Vulnerability Detection with Deep Learning

Fang Wu[1], Jigang Wang[2], Jiqiang Liu[1], Wei Wang*[1]
[1]Beijing Key Laboratory of Security and Privacy in Intelligent Transportation,
Beijing Jiaotong University, 3 Shangyuancun, Beijing 100044, China
[2]ZTE Corporation, Nanjing, China
e-mail: {wufangjsj, jqliu, wangwei1}@bjtu.edu.cn, wang.jigang@zte.com.cn

***Abstract*—Vulnerability detection is an import issue in information system security. In this work, we propose the deep learning method for vulnerability detection. We present three deep learning models, namely, convolution neural network (CNN), long short term memory (LSTM) and convolution neural network - long short term memory (CNN-LSTM). In order to test the performance of our approach, we collected 9872 sequences of function calls as features to represent the patterns of binary programs during their execution. We apply our deep learning models to predict the vulnerabilities of these binary programs based on the collected data. The experimental results show that the prediction accuracy of our proposed method reaches 83.6%, which is superior to that of traditional method like multi-layer perceptron (MLP).**

Keywords-vulnerability detection; deep learning; convolution neural network; long short term memory

I. INTRODUCTION

Software is one of the most important components of information system and information products. Vulnerabilities in software have become the determinants that directly affect the security of information systems. It has been proven that most information security incidents are initiated by the attackers using software vulnerabilities. In recent years, such incidents have become increasingly serious. As a consequence, an emerging and considerable issue for software programmers is to identify which components of software are vulnerable and require further analysis to estimate their grade of risk, and if necessary, to rapidly assign appropriate patches.

A. Vulnerability Detection

Many various vulnerability discovery procedures have been presented in the computer security literatures to detect potential vulnerable problems in software. According to whether the program is running during the process of analysis, vulnerability detection can be divided into static analysis and dynamic analysis.

Static analysis tools have been proven to be particularly effective in specific application fields, such as embedded systems or aviation applications [1]. However, using these techniques on more common software is much harder and time-consuming. There are also some lightweight static analysis tools which can be applied to the context of vulnerability detection [2]. These methods generally focus on a specific kind of vulnerability and the likelihood of false positives is usually very high. Software programmers generally use static analysis techniques for preprocessing information and combine with other tools like concolic execution to complete vulnerability analysis [3]. In our previous work, we also conducted static analysis for Android malware detection [4-7].

Dynamic analysis methods like fuzzing technologies are considered to be the most effective approaches to identify vulnerabilities in large-scale software, which feed the target application with large numbers of various inputs and find abnormal program termination [8]. The crucial step in fuzzing technologies is clearly to generate relevant unexpected inputs, and currently the most common technology is to generate inputs with the help of program code. Hence, many procedures have been developed in this direction and their capacity to ferret out vulnerabilities has been illustrated on several case studies.

However, current analysis tools are limited to detecting well-known vulnerabilities. The general discovery of vulnerabilities still bases on tedious manual auditing, which requires considerable expertise and resources.

With the development of artificial intelligence, some tools like VDiscover [9] that uses advanced machine learning technologies like random forest to find vulnerabilities are proposed. VCCFinder [10] uses a trained SVM classifier to flag suspicious commits with a significantly lower false-positive rate than its comparable systems. These tools can predict which programs contained vulnerabilities with reasonable accuracy and demonstrate the feasibility of using machine learning technologies to vulnerabilities detection.

B. Deep Learning

Deep learning is a popular machine learning method with multi-layer neural networks [11]. As an efficient representation learning method, deep learning has turned out to be very good at discovering useful features in high-dimension raw input without hand-craft feature extraction [12]. In the last few years, researchers have proposed plenty of outstanding deep learning architectures, such as deep convolutional neural networks (CNNs), deep belief networks (DBNs) and deep recurrent neural networks (RNNs).

More recently, deep learning has made plenty of breakthroughs in various fields, like computer vision [13], speech recognition [14] and natural language processing [15], reaching state-of-the-art performances. With AlphaGo making a hit, deep learning becomes well-known and shows a promising approach for other fields, including vulnerability detection [16].

