSTM, iii) the linear operation weight W^{attn} in the attention-based tweet content embedding module as well as W^{cat} after concatenating vectors, and iv) the parameters in the graph convolutional layers.

5 Experimental Evaluation

5.1 Experiment Settings

5.1.1 Pre-Processing: For each CVE, we collect tweets within 3 days of the first mention of the CVE in a tweet. We then remove some obviously useless tweets, e.g., tweets only with a CVE number and a URL without any hashtags or other text. We also remove stop words using the NLTK tool.

5.1.2 Evaluation Setting: We select the first 6 months of data (January 01 to June 30) for training and the next 8 weeks (July 01 to August 26) for testing. We use the first mention date of a CVE to split CVEs into train/test sets. For instance, a CVE that first appears on Twitter on June 25 will be treated as training data, while a CVE that is first mentioned on July 03 will be put into the test set. Moreover, in selecting the training set, we remove all CVEs whose CVSS scores are not released before the test date, so that it is guaranteed that we do not unfairly exploit future information that is not available.

For evaluation, we perform bi-weekly *rolling predictions*, where for every two weeks, we do predictions for the CVSS scores of the CVEs whose *first-mention date* is within the two weeks. For every two weeks, we train a different model. We use the ADAM optimizer [28] in our training process, with a learning rate of 0.005.

5.1.3 Hyper-parameters: For the aggregation function, we test k-max pooling and mean pooling to have h^{pool}. For the combination function, we test the 1D column-wise convolutions of LGCN which is more complex than the linear-based combination of GraphSAGE. For the attention-based LSTM in the embedding layer, the maximum sequence length is set to the maximum sentence length in our dataset. We maintain one vocabulary for the 8-month tweets, whose size is 4,584 words after removing stop words, numbers, and URLs. Each word in the vocabulary is embedded into a 48-dimensional vector in the input layer of the bi-directional LSTM. In each training epoch, 15 tweets are sampled for each CVE as inputs to the bi-directional LSTM. As described in Section 4.3.2, the moving average is adopted to capture latent information from the entire tweets pool. We use 128 as the dimension of the hidden vector for the bi-directional LSTM and finally get a 256-dimensional hidden vector after concatenating forward and backward hidden vectors for each CVE. For the edge creation threshold, we test $\kappa = \{0.8, 0.9\}$. The best hyperparameter configuration is the one with the lowest MSE loss in the training process.

5.2 Prediction Results

We first compare GCN-AE with several classical regression models (Linear Regression, Bayes Regression, Random Forest, Lasso, and Ridge Regression) which do not leverage correlations between CVEs. Hyper-parameter search is also conducted for the baseline methods to find the best hyper-parameter using 10-fold cross-validation.

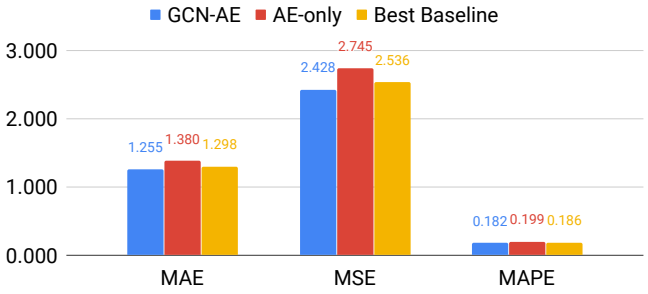

Fig. 3: Comparison results for CVSS score prediction.

5.2.1 Prediction Results: Figure 3 shows the results in terms of mean absolute error (MAE), mean squared error (MSE) and mean absolute percentage error (MAPE) for predicting CVSS scores using three approaches: i) "GCN-AE" which uses GCN-AE, ii) "AE-only" which uses only the attention-based embedding layer, iii) "Best Baseline" which is the *best result* among all the baselines described above. For all the metrics, GCN-AE yields the best results. GCN-AE obtains MAE of 1.255, and GCN-AE with direct regression is able to obtain MSE of 2.428 and MAPE of 0.182.

As an ablation study, we also compare with the AE-only model where we directly predict using X^0. We apply the same Sigmoid activation multiplied by 10 to X^0. Therefore, the difference between GCN-AE and AE-only is the existence of the GCN layers. Figure 3 shows that without the GCN layers, the prediction accuracy is sub-optimal, e.g. MAE of 1.255 in GCN-AE vs. 1.380 in AE-only.

5.3 Timeliness of VASE:

Figure 4 illustrates the prediction timing of VASE, where the X-axis is the number of days that the prediction of a CVE can be made before the NIST's official announcement, and the Y-axis is the corresponding number of CVEs. For instance, the point (20, 172) means that there are 172 CVEs

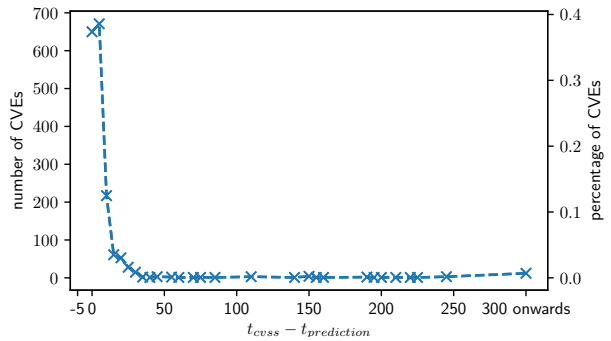

Fig. 4: CVE quantity distributions w.r.t. the number of days that prediction can be made to a CVE before NIST's official release of CVSS scores and attributes, collected from January to August, 2017.

