Security Issue Classification for Vulnerability Management with Semi-supervised Learning

Emil Wåreus[1,2], Anton Duppils[1], Magnus Tullberg[1] and Martin Hell[2]
[1]*Debricked AB, Malmö, Sweden*
[2]*Dept. of Electrical and Information Technology, Lund University, Lund, Sweden*

Keywords: Machine Learning, Open-Source Software, Vulnerabilities, Semi-supervised Learning, Classification.

Abstract: Open-Source Software (OSS) is increasingly common in industry software and enables developers to build better applications, at a higher pace, and with better security. These advantages also come with the cost of including vulnerabilities through these third-party libraries. The largest publicly available database of easily machine-readable vulnerabilities is the National Vulnerability Database (NVD). However, reporting to this database is a human-dependent process, and it fails to provide an acceptable coverage of all open source vulnerabilities. We propose the use of semi-supervised machine learning to classify issues as security-related to provide additional vulnerabilities in an automated pipeline. Our models, based on a Hierarchical Attention Network (HAN), outperform previously proposed models on our manually labelled test dataset, with an F1 score of 71%. Based on the results and the vast number of GitHub issues, our model potentially identifies about 191 036 security-related issues with prediction power over 80%.

1 INTRODUCTION

In today's competitive environment Open-Source Software (OSS) enables organizations to leverage technology to better meet customer needs. A report from Synopsys found that 99% of codebases contain open source, and 70% of each codebase was open source (Synopsys, 2020). The increased exposure to open source software is often regarded as an enabler of higher productivity, but which may come at the cost of higher susceptibility to attacks enabled through vulnerabilities in OSS. To maintain control over the security of the proprietary software, the maintainers need to monitor vulnerabilities introduced through external software. This is part of what is known as Software Composition Analysis (SCA).

Vulnerabilities can be reported and be given a Common Vulnerabilities and Exposures (CVE) identifier and can then be stored in databases such as the National Vulnerability Database (NVD). This allows for a centralized repository and collection of vulnerabilities. However, not all vulnerabilities are given a CVE identifier. Even if such vulnerabilities are patched in the OSS component, it can be difficult for end users to identify the need to update the component to its most recent version. This will risk software being exposed to attacks.

In OSS, contributors, users, and community members often use `issues` to organize their work, specify requirements, and report bugs in the software. These issues may contain security-related information about the OSS, such as bugs with security implications, vulnerability reports, or information on a security update. Vulnerabilities reported as issues may sometimes not find their way to a CVE. Being able to classify issues as security-related is the first step towards assessing if they describe a vulnerability.

In this paper, we use Natural Language Processing (NLP) to automate the process of finding such security-related issues. Our model leverages unlabeled issues through Semi-Supervised Learning (SSL) to increase the performance during inference. SSL enables the model to better generalize to the task and to better learn the underlying software-related semantics in the issues. Our contributions are summarized as follows:

- We analyze, through Term-Frequency Inverse Document Frequency (TF-IDF) and truncated Singular Value Decomposition (SVD), the use of issue labels and CVE summaries as labeled training data, and show that such data should not be used to train an issue classifier.
- We describe how to model the problem with a

Wåreus, E., Duppils, A., Tullberg, M. and Hell, M.
Security Issue Classification for Vulnerability Management with Semi-supervised Learning.
DOI: 10.5220/0010813000003120
In *Proceedings of the 8th International Conference on Information Systems Security and Privacy (ICISSP 2022)*, pages 84-95
ISBN: 978-989-758-553-1; ISSN: 2184-4356

Hierarchical Attention Network (HAN) with Virtual Adversarial Training (VAT) and show that this model provides better results than previously published models.

The result is a state-of-the-art classification of GitHub issues into security and non-security related issues. We also provide our manually labeled test dataset for future comparison[1].

The paper is outlined as follows. Section 2 provides some background on NLP, vulnerabilities, issues, and our method of evaluation. In Section 3, we describe and analyze the data that is used in our model. Then, the model is detailed in Section 4, followed by the results in Section 5. We compare our approach and results to related works in Section 6, before the paper is concluded in Section 7.

2 BACKGROUND

2.1 Natural Language Processing

NLP is the task to make computers understand linguistics, usually with the support of machine learning. Within NLP, tasks such as machine translation, document classification, question answering systems, automatic summary generation, and speech recognition are common (Khurana et al., 2017). One of the main advantages of using machine learning for NLP is that the algorithms may gain a contextual semantic understanding of text where classifications are not dependent on a single word, but rather a complex sequence of words that can completely alter the meaning of the document. This is beneficial in the endeavor to find vulnerabilities through a classification of issues, as the context of single words in these documents may matter.

Supervised classification methods require lots of labeled training data. As in many real-world use cases of machine learning, there is a limit to the amount of available training data. In a semi-supervised approach, a more limited set of labeled training data can be used together with a large set of unlabeled data. There are multiple different techniques for leveraging the unlabeled data during training. In our approach, we use a technique called Virtual Adversarial Training (VAT), which helps the model generalize by altering the input of labeled examples during training. VAT was chosen as the SSL-method due to that it is usable with already implemented neural networks, which makes it good to benchmark SSL vs non-SSL methods.

[1]All data used for training, validation and tests will be made publically available if the paper is accepted.