In this paper, we make the following contributions:

978-1-5090-6352-9/17/$31.00 ©2017 IEEE

- We use deep learning methods to predict if a sample is likely to contain a software vulnerability based on dynamic analysis. We collected 9872 sequences of function calls as the features and conduct experiments with the collected data. The experimental results demonstrate that deep learning techniques are effective for vulnerability discovery.
- We propose three deep learning models (CNN, LSTM and CNN-LSTM) for vulnerability detection. These models outperform traditional machine learning models, and the prediction accuracy is improved by around 30%.

The rest of this paper is organized as follows. In Section II we introduce the data and preprocessing method. Section III presents the deep learning models. Section IV gives the experimental procedure and results of different models, as well as a brief discussion. We conclude this paper in Section V.

II. DATA SETS

Before training, we need massive data as dataset during the training phase, which is the fundamental premise of our study. Besides, additional examples are required to measure a trained predictor and data preprocessing is also essential.

A. Data Sets

Our data is created by analyzing 9872 binary programs in "/src/bin" and "/usr/sbin/" directory in a 32-bits Linux machine. The dynamic features are supposed to abstract usage patterns of the C standard library and they are represented as variable-length sequences.

The features are extracted through three steps. First, a binary program is executed in a limited time. Second, the program events are hooked using pythonptrace binding and GNU Binutils. Finally, the events are collected in a sequence. The concrete sequential calls are constituted of the C standard library function calls augmented with its arguments and the final state of the execution which uses the experience of the VDiscover tool [9].

$$fc_i(arg_1, \cdots, arg_n) \mid FinalState \quad (1)$$

The final state will be analyzed to confirm which event will be the last of the execution. In this study, a binary program can finish with an exit, an abnormal termination, an induced abnormal termination or, the execution can run out of time.

The reality that the arguments of function calls are low level computational values, like pointers and integers, becomes a problem for learning patterns in traces. To deal with an enormous range of different arguments, we tag every argument value with a suitable subtype, which uses the experience of the VDiscover tool [9].

In order to acquire the label of every program, at the last step of collecting data, we use zzuf[1], a popular multi-purpose fuzzer to detect crashes, aborts and other interesting unexpected behavior. If zzuf detects a crash, an abort, a timeout or running out of memory, the testcase is flagged as vulnerable and the fuzzing is stopped.

B. Data Preprocessing

Before training the vulnerability predictor, the features of our dataset are preprocessed. Data preprocessing is essential to train and test deep learning models. This procedure should also reduce the dimension of the sequential data, since our learning models require to use fixed length inputs.

In order to process the sequence of function calls, we used tokenizer class in Keras[2] that is often used for text processing and mining. We consider each trace as a text document and use the tokenizer class to reflect words in the text to a vector, which is a list of word indexes, where the word of rank *i* in the dataset has index *i*.

It is obvious that the length of different features in our dataset is unequal and highly variant, which is hard to feed into the deep learning architectures. Therefore, we randomly cut out a vector fragment whose length is between 0 and 25, and fill 0 when the length of the sequence is less than 25. An example of data processing is shown in Figure 1.

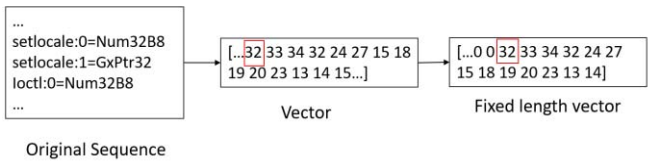

Figure 1. An example of data processing procedure

Finally, we get 9872 binary programs. The dataset is grouped into training set, validation set and test set using a 8:1:1 split.

III. METHODS

Vulnerability detection using machine learning methods has become popular in recent years. The vulnerability detection task is very challenging and motivates us to attempt more efficient approaches. In this paper, we try to use deep learning models to predict the vulnerability. These deep neural networks can learn the relationships between defect, infection and failure. After getting the data, aka the concrete sequential calls to the C standard library, we can refer to the sentiment classification of text where CNNs and LSTM can do well. To gain the best performance for our vulnerability detection task, we have tried a number of different deep learning models, as shown in Figure 2.

