Title: Automated Vulnerability Detection in Source Code Using Deep Neural Networks

GR-284

Mst Shapna Akter (CS PhD Candidate), Research Mentors: Dr. Dan Lo

Abstract

One of the most important challenges in the field of a software code audit is the presence of vulnerabilities in software source code. Every year, more and more software flaws are found, either internally in proprietary code or revealed publicly. These flaws are highly likely exploited and lead to system compromise, data leakage, or denial of service. C and C++ open-source code are now available in order to create a large-scale, machine-learning system for function-level vulnerability identification. We assembled a sizable dataset of millions of open-source functions that point to potential exploits. We created an efficient and scalable vulnerability detection method based on deep neural network models that learn features extracted from the source codes. To remove the pointless components and shorten the dependency, the source code is first converted into a minimal intermediate representation. We keep the semantic and syntactic information using state-of-the-art word embedding algorithms. The embedded vectors are subsequently fed into convolutional neural networks to classify the possible vulnerabilities. Furthermore, we proposed a new neural network model which seems to overcome issues associated with traditional neural networks. To measure the performance, we used evaluation metrics such as f1 score, precision, recall, accuracy, and total execution time.

Introduction & Motivation

Recent well-publicized exploits have demonstrated that these security flaws can have catastrophic impacts on society and the economy in our healthcare, energy, defense other critical infrastructure systems [2]. For instance, the ransomware Wannacry swept the globe by using a flaw in the Windows server message block protocol [3]. According to the Microsoft Security Response Center, half of 2015 had an industry-wide surge in high-severity vulnerabilities of 41.7%. This represents 41.8% of the total greatest proportion of software vulnerabilities in at least three years [4]. Furthermore, according to a Frost and Sullivan (a global growth consulting company) analysis released in 2018, there was an increase in severe and high severity vulnera-bilities, going from 693 in 2016 to 929 in 2017, with Google Project Zero coming in second place in terms of disclosing such flaws. On August 14, 2019, Intel issued a warning on a high-severity vulnerability in software it uses to identify the specifications of Intel processors in Windows PCs [5] . The paper claims that these defects, including information leaking and denial of service assaults, might substantially affect software systems. Although the company issued an update to remedy the problems, an attacker can still use these vulnerabilities to escalate their privileges on a machine that has already been compromised. We have come up with a solution for detecting software vulnerabilities using deep neural networks such as Simple RNN, LSTM, BilSTM, BERT, GPT2, and LSTM-Autoencoder. Moreover, we developed a neural network that worked best among all the models. All of the models have been evaluated using evaluation metrics such as f1 score, precision, recall, accuracy, and total execution time

Related Work

- Previously, Zeng et al. [6] reviewed software vulnerability analysis and discovery using deep learning techniques. They found four game changers who contributed most to the software vulnerability using deep learn-ing techniques.
- Yamaguchi et al. [7] put forth an anomaly detection technique for taint-style vulnerabilities. It groups the variables that pass on to functions with sensitive security. Then, the violation is reported as a potential vulnerability by anomaly detection. This strategy works well with taint-style vulnerability but not with all vulnerabilities.
- Wang et al. [8] proposed an automatic semantic learning process using deep learning models for defect detection. They used DBN, a generative graphical model, capable of learning representation that can reconstruct training data with a high probabilities.
- Kim et al. [9] proposed a technique for identifying vulnerabilities based on similarity. Although, this approach is only effective against vulnerabilities brought on by code cloning

Method

We chose to analyze software packages at the function-level because it is the lowest granularity level that captures a subroutine's overall flow. We compileda vast dataset of millions of function level examples of C and C++ code from the SATE IV Juliet Test Suite, Debian Linux distribution, and public Git repositories on GitHub described in the paper of Russel [36]. In this project we have used CWE-119 vulnerability feature. This feature basically indicates issues associated with buffer overflow vulnerability. A buffer overflow occurs when data is written to the buffer that is longer than the buffer itself, overwriting storage units outside the buffer in the process We first transformed the samples of source code into the minimum intermediate representations through dependency analysis, program slicing, tokenization, and serialization. Later, we extracted semantic features using word embedding algorithms such as GloVe and fastText. After finishing the data preprocessing stage, we fed the input representation to the deep neural networks for classification.

GloVe: is a language algorithm for prevailing vector representations of words. This is an unsupervised learning algorithm; the training process has been performed on global word-word co-occurrence statistics from a corpus.

FasText: is an embedding method that uses a word's deep-down structure to improve the vector representations acquired from the skip-gram method.