2.2 Vulnerabilities and Security Related Issues

Vulnerability data is to a large extent centralized through CVE identifiers, maintained by Mitre. Each vulnerability also comes with a very short summary, describing e.g., the affected software, the nature of the vulnerability, and the potential impact in plain text. These vulnerabilities are collected in NVD, which adds further information useful for better understanding and analyzing the vulnerabilities. This includes a severity score and Common Platform Enumeration (CPE) identifiers that uniquely identify the product and versions that are vulnerable. Some CVE entries also relate the vulnerability to the underlying weakness by providing a Common Weakness Enumeration (CWE) identifier. NVD currently collects around 15k-20k data annually, and the database currently contains around 150k vulnerabilities.

GitHub is the world's largest host of source code and provides services such as source code management, version control, issue tracking, and continuous integration. Issues are used to track ideas, enhancements, bugs, and tasks related to a repository. Issues can be assigned metadata for categorizing it, e.g., bug, feature, question, etc. In particular, the security label can be used to mark that an issue is security-related. However, it is at the user's discretion to choose labels, so this tag may or may not be accurate, depending on the project and developers. For purposes of accessing GitHub data, an API is provided that allows anyone to fetch GitHub information without having to parse web pages.

2.3 Evaluation in Machine Learning

Data is typically divided into three independent sets. First, the training data is used to train the ML model. Then, a validation dataset is used to monitor model performance during development. Finally, a test set is used to evaluate the performance of the model. In the evaluation, standard tools and notions include *precision*, *recall*, *F1*, and *receiver operating characteristic - area under curve (ROC-AUC)*. Denote the number of true positives as TP, false positives as FP, true negatives as TN, and false negatives as FN. Fixing the decision threshold for a classifier, the first three are then defined by

$$precision = \frac{TP}{TP+FP}, \qquad recall = \frac{TP}{TP+FN},$$

and their harmonic mean *F1*

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}.$$

Our algorithms will be optimized for *F1* in order to reduce the risk of building a trivial classifier.

ROC-AUC is derived from the integral of the curve that is created from varying the classification threshold for the true-positive rate (tpr) and the false-positive rate (fpr). These are derived from

$$tpr = \frac{Positives\ correctly\ classified}{Total\ positives} = \frac{TP}{TP+FN},$$

and

$$fpr = \frac{Negatives\ incorrectly\ classified}{Total\ negatives} = \frac{FP}{FP+TN}.$$

3 UNDERSTANDING AND EXPLORING THE DATA

As a first step, we need to have a good understanding of the underlying data. We divide this process into *data acquisition*, *data cleaning*, and *exploratory analysis*. The results will be used when developing the classification model.

3.1 Data Acquisition

Three different data-sources were used.

- Issues from GitHub, where over 7 000 000 issues were acquired from the GitHub API, covering over 30 000 of the most popular repositories according to the number of stars.
- CVEs from NVD, which can safely be considered as security-related in their text-semantics.
- SecureReqNet (SRN) (Palacio et al., 2019), where the authors provided a labeled open-source dataset of security-related issues collected from GitHub, NVD, and GitLab. The labels for this dataset were automatically generated from issue-labels, where 10% of the data has been sampled for quality assurance.

GitHub issues include several data points, including e.g., title, body, creation date, labels, closing date, and references to commits and releases in which the issue was closed. A CVE, as provided by NVD, consists of e.g., the CVE id, when it was updated, a brief summary of the vulnerability, a CWE identifier, links to other resources, and a list of CPEs.

The data from GitHub and SRN primarily consist of community-generated data in a weakly controlled environment with varying quality of each data point. In contrast, CVE data stems from a more controlled environment with higher data quality. While 150k security-related texts (CVE summaries) appear to be excellent candidates for training data, this fundamental difference requires careful analysis. Otherwise, there is a risk that an NLP model would only learn to differ between data sources. Issues can be linked to CVEs by being listed as an external resource on NVD. Such issues can safely be regarded as security-related. This provided us with an additional 941 security-related issues from GitHub, among which 847 are used for training and 94 for validation.

Issues may be linked to labels in a zero-to-many relationship. Labels can provide additional signal to the distinction between security-related and non-security-related issues. Thus, another candidate for training data are issues with an explicit, user-generated, security label. These issues will also be explored in more detail in order to determine their suitability for training the classifier.

3.2 Data Cleaning

To prepare the data for further analysis and classification we must clean noisy parts of the data that will not contribute signal to the model. To properly work with text, input documents are tokenized, i.e., split into tokens, such as words and punctuation. To give the model as useful tokens as possible, the following cleaning measures were implemented:

- Non-English documents were removed.
- All emojis were removed.
- Everything within the HTML-tag *code* was removed.
- All text was converted to lowercase.
- Special characters were removed and end of sentences were replaced with a special tag.
- Very long (over 60,000 characters) and very short (under 10 characters) documents were removed.
- Words were stemmed to their root word.

The reason to remove code from the documents is that code and text are vastly different and would require a separate model to accurately extract signals from the code.