A. Convolution Neural Network Model

We use a deep convolutional neural network model at first. Convolutional neural networks are popular in computer vision, and they are the major reasons for breakthroughs in image classification, detection and segmentation. More recently, CNNs are applied to the field of natural language

[1] http://caca.zoy.org/wiki/zzuf

[2] https://keras-cn.readthedocs.io/en/latest/

processing (NLP), and have made some interesting results [17]. Convolutional neural networks excel at learning the spatial structure in input data, and are suitable for classifications tasks, such as sentiment analysis, spam detection or topic categorization, as well as vulnerability detection task.

Building a CNN architecture means there are many hyperparameters to choose, including the numbers and sizes of convolution filters, pooling strategies, activation functions, and so on. The input layer is a sentence comprised of word embeddings, with a dimension of 25. We use the dropout technology after the embedding layer and keep the dropout rate to 0.5, followed by a one-dimension convolutional (Conv1D) layer with 32 filters. The kernel size of our Conv1D is set to 3 and the stride is 1. A global max pooling layer is then used to pick up the maximum value. This is a way to reduce the dimension, while keeping the most salient information. The dense layers are the full connected layers. Each convolutional layer and dense layer are followed by a rectified linear units (ReLU) as the nonlinearity

$$f(x) = max(0, x) \tag{2}$$

where x is the input of ReLU and $f(x)$ is the output. Note that we want a binary classification and the last dense layer is dimension one. In the end, a sigmoid function is applied to make the prediction.

B. *Long Short Term Memory Network Model*

Although CNNs have good performance in many classification tasks, a big problem of CNNs comes from the fixed filter size of vision field. On the one hand, this can't model long sequence information. On the other hand, the parameter of filter size is also very complicated to choose. CNNs are essentially feature expressions of texts, while LSTMs are more commonly used in natural language processing, which can better express contextual information [18]. In text categorization tasks, LSTMs can be interpreted in a way to capture the long-term and bi-directional information.

The first layer of our LSTM model is an embedded layer that uses 10 length vectors to represent each word. The second layer is a LSTM layer with 32 memory units, followed by a dropout layer. The LSTM generally has a problem of overfitting, while dropout can be applied to overcome this. Finally, we use a dense output layer with a single neuron and a sigmoid activation function to make prediction for the two classes in our task.

C. *Convolution Neural Network-Long Short Term Memory Model*

A big argument for CNNs is that they are much faster than LSTMs, and CNNs are also efficient in terms of representation. Convolutional filters can learn good representations automatically, without needing to represent the whole vocabulary. However, CNNs use different window sizes to model the relationship between different scales, leading to lose most of the context information [19]. In this section, we will combine the spatial structure learning properties of CNNs with the sequence learning of LSTMs to our vulnerability detection task. The data has a one-dimensional spatial structure in the sequence of words and the CNNs may be able to pick out invariant features. And then, these spatial features may be learned as sequences by the LSTM layer [20].

In the CNN-LSTM model, the embedding output is not directly access the LSTM layer, but to the CNN layer. This means some sequence is obtained by the CNN, and then these sequences will be access to the LSTM. We can easily add a Conv1D layer and a max pooling layer after the embedding layer, and feeding the consolidated features to the LSTM layer. The hyperparameters of the Conv1D layer and the LSTM layer are the same with the former models. After that, we have 2 dense layers followed and use a sigmoid activation layer to output the result.