Classification Models: The vector representation of the software source code has been fed to the LSTM, BiLSTM, LSTM-autoencoder, Word2vec, BERT, and GPT2 models. The dataset has been divided into training and validation portions for the purpose of training the models. Finally, the test dataset is used to evaluate each trained model.

```
1  void bar(char *buf, char *src) {
2      strcpy(buf, src);
3  }
4  int main() {
5      char buf[10];
6      char src[10];
7      memset(src, 'A', 10);
8      src[10 - 1] = '\0';
9      bar(buf, src);
10     for (int i = 0; i <= 10; i++)
11     //writes buf [10] and overruns memory
12         buf[i] = 'B';
13     return 0;
14 }
```

Figure 1: An intra-procedural buffer overflow vulnerability

Proposed Architecture

- **Proposed model:** We created a model using an autoencoder and LSTM. The network aims to close the distance between the reconstructed representation and the original input as it learns. We stacked the LSTM and autoencoder layers. The term "stacked LSTM network" refers to an LSTM network that has many stacked LSTM layers. The network becomes more complex and deeper when LSTM layers are stacked. In a stacked LSTM encoder part, an LSTM layer below offers several outputs to its above LSTM layer as opposed to a single output. In other words, it generates one output for each input time step as opposed to one output for all input phases.

Results & Further Work

Models	Accuracy	precision	Recall	F1 Score	Execution Time
Simple RNN	0.90	0.94	0.93	0.95	1h 1min
LSTM	0.91	0.91	0.93	0.92	40min 2s
BiLSTM	0.91	0.93	0.93	0.99	2h 38min
GRU	0.90	0.95	0.96	0.98	1h 55min
Word2vec	0.92	0.92	0.97	0.99	42min 56s
BERT	0.94	0.98	0.98	0.99	5h 16min
gpt2	0.95	0.97	0.98	0.97	8h 33min
LSTMAutoencoder	0.92	0.98	0.94	0.94	59min 13s
Proposed Model	0.95	0.99	0.98	0.99	2h 46min

Table : displays a portion of Vulnerable Source code Classification results using different Neural network models with no word embedding algorithms

Considering the experimental results, the proposed model gives highest accuracy, f1-score, recall. GPT-2 model provides almost similar result as our proposed model but our model is more efficient in terms of execution time.,

Conclusion

The advancement of machine learning technology incorporates new approaches to address the limitations of conventional approaches. One of the key research directions is to develop intelligent source code-based vulnerability detection systems. We developed a powerful and adaptable vulnerability detection technique based on deep neural network models that take in information from source code attributes. The source code is initially transformed into a minimum intermediate representation in order to eliminate the unnecessary components and reduce the dependency. Using cutting-edge word embedding methods, we maintain the semantic and syntactic information. Convolutional neural networks are then fed the embedded vectors to categorize the potential vulnerabilities. Additionally, we put forth a novel neural network model that appears to resolve problems with conventional neural networks. Evaluation measures such the f1 score, precision, recall, accuracy, and total execution time were utilized to gauge performance. We found that our proposed model provides better accuracy and more efficient in terms of execution time.

References

1] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: a study of developer work habits," in Proceedings of the 28th international conference on Software engineering, pp. 492–501, 2006.
[2] D. Yadron, "After heartbleed bug, a race to plug internet hole," Wall Street Journal, vol. 9, 2014.
[3] T. Manikandan, B. Balamurugan, C. Senthilkumar, R. R. A. Harinarayan, and R. R. Subramanian, "Cyberwar is coming," Cyber Security in Parallel and Distributed Computing: Concepts, Techniques, Applications and Case Studies, pp. 79–89, 2019.
[4] A. Arora and R. Telang, "Economics of software vulnerability disclosure," IEEE security & privacy, vol. 3, no. 1, pp. 20–25, 2005.
[5] K. Jochem, "It security matters,"
[6] P. Zeng, G. Lin, L. Pan, Y. Tai, and J. Zhang, "Software vulnerability analysis and discovery using deep learning techniques: A survey," IEEE Access, vol. 8, pp. 197158–197172, 2020.
[7] F. Yamaguchi, A. Maier, H. Gascon, and K. Rieck, "Automatic inference of search patterns for taint-style vulnerabilities," in 2015 IEEE Symposium on Security and Privacy, pp. 797–812, IEEE, 2015.
[8] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pp. 297–308, IEEE, 2016.
[9] S. Kim, S. Woo, H. Lee, and H. Oh, "Vuddy: A scalable approach for vulnerable code clone discovery," in 2017 IEEE Symposium on Security and Privacy (SP), pp. 595–614, IEEE, 2017

KENNESAW STATE UNIVERSITY