3.3 Exploratory Analysis

We start by exploring the labels. Since these are user-generated, labels with different casing, wording

and format may in fact be the same underlying label. To handle different variations of the same label, all labels were clustered using character-level term frequency–inverse document frequency (TF-IDF) and K-means clustering, and similar labels were aggregated to the resulting clusters. Each cluster name was determined by the most common label in that particular cluster. The *security*-labeled issues represent about 0.2% of all labels, while *help wanted* makes up 23%, *bug* makes up 13%, and *enhancement* makes up 9%. It is clear that the share of security-labeled issues was very low, which suggests that labels should not be included in the model. Furthermore, variations within label groups were significant and may provide too much noise to the model. This conclusion will also be supported by the data visualization below.

To evaluate the potential use of CVE summaries as training data for security-related text, we first extract bigrams and trigrams from both GitHub issues and CVEs. The ten most common bigrams/trigrams are given in Table 1. Looking at these samples, there is no apparent similarity between the two datasets in terms of term frequency.

To verify this non-similarity further, we perform a dimensionality reduction to visualize this. The cleaned issues and CVE-summaries were fed into a term frequency–inverse document frequency (TF-IDF) model, which calculates the term importance of each term in a document in relation to the full corpus. The resulting high dimensional sparse matrix is a numerical representation of the input documents. For visualization, the dimensionality of the data was reduced with a truncated singular value decomposition model (Truncated SVD) (Halko et al., 2011). Fig. 1 can be viewed as term-frequency-similarity between different data labels. It is clear that there is a large difference in term-frequency distribution between CVE-summaries and issues. There is also no meaningful difference between security-related issues and non-security related issues, which makes this a non-trivial problem. In conclusion, this tells us that CVEs and issues use vastly different words in their text documents, which may be expected as CVEs are formally written by a very limited number of people, while issues can be written by anyone. Moreover, since there is no significant difference between the different labels, and all labels use a significantly different language than CVEs, the label information is not considered in our model.

From this analysis, we decided to exclude CVE-summaries from the training phase of the model. If such data would be included, the model would simply become a classifier that determines the source of the input data, rather than a classifier that determines the security-relevance of the document.

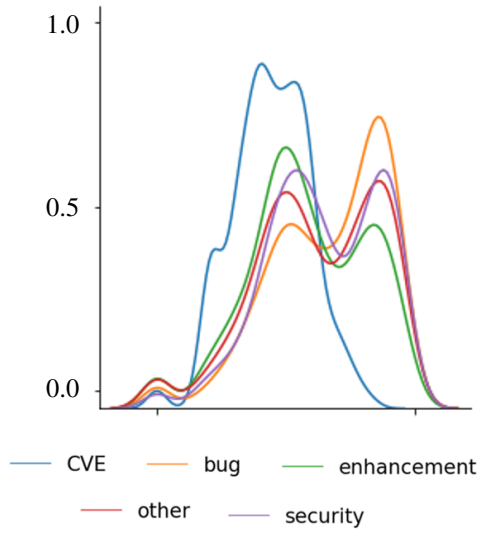

Figure 1: TF-IDF and Truncated SVD over cleaned issues and cleaned CVEs, where the distribution of the first feature is visualized. The labels enhancement (13%), bug (10%), and security (0.2%) were picked out from the issue dataset, and all other issues are marked as other. 1000 data-points were sampled from each class to visualize the distributions.

Since security-related and non-security related issues are not linearly separable, to build an effective model for a classifier, a non-linear model is required. To further increase the performance we will explore the option of using semi-supervised learning, which uses unlabeled examples to help the model generalize well.

A summary of the data used for training and validation is given in Table 2. It is a mix between data provided by SRN (Palacio et al., 2019) (their training and validation sets, unfortunately, they did not release their test set) and our own dataset collected from GitHub. We split the datasets into four different sets for clear validation. The labeled training set consists of examples from the SRN training set, issues from our GitHub dataset that are references in CVEs from NVD, as well as unlabeled data from our own GitHub dataset. The validation set consists of a hold-out subset from the SRN training set. We test our final model on the SRN Validation Dataset to enable a fair comparison of results. The User Labeled Test Dataset is a separate test set consisting of sampled issues, uniformly sampled from our own GitHub dataset, that has been manually labeled by us with the Annotation Guidelines described in the Appendix .1. The User Labeled Test Dataset gives us a gold standard to compare the SRN datasets to, as these sets are automatically generated as described in (Palacio et al., 2019).

Table 1: The top 10 Bigrams and Trigrams from NVD CVE summaries and cleaned GitHub Issue bodies respectively. (There is no relationship between bigrams and trigrams on the same line more than them having same ranking within their own dataset.).

Bigram		Trigram	
NVD Summaries	**Github Issues**	**NVD Summaries**	**Github Issues**
remote attackers	step reproduc	allows remote attackers	unknown sourc java
allows remote	java org	cause denial service	java desktop java
execute arbitrary	expect behavior	attackers execute arbitrary	desktop java awt
denial service	node modul	remote attackers execute	sourc java desktop
cause denial	py line	execute arbitrary code	java awt eventdispatchthread
attackers execute	unknown sourc	cross site scripting	java org apach
arbitrary code	oper system	attackers cause denial	avail avail avail
via crafted	java lang	site scripting xss	python site packag
cross site	java android	remote attackers cause	java android view
attackers cause	java awt	arbitrary web script	java awt eventqueu

Table 2: Summary of the data used in our training, evaluation, and testing. Note that only 3M of the available 7M of the unlabeled GitHub issues were used for training. This was due to time-complexity and diminishing returns.