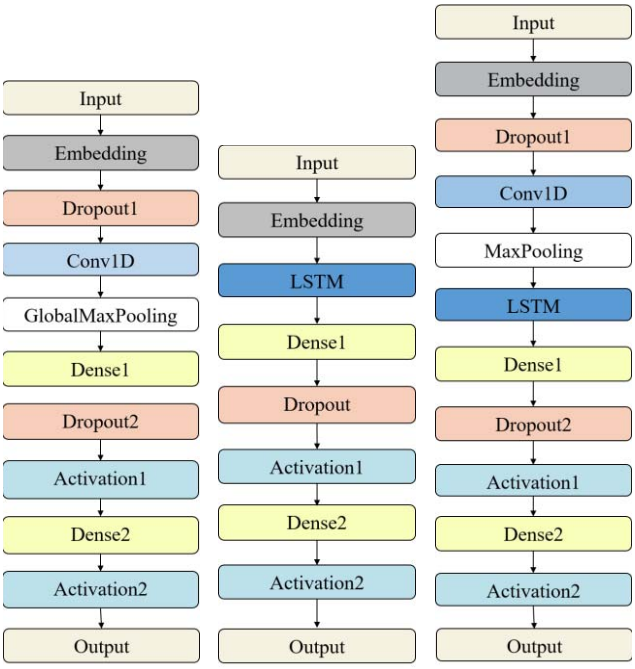

Figure 2. Deep learning models for vulnerability detection, left: CNN model, middle: LSTM model, right: CNN-LSTM model.

IV. EXPERIMENTS

To show the effectiveness of our approach, we conduct experiments to predict vulnerability using the designed models. In this section, we describe the details of the training procedure and make an analysis of the experimental results. In the end, we discuss our method and the future research direction.

A. *Training Procedure*

In the following experiments, we choose Tensorflow and Keras as the deep learning frameworks. Since vulnerability detection is a binary classification problem, log loss is used as the loss function. In addition, we use the efficient Adam optimization algorithm to train our models [21]. In each

training step, we randomly select a mini-batch of 64 samples from the dataset to update the weights. We adopt the glorot_uniform function to initialize weights of the Conv1D layer and the LSTM layer.

The training process is applied on a computer with a 4GHz Intel i5-8250U CPU and 8GB of memory.

TABLE I. DETECTION RESULTS WITH DIFFERENT MODELS FOR VULNERABILITY DETECTION

Model	Training Accuracy	Validation Accuracy	Test Accuracy	Training Loss	Validation Loss	Test Loss
CNN	**0.8725**	0.8250	**0.86**	**0.2985**	0.5177	**0.39**
LSTM	0.8519	0.8313	0.80	0.3501	0.4203	0.53
CNN-LSTM	0.8313	**0.8438**	0.76	0.3431	**0.3898**	0.55
MLP	0.5410	0.5585	0.51	0.6911	0.6845	0.69

B. *Results*

Training accuracy curves of different models are presented in Figure 3. In the first epoch, the accuracy curves of deep learning models increase very fast. In the following, the accuracy curves increase slowly and remain almost unchanged after 2 epochs. In the end, our deep learning methods get an average accuracy of over 80%.

Training loss curves of different models are also presented in Figure 4. We can see that all the loss curves decrease quickly in the first epoch and become slowly after then. Eventually, the loss curves converge and remain stable in the following epochs.

We also compare our deep learning models with multi-layer perceptron (MLP) model, one of the classical machine learning models. The MLP model we designed in this paper has 2 hidden layers. The prediction accuracy of MLP model can reach up to 55.8%, which is about 25% less than the deep learning models, and the loss is about 30% higher. It is obvious that the performance of MLP model is much worse than that of our deep learning models.

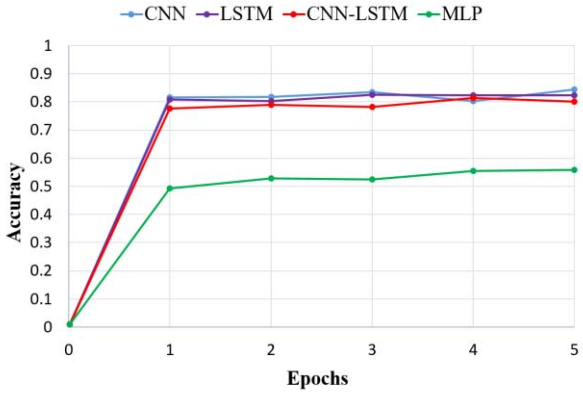

Figure 3. Training accuracy of different models.