whose CVSS scores can be predicted by VASE 20 days earlier than NIST. We see that, for almost all the CVEs, VASE can make predictions before NIST. VASE can make predictions for i) 37.85% of the CVEs one week earlier, ii) 6.04% of the CVEs one month earlier, and iii) 2.36% of the CVEs three months earlier. These cover a huge number of CVEs considering that there are tens of thousands of CVEs that are disclosed each year. As cyberattacks may happen any time after MITRE assigns CVE numbers, being able to advance the prediction by even one week is critical for CISOs around the world who can then decide how to allocate resources to protect their enterprise.

6 Conclusion

As pointed out in an earlier paper [4], there exists a gap of 132.7 days between the time a CVE number is assigned by Mitre and the time NIST releases the CVSS score for that vulnerability. System security officers around the world use CVSS scores in assessing what CVEs to patch and when. Having an early warning system to predict CVSS scores is therefore critical for protecting most companies. VASE is the first system to predict a CVE's severity score. VASE uses a novel mix of GCNs to leverage correlations among different CVEs, together with a novel attention-based embedding method to extract useful latent features from raw tweet texts for each CVE. Experimental results demonstrate that VASE is able to serve as a qualified early warning system before the detailed evaluations from NIST.

Acknowledgment

This work is supported by ONR grants N00014-18-1-2670 and N00014-16-1-2896 and ARO grant W911NF-13-1-0421.

References

[1] Moazzam Khan. Security analysts are overworked, understaffed and overwhelmed — here's how ai can help. *Security Intelligence*, 2018.

[2] Katheryn A. Farris, Ankit Shah, George Cybenko, Rajesh Ganesan, and Sushil Jajodia. Vulcon: A system for vulnerability prioritization, mitigation, and management. *ACM Transactions on Privacy and Security*, 21(4):16:1–16:28, June 2018.

[3] Sushil Jajodia, Noseong Park, Edoardo Serra, and V. S. Subrahmanian. Share: A stackelberg honey-based adversarial reasoning engine. *ACM Transactions on Internet Technology*, 18(3):30:1–30:41, March 2018.

[4] Haipeng Chen, Rui Liu, Noseong Park, and VS Subrahmanian. Using twitter to predict when vulnerabilities will be exploited. In *KDD*, pages 3143–3152, 2019.

[5] Fahmida Y. Rashid. Predict which security flaws will be exploited, patch those bugs. 2019. https://duo.com/decipher/predict-which-security-flaws-exploited-patch-those-bugs.

[6] Carl Sabottke, Octavian Suciu, and Tudor Dumitras. Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits. In *USENIX Security*, pages 1041–1056, 2015.

[7] Xiaojing Liao, Kan Yuan, XiaoFeng Wang, Zhou Li, Luyi Xing, and Raheem Beyah. Acing the ioc game: Toward automatic discovery and analysis of open-source cyber threat intelligence. In *CCS*, pages 755–766, 2016.

[8] Richard P Lippmann, Joseph P Campbell, David J Weller-Fahy, Alyssa C Mensch, and William M Campbell. Finding malicious cyber discussions in social media. Technical report, Massachusetts Institute of Technology, 2016.

[9] Rupinder Paul Khandpur, Taoran Ji, Steve Jan, Gang Wang, Chang-Tien Lu, and Naren Ramakrishnan. Crowdsourcing cybersecurity: Cyber attack detection using social media. In *CIKM*, pages 1049–1057, 2017.

[10] Haipeng Chen, Jing Liu, Rui Liu, Noseong Park, and V. S. Subrahmanian. Vest: A system for vulnerability exploit scoring & timing. In *IJCAI*, pages 6503–6505, 2019.

[11] Benjamin L Bullough, Anna K Yanchenko, Christopher L Smith, and Joseph R Zipkin. Predicting exploitation of disclosed software vulnerabilities using open-source data. In *Proceedings of the 3rd ACM on International Workshop on Security And Privacy Analytics*, pages 45–53. ACM, 2017.

[12] Shi Zong, Alan Ritter, Graham Mueller, and Evan Wright. Analyzing the perceived severity of cybersecurity threats reported on social media. *arXiv preprint arXiv:1902.10680*, 2019.

[13] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[14] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[15] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[16] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

[17] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[18] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2018.

[19] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. *CoRR*, abs/1806.01973, 2018.

[20] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[21] Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*, 2018.

[22] FIRST. Common vulnerability scoring system sig. 2019. https://www.first.org/cvss/.

[23] MITRE. Common vulnerabilities and exposures. 2019. https://cve.mitre.org/cve/.

[24] NIST. Vulnerability metrics. 2019. https://nvd.nist.gov/.

[25] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[26] Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A Smith. Transition-based dependency parsing with stack long short-term memory. *arXiv preprint arXiv:1505.08075*, 2015.

[27] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *CoRR*, abs/1609.01704, 2016.

[28] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.