Dataset	GitHub	Gitlab	Source
Train Dataset			
non-security related	47095	460	SRN
security	3691 (2844, 847)	452	SRN + Author
unlabeled	3M	0	Author
Validation Dataset			
non-security related	4683	66	SRN
security	453 (359, 94)	55	SRN + Author
SRN Validation Dataset (used as a test set)			
non-security related	555	0	SRN
security	514	0	SRN
User Labeled Test Dataset			
non-security related	835	0	Author
security	112	0	Author

4 MODELING

In this section, we describe the model used for our classifier. The amount of labeled training data, in particular security-related, is very limited, requiring a semi-supervised model. We describe a Hierarchical Attention Network, and then show how we combine this with Virtual Adversarial Training in order to support a semi-supervised approach.

4.1 Hierarchical Attention Network

Hierarchical Attention Network (HAN) was introduced in (Yang et al., 2016) to better model full documents with an attention-based neural network (Vaswani et al., 2017). It tries to mirror the input document by having one attention-mechanism for the word level, and one for the sentence level in a hierarchical structure. This helps the model learn sparse semantics in documents and creates a better contextual representation of important terminology, attractive properties for our task.

The model takes the stemmed word sequences as input $w_{it}, t \in [1,T]$, for the i-th sentence and the t-th word in the sequence length T. The text is converted to a numerical representation through pre-trained embeddings provided by SRN (Palacio et al., 2019), which is represented as matrix W_e. From the numerical word vectors, seen as input in Fig. 2, a bi-directional contextual word-encoding is derived from two LSTM (Hochreiter and Schmidhuber, 1997) layers running in different directions on the input sequence. The outputs from the LSTM-cells are derived as

$$x_{it} = W_e w_{it}, \quad t \in [1,T], \tag{1}$$

$$\overrightarrow{h_{it}} = \overrightarrow{LSTM}(x_{it}), \quad t \in [1,T], \tag{2}$$

$$\overleftarrow{h_{it}} = \overleftarrow{LSTM}(x_{it}), \quad t \in [T,1], \tag{3}$$

and the bi-directional results are concatenated as $h_{it} = [\overrightarrow{h_{it}}, \overleftarrow{h_{it}}]$. Then, h_{it} is used as input to the word-level attention layer, which directs the model's attention to more important words, e.g., buffer-overflow, XSS, and injection. Such words contribute more signal to positive security classification. The attention mechanism is derived from

$$u_{it} = \tanh(W_w h_{it} + b_w), \tag{4}$$

$$\alpha_{it} = \frac{\exp(u_{it}^T u_w)}{\sum_t \exp(u_{it}^T u_w)}, \tag{5}$$

$$s_i = \sum_t \alpha_{it} h_{it}, \tag{6}$$

where u_{it} is the output from the learned weights and biases that transforms every single word. This output is the prediction-power for the attention for that particular word. Then, Eq. (5) normalizes the attention for sentence i and multiplies u_{it} with a trainable context vector u_w to find the relevant words to give attention to. Eq. (6) multiplies each word with its attention to either increase or decrease its relative importance.

After the word level attention, sentences are encoded to further propagate through the model. The sentence encodings are then derived from another bi-directional LSTM-layer as

$$\overrightarrow{h_i} = \overrightarrow{\text{LSTM}}(s_i), \quad s \in [1, L], \tag{7}$$

$$\overleftarrow{h_i} = \overleftarrow{\text{LSTM}}(s_i), \quad s \in [L, 1], \tag{8}$$

and concatenated to $h_i = [\overrightarrow{h_i}, \overleftarrow{h_i}]$. From this transformation, h_i becomes a contextual representation of sentence i that considers the results from neighboring sentences. This is then passed on to a sentence-level attention layer, derived in a similar manner as word-level attention, but with variables trained with sentence-level context. The output v (see Fig. 2) is then used for further modeling. A visualization of word- and sentence-layer attention is given in Fig. 3, where high impact word and sentences are shown in red.

4.2 Adversarial Training

Adversarial Training (Goodfellow et al., 2014) is a supervised learning method based upon creating adversarial examples. These are created by slightly modifying existing examples, making the model misclassify the adversarial example. The idea is to use observations that are very close in input space but are very different in their output. For these, there exists a small variation to the input data, perturbations, that will make the model misclassify that example by adding the perturbation to the input data, which creates the adversarial examples. By training on these adversarial examples, the model can regularize and generalize better.

Adversarial Training modifies only the loss function and can thus be applied to already existing models. Denote x as the input, y as the label paired with x, θ as the parameters of the model, $\hat{\theta}$ as the parameters with a backpropagation stop, and r as a small uniformly sampled perturbation with the same dimension as x. The adversarial loss L_{adv} is then given by

$$L_{adv}(\theta) = -\log p(y|x + r_{adv}; \theta), \tag{9}$$

where

$$r_{adv} = \arg\min_{r, ||r|| \leq \varepsilon} \log p(y|x + r; \hat{\theta}). \tag{10}$$

The ε is a hyperparameter that restricts the absolute value of r. Stopping the backpropagation in $\hat{\theta}$ means that the backpropagation algorithm should not be used to propagate the gradients in the case of $\hat{\theta}$.