To compare the accuracy and loss of different models in training set, validation set and test set, we depict all the results in Table I. We can see that all the three deep learning models have similar performance, superior to the MLP model. In general, our deep learning models are effective and efficient for vulnerability detection task.

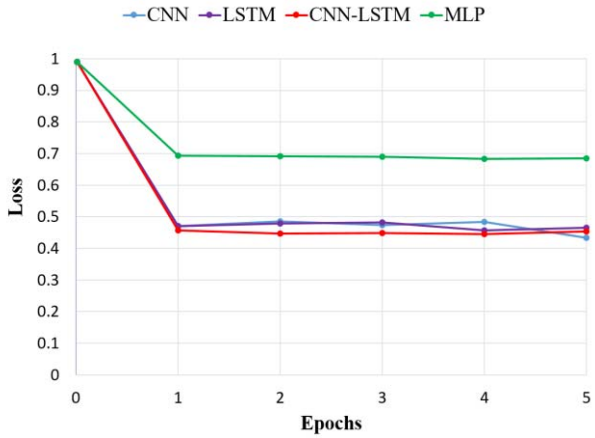

Figure 4. Training loss of different models.

The prediction accuracy of the test data is presented in Table II. The characteristics we use to evaluate the performance of the structures are set as follows: FPR (false positive rate), TPR (true positive rate), ACC (accuracy), F-Score (the harmonic mean of precision and sensitivity: *F-Score=2TP/(2TP+FP+FN)*). We can see that CNN-LSTM model has the highest F-Score.

TABLE II. DETECTION ACCURACY WITH DIFFERENT MODELS

Model	FPR	TPR	ACC	F-Score
CNN	0.220	0.829	0.803	0.794
LSTM	**0.190**	0.860	0.833	0.825
CNN-LSTM	0.211	**0.891**	**0.836**	**0.833**
MLP	0.419	0.547	0.566	0.537

C. *Discussion*

In summary, our deep learning models have been proved to be excellent in vulnerability detection task, reaching an over 80% accuracy. This exceeds the traditional method to a large extent. However, compared with other vulnerability detection methods, our work also has the following limitations:

- The amount of dataset we use is small. Although we have almost ten thousand sequences of function calls

in total, it is still a small amount compared with other deep learning tasks. A large data set may result in better performance with deep learning methods.

- As mentioned in the subsection of data preprocessing, the fixed length of input is 25. We randomly cut out a vector fragment whose length is between 0 and 25, and fill 0 when the length of the sequence is less than 25. For long sequences, 25 is too small to be sufficiently abstracted to execute the program. We will try to use word2vec technique which has successfully used in a variety of text mining applications to overcome this problem [22].
- The performance of deep learning methods is related to network architecture and hyperparameters. It is an empirical task and needs further optimization. In addition, deep learning methods lack interpretability, especially on the analysis of bad cases. We will study the inner mechanism of deep learning models for vulnerability detection.

V. CONCLUSION

In this work, we have applied deep learning methods for vulnerability detection. We extract dynamic features from 9872 binary programs and train different deep learning models to detect vulnerability. The experimental results show that the prediction accuracy of deep learning models reaches 83.6%, which is much higher than that of traditional method like MLP. In the future, we will collect more high-quality data and explore more efficient deep learning models to improve the accuracy of vulnerability detection.

ACKNOWLEDGMENT

We thank Kun Shao in Institute of Automation of CAS for his helpful suggestions. The work reported in this paper was supported in part by Science and Technology on Electronic Information Control Laboratory, under Grant K16GY00040, in part by Shanghai Key Laboratory of Integrated Administration Technologies for Information Security, under Grant AGK2015002, in part by ZTE Corporation Foundation, under Grant K17L00190, in part by the Fundamental Research funds for the central Universities of China, under grant K17JB00060 and K17JB00020, in part by National Key R&D Program of China, under grant 2017YFB0802805, and in part by Natural Science Foundation of China, under Grant 61672092.