4.3 Virtual Adversarial Training

Virtual Adversarial Training (VAT) (Miyato et al., 2015) is an extension of Adversarial Training, making it accessible in a semi-supervised environment. Instead of using the labels to determine the perturbations, the direction of the gradient is followed using an approximation. This is done by calculating the Kullback-Leibler divergence (D_{KL}) between the input probability distribution and the input probability distribution plus a small random perturbation.

The D_{KL} between two discrete probability distributions P and Q over the same probability space χ is defined as

$$D_{KL}[P||Q] = \sum_{x \in \chi} P(x) \log \left(\frac{P(x)}{Q(x)} \right). \tag{11}$$

The VAT cost is given by

$$L_{v\text{-}adv}(\theta) = D_{KL}[p(\cdot|x, \hat{\theta})||p(\cdot|x + r_{v-adv}; \theta)], \tag{12}$$

where

$$r_{v\text{-}adv} = \arg\max_{r, ||r|| \leq \varepsilon} D_{KL}[p(\cdot|x; \hat{\theta})||p(\cdot|x + r; \hat{\theta})]. \tag{13}$$

A classifier is trained to be smooth by minimizing Eq. (12), which can be seen as making the classifier resilient to worst-case perturbation(Miyato et al., 2015).

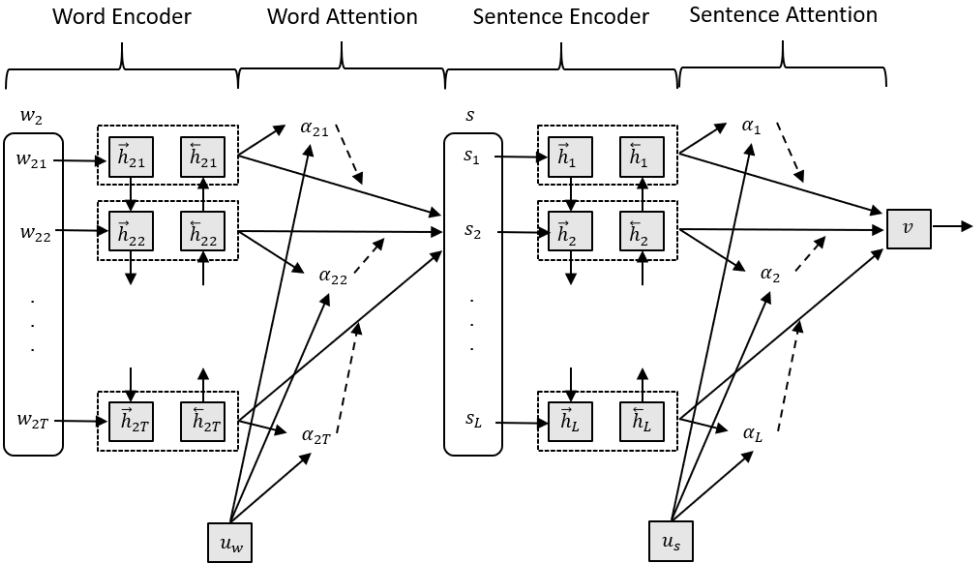

Figure 2: The structure of HAN.

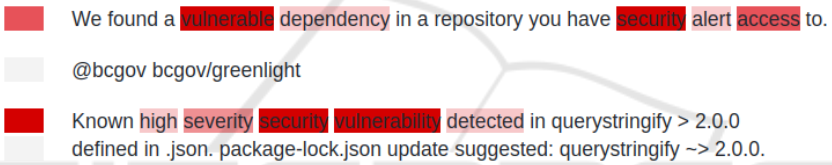

Figure 3: Example of attention mechanism both for word level and sentence level attention. The red word highlights indicates relevance to the sentence. The red highlights to the left of each sentence shows the relevance of each sentence to the document. Grey or non-highlighted words are deemed irrelevant to the core message of the document.

4.3.1 VAT in Text Classification

The original formulation of VAT, as described in Section 4.3, does not consider sequential data with arbitrary length. Therefore, the technique needs to be repurposed for this case, as proposed in (Miyato et al., 2017). Let s be a sequence containing word embeddings, $s = [\hat{v_1}, \hat{v_2}, \ldots, \hat{v_T}]$ where $\hat{v_i}$ is a normalized word embedding derived as

$$\hat{v_i} = \frac{v_i - E(v)}{\sqrt{Var(v)}}. \tag{14}$$

By using a sequence of word embeddings as the input instead of the sequence of the tokenized words, applying the perturbations obtained from the VAT-calculation directly on the embeddings will create adversarial examples suitable for text, as shown in Fig. 4.

In VAT for text classification, the approximated virtual adversarial perturbation is calculated during the training step as

$$L_{v\text{-}adv}(\theta) = \frac{1}{N'} \sum_{n'=1}^{N'} D_{KL}[p(\cdot|s_{n'};\hat{\theta})||p(\cdot|s_{n'} + r_{v\text{-}adv,n'};\theta)], \tag{15}$$

where

$$r_{v\text{-}adv} = \varepsilon g/||g||_2, \tag{16}$$

and

$$g = \nabla_{s+d} D_{KL}[p(\cdot|s;\hat{\theta})||p(\cdot|s+d;\hat{\theta})]. \tag{17}$$

N denotes the number of labeled examples and the unlabeled examples are denoted as N'. The symbol ∇_x is the gradient using the observation x during back-propagation.