REFERENCES

[1] F. Kirchner, N. Kosmatov, V. Prevosto, et al. “Frama-C: A software analysis perspective,” Formal Aspects of Computing, vol .27, May. 2015, pp. 573-609.

[2] D. Evans and D. Larochelle, “Improving security using extensible lightweight static analysis,” IEEE Software, 2002.

[3] S. Cha, T. Avgerinos, A. Rebert, et al. “Unleashing mayhem on binary code,” Proceedings of the 2012 IEEE Symposium on Security and Privacy, May 20-25, 2012, pp. 380-394.

[4] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, X. Zhang, “Exploring Permission-induced Risk in Android Applications for Malicious Application Detection,” IEEE Transactions on Information Forensics and Security (TIFS), vol. 9, no. 11, 2014, pp.1869-1882.

[5] W. Wang, Y. Li, X. Wang, J. Liu, X. Zhang, “Detecting Android Malicious Apps and Categorizing Benign Apps with Ensemble of Classifiers,” Future Generation Computer Systems, vol.78, 2018, pp. 987-994.

[6] X. Liu, J. Liu, W. Wang, Y. He, X. Zhang, “Discovering and Understanding Android Sensor Usage Behaviors with Data Flow Analysis,” World Wide Web (to appear), 2017.

[7] X. Wang, W. Wang, Y. He, J. Liu, Z. Han, X. Zhang, “Characterizing Android Apps’ Behavior for Effective Detection of Malapps at Large Scale,” Future Generation Computer Systems, vol. 75, 2017, pp. 30-45.

[8] M. Sutton, A. Greene, and P. Amini, “Fuzzing: Brute Force Vulnerability Discovery,” Addison-Wesley Professional, 2007.

[9] G. Grieco, G. L. Grinblat, L. Uzal, et al. “Toward Large-Scale Vulnerability Discovery using Machine Learning,” Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, New Orleans, Louisiana, USA, Mar 09-11, 2016, pp. 85-96.

[10] H. Perl, S. Dechand, M. Smith, et al. “VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits” Acm Sigsac Conference on Computer & Communications Security, 2015, pp. 426-437.

[11] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” Nature, vol. 521, no. 7553, 2015, pp. 436-444.

[12] J. Schmidhuber, “Deep learning in neural networks: An overview,” Neural Netw., vol. 61, 2015, pp. 85-117.

[13] A. Krizhevsky, I. Sutskever, and G. E.Hinton, “Imagenet classification with deep convolutional neural networks,” in Proceeding of the Neural Information and Processing Systems, 2012, pp. 1097-1105.

[14] G. Dahl, D. Yu, L. Deng, and A. Acero, “Context-dependent pretrained deep neural networks for large-vocabulary speech recognition,” IEEE Trans. Audio Speech Lang. Processing, vol. 20, no. 1, Jan. 2012, pp. 30-42.

[15] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to Sequence Learning with Neural Networks,” Proceeding of the Neural Information and Processing Systems, 2014.

[16] D. Silver, A. Huang, C. Maddison, et al. “Mastering the game of Go with deep neural networks and tree search.” Nature, 2016, pp. 484-489.

[17] Y. Kim. “Convolutional neural networks for sentence classification,” Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1746-1751.

[18] P. Liu, X. Qiu, X. Huang. “Recurrent neural network for text classification with multi-task learning,” International Joint Conference on Artificial Intelligence, 2016, pp. 2873-2879.

[19] S. Lai, L. Xu, K. Liu, et al. “Recurrent Convolutional Neural Networks for Text Classification,” Association for the Advancement of Artificial Intelligence(AAAI), 2015, pp. 2267-2273.

[20] C. Zhou, C. Sun, Z. Liu, et al. “A C-LSTM neural network for text classification,” arXiv preprint arXiv:1511.08630, 2015.

[21] D. Kingma, J. Ba, “Adam: A method for stochastic optimization,” Proceedings of the International Conference on Learning Representations, 2015.

[22] Y. Goldberg, O. Levy, “word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method,” arXiv preprint arXiv:1402.3722, 2014