4.4 Hierarchical Attention Virtual Adversarial Network

The HAN architecture is also expanded with a VAT-implementation. Hierarchical Attention Virtual Adversarial Network (HAVAN) still retained the HAN-layer structure, but with some extra SSL steps added to it. The embeddings are normalized using Eq. (14). The loss $L_{v\text{-}adv}$ from Eq. (15) is then added to the loss

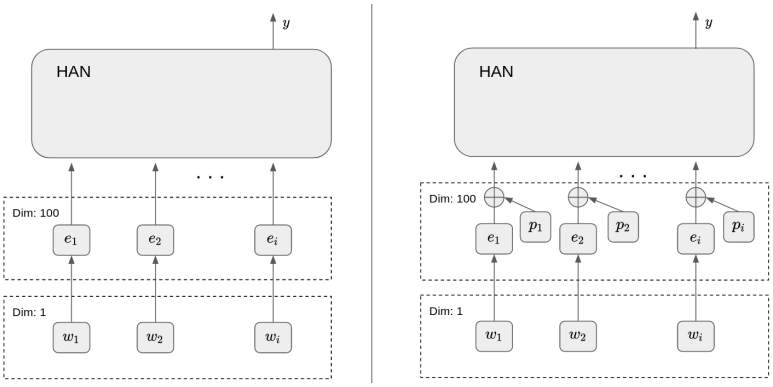

Figure 4: An overview of embeddings with HAN (left), and the perturbed embeddings with HAN (right). Dim is the output dimension of each layer and *y* is the output of the network.

function as well as the option to perturb the embeddings of the model during a training step. In HAVAN, both labeled and unlabeled data is used during training, making it an SSL-based approach. Labeled data is used for the standard loss function, while both unlabeled and labeled data are used for the VAT loss function.

Since we have a binary classification problem, the Bernoulli distribution is used as the distributions in Eq. (15). An overview of the model can be seen in Fig. 5.

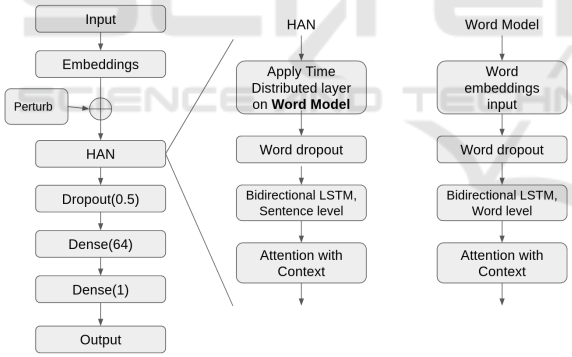

Figure 5: Overview of the layer structure of HAVAN (HAN with VAT). Perturb is a perturbation that is added to the embeddings.

5 RESULTS AND DISCUSSION

5.1 Main Results

We evaluate four different models. A simple *logistic regression* is used as a base model. The *SRN* from (Palacio et al., 2019) was adapted for comparison with the two models proposed in this paper, namely the *HAN* and the *HAVAN* models as detailed in Section 4. The SRN model was retrained on the same data with configuration according to their GitHub Repository (SEMERU-Lab, 2021), and optimized to equal extent as other models. All models are evaluated against the automatically generated SRN Validation Dataset and our User Labeled Test Dataset. The results are presented in Table 3. The best performance in each security category is presented for each dataset with bold numbers.

For the *SRN Validation Dataset*, it is clear that the HAN-model outperforms the other models with a macro average of 73% F1, and security classification of 65% F1.

For this dataset, all models have very high precision scores and quite low recall, which may indicate that the models are finding somewhat similar examples, but not other variants of security issues. This is an indication of a skewed dataset, which is not uncommon when labels are automatically generated as in the case of the SRN Validation Dataset. To further analyze the skewness of the dataset, it would be interesting to link the training and validation set-issues to CWEs when applicable. CWE information could indicate if we have higher or lower performance for certain types of security weaknesses.

Looking at the *User Labeled Test Dataset*, one can observe a significant decrease in performance in terms of classifying as security-related, with the best performing modeling being HAVAN at a macro F1 of 71%, and a security F1 of 48%. The most probable reason for the different levels of performance is that the different testing sets draw data from different distributions, as one is automatically generated and one is manually labeled. It may be that the User Labeled Test Dataset is more inclusive in its definition of security-related issues, which can be analyzed in the Annotations Guidelines in Appendix .1.

In Fig. 6, we can see the 95% confidence of the models with error on the y-axis, and it is clear that the variance is quite high for the User Labeled Test

Dataset as it contains quite few examples. This makes a good argument to try to label more data with high confidence on its security relevance to further increase the testing and training capabilities. A larger, less skewed training set would also further increase generalization to the underlying challenge and real-world performance of the model.

Looking at the AUC in Table 5, we observe that the models achieve a much higher score on the SRN Validation Dataset than the User Labeled Test Dataset. This is because the training set draws from the same, possibly skewed, distribution as the SRN Validation Dataset. The Logistic Regression outperforms the other models in AUC, which means that there is a more distinct separation of the distributions of security and non-security. This may be an attractive property if the predictions are used as a "prioritized list" to pop for further analysis, which may be a valid use-case.

5.2 Findings from Predictions

Perhaps against intuition, the closing rate of security-related issues is lower than the global closing rate. In our results, we found that issues with security prediction power over 80% had a closing rate of 80.54%, and the global closing rate is 81.45%. This is also visible in the longevity of issues, as it takes on average 11.6% longer to close a security issue in comparison to a non-security related issue. On average, it takes 119 days to close an issue with security prediction power above 80%, in comparison to 107 days for issues with a security prediction power $\leq 50\%$. In total, with predictions on 7M issues, we found 497 019 (7%) issues with a prediction power over 50%, and 191 036 (2.7%) with over 80%. These numbers can be compared to the existing 158 000 vulnerabilities on NVD, indicating that our approach could potentially be used to additionally identify and enumerate a large number of new vulnerabilities. Centralizing all vulnerabilities allows more efficient processes to identify, evaluate and prioritize vulnerabilities in software, and subsequently to apply adequate remediation. As of now, there are over 72M issues on GitHub, which could result in about 2M security-related issues with a prediction power of over 80%, assuming the distribution is representative.

6 RELATED WORK

In (Palacio et al., 2019), Palacio et. al implemented a model to perform the same task, but with a different model architecture and a supervised approach. They released an open-source version of their model and data called SecureReqNet (SRN). They use a convolutional neural network (CNN) with a strong analogy to N-gram features in the documents and achieve a performance of 98.6% AUC, but evaluate on a different dataset than our experiments as their test set is not publicly available. Their dataset was automatically derived from CVE-references and validate by experts through random-sampling. This process opens up the model to bias towards security issues with strong link to NVD, and may not be a fit representation of security related issues in a broader sense, and thus not an accurate representation of the reality. In this paper, we derived our final testing set from completely random-sampled issues, with no prior bias, to get a better representation of the real underlying dataset. We call this the User Labeled Test Dataset, and a substantial difference in performance is presented in Table 3 between our model and the model presented in (Palacio et al., 2019).

In (Chen et al., 2020) the authors are using a SSL to find vulnerability candidates from commit-messages, issues, pull requests, and patches, with a high F1-measure of 70.5% (72% recall, 69% precision). Their approach uses *self-learning* (Nigam and Ghani, 2000) as their SSL-approach, which may treat predictions on unlabeled examples as labels on succeeding training iterations. This SSL-approach drastically improves their performance, which also implies the hypothesis that unlabeled data should be used to increase performance for this task. This work is based on work from some of the same authors where they conduct similar research without SSL (Zhou and Sharma, 2017).

The authors of (Zou et al., 2018) present a solution to distinguish security related bug reports and non-security related bug reports. The model was trained with a supervised approach with textual and meta-features extracted from 23 608 reports from Bugzilla with bugs in Firefox, Seamonkey, and Thunderbird. For this, more narrow approach, they achieved the strong F1 of 88.6% (79.9% recall, 99.4% precision).

7 CONCLUSION

We propose the use of a Hierarchical Attention Network (HAN) to classify GitHub issues as security related. To increase the amount of training data, we also propose to use Virtual Adversarial Training (VAT). The models are compared to the previously proposed SRN model using both the automatically labeled SRN validation set as testset and a manually labelled testset provided in this paper. Comparing the models, the

Table 3: The best results for each model over the different test sets. Bold entries shows the best result for security classification in each column.

	SRN Validation Dataset			User Labeled Test Dataset		
	Precision	**Recall**	**F1 score**	**Precision**	**Recall**	**F1 score**
Logistic Regression						
non-security related	65%	99%	79%	92%	92%	92%
security	**99%**	42%	59%	40%	**39%**	40%
macro average	81%	72%	69%	66%	66%	66%
SRN						
non-security related	66%	99%	79%	91%	89%	90%
security	97%	44%	61%	30%	36%	33%
macro average	81%	71%	70%	61%	62%	61%
HAN						
non-security related	68%	99%	80%	91%	97%	94%
security	97%	**49%**	**65%**	57%	33%	42%
macro average	**82%**	**74%**	**73%**	74%	65%	68%
HAVAN (HAN w/ VAT)						
non-security related	66%	99%	79%	92%	98%	95%
security	97%	44%	61%	**75%**	35%	**48%**
macro average	**82%**	72%	70%	**83%**	**67%**	**71%**

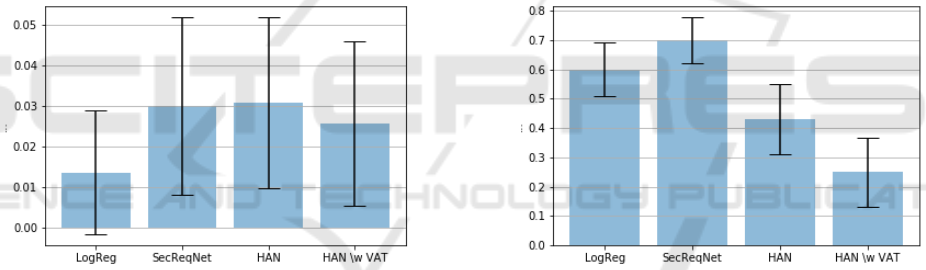

Figure 6: The average error on security related data with its 95% confidence interval for the shows the SRN Validation Dataset (a) and the User Labeled Test Dataset (b). The y-axes represent the average error. Note that the scales in the two graphs are different.

Table 4: The Area Under Curve (AUC) score for each model and test set.

Logistic Regression	
SRN Validation Dataset	**0.962**
User Labeled Test Dataset	**0.729**
SRN	
SRN Validation Dataset	0.955
User Labeled Test Dataset	0.634

HAN	
SRN Validation Dataset	0.932
User Labeled Test Dataset	0.666
HAVAN (HAN w/ VAT)	
SRN Validation Dataset	0.939
User Labeled Test Dataset	0.707

HAN model outperforms the other models on the automatically generated test data, though the differences are not dramatic for any of the macro average scores. For the manually labeled dataset, the HAVAN model gives the best results, with precision being particularly strong. At the same time, SRN has the worst performance for this dataset.

It seems clear that there are performance advantages to use a model tailored to full document classification, the HAN-model, for classifying issues. The attention mechanisms could also enable deeper analysis of important parts of each document, and even potentially UX-capabilities with short summaries of each document where the sentence with the most at-

Table 5: The AUC score for each model using the SRN Validation Dataset (SRN) and our User Labeled Test Dataset (UL).

	SRN	UL
Logistic Regression	**0.962**	**0.729**
SRN	0.955	0.634
HAN	0.932	0.666
HAVAN (HAN w/ VAT)	0.939	0.707

tention could be presented to a user. It also seems advantageous to leverage the vast number of unlabeled examples with semi-supervised learning to classify issues as security-related. Still, for our approach, it is important to note that the number of labeled relevant security examples is relatively few in comparison to the full unlabeled dataset. To enable the use of more aggressive SSL-methods, there is a need to acquire more labeled examples.

REFERENCES

Chen, Y., Santosa, A. E., Yi, A. M., Sharma, A., Sharma, A., and Lo, D. (2020). A machine learning approach for vulnerability curation. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR '20, page 32–42. Association for Computing Machinery.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. arXiv:1412.6572.

Halko, N., Martinsson, P. G., and Tropp, J. A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

Khurana, D., Koli, A., Khatter, K., and Singh, S. (2017). Natural language processing: State of the art, current trends and challenges. arXiv:1708.05148.

Miyato, T., Dai, A. M., and Goodfellow, I. (2017). Adversarial training methods for semi-supervised text classification. arXiv:1605.07725.

Miyato, T., Maeda, S.-i., Koyama, M., Nakae, K., and Ishii, S. (2015). Distributional smoothing with virtual adversarial training. arXiv:1507.00677.

Nigam, K. and Ghani, R. (2000). Analyzing the effectiveness and applicability of co-training. In *Proceedings of the Ninth International Conference on Information and Knowledge Management*, CIKM '00, page 86–93. Association for Computing Machinery.

Palacio, D. N., McCrystal, D., Moran, K., Bernal-Cárdenas, C., Poshyvanyk, D., and Shenefiel, C. (2019). Learning to identify security-related issues using convolutional neural networks. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 140–144.

SEMERU-Lab (2021). SecureReqnet. https://github.com/WM-SEMERU/SecureReqNet.

Synopsys (2020). 2020 open source security and risk analysis. https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., and Hovy, E. (2016). Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489. Association for Computational Linguistics.

Zhou, Y. and Sharma, A. (2017). Automated identification of security issues from commit messages and bug reports. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, page 914–919. Association for Computing Machinery.

Zou, D., Deng, Z., Li, Z., and Jin, H. (2018). Automatically identifying security bug reports via multitype features analysis. In Susilo, W. and Yang, G., editors, *Information Security and Privacy*, pages 619–633, Cham. Springer.

APPENDIX

A Annotation Guidelines

An annotation policy was established in order to make the annotation process more efficient and to favor repeatability and reproducibility. All data in the User Labeled Test Dataset was annotated by one of the authors with knowledge in the field of cybersecurity, a condition that must be met in order to adequately label data as relating to cybersecurity. Some data was annotated by multiple parties and compared in the cases of mismatch to ensure the annotations were similar.

Many issues were ambiguous and unclear, making it important to create a policy. The annotation guideline was used to establish a unified labeling method. It was updated regularly during the annotation phase whenever a new kind of case arose. The categories do not discriminate between questions, warnings, or other discussions about a certain topic. The text is annotated as the most severe category that accurately describes it. The priority goes from Vuln being highest to Safe being lowest.

Vuln: Presence of known exploits, user-reported vulnerabilities.

Risk: Commonly exploited methods such as unrestricted user input, memory leaks, unexpected/unintended r/w/e os/database access, overflows, user-reported potential risk, segmentation fault, access violation.
Caution: Breaking changes, breaking dependencies, breaking compilation, breaking updates, installation issues, authentication problems, port or socket malfunctioning, firewall issues service unavailable, site down, failed tests, out of memory, crash due to instabilities, unexpected/unintended r/w/e os/database deny, broken links, unknown CPU usage (mostly high usage with no obvious reason for it), incorrect mathematical calculations (with potential side effects), runtime errors, unknown memory issues, configuration problems of server, error-flags concerning security, talks about computer security in some way.
Unsure: Unexpected behavior, minor breaking changes (e.g., new functionality that has not been used in production in a previous version), lack of confidence in its safety, UI bugs, development mode only issues
Safe: Text does not cover topics concerning the categories above, such as issues asking for help with potential programming mistakes.

During the evaluation, the issues labeled with Vuln, Risk, and Caution were considered security-related in our binary classification. Unsure and Safe was considered not security-related.