Research Proposal: Machine-learning Guided Software Security analysis

Wenke Yang

Apr. 2020

1 Background

1.1 Motivation

Computer software has now penetrated every aspect of our lives. It plays a vital role in key industries, such as healthcare, banking, aviation, transportation, and e-commerce. Therefore, it is crucial for us to develop secure software that operates correctly without unexpected behaviours. According to the white paper published by Tricentis[2], software failures impacted half of the world's population (3.7 billion people) and caused more than $1.7 trillion assets loss in 2018. In the last few decades, researchers have devoted a large number of efforts in the area of software security and a number of software security analysis tools were developed to improve the quality of software.

1.2 Existing efforts

The traditional software security analysis can be classified into two types: static code analysis and dynamic testing. Static code analysis is a preventive action performed on source code before execution trying to detect possible vulnerabilities and errors. While dynamic testing inserts additional code to the program to capture errors at runtime. The advantages of static code analysis over dynamic testing are: (1) static analysis can find bugs during early stages; (2) static analysis is also known as white-box testing which can spot the segments of code where the errors occurred to help programmer fix bugs. However, static analysis is usually time-consuming and can easily run out of memory because traditional methods build an abstract syntax tree(AST) and traversing through the tree to do flow tracing(data flow analysis) or pattern matching, which are computationally intensive. It can take days to analyse a medium-sized project and is often impossible to scale to large projects in practice. Besides, the accuracy of static analysis is also a big concern, because only limited patterns and security issues can be detected through the traversal. Last but not least, static analysis needs lots of human experts' interactions to define different security patterns.

Recently there emerges a different approach to do static code analysis which is using machine learning to overcome the above-mentioned shortcomings. Since source code is indeed a collection of texts, we can transfer deep learning techniques developed in natural language processing(NLP) to analyse the code. One feature of program analysis is same as NLP problems that is the length of the text is unknown. Recurrent Neural Network (RNN) can recursively take input which fits this feature well. However, one of the problems is that NLP techniques, for instance, recurrent neural networks(RNNs), are designed to process human languages which do not consider the hierarchical information(AST) of source code. Hence, significant modification and/or re-design work needs to be done to apply such deep learning techniques in static code analysis.

Secondly, as source code can be represented as a tree, or more generally a graph, it is promising to efficiently and effectively do static code analysis using graph neural networks(GNNs)[7]. However, current GNN based methods are unable to report the locations of errors and can only identify some trivial issues. Therefore, my research will focus on improving existing methods to provide more efficient and accurate analysis.

Furthermore, different machine learning approaches have been applied to other software analysis fields, such as source code summarisation[9] and source code retrieval[8]. For security analysis, the similar approaches such as attention network learning and reinforcement learning can also be adopted with modifications.

2 Project Aims

The general objective of this proposed project is to develop a static analysis tool based on machine learning approaches to improve the accuracy and efficiency of software vulnerabilities detection. We aim to design novel neural network models(e.g. RNNs, GNNs) and machine learning techniques to overcome the limitations of traditional methods. My approach will mainly focus on bridging the gap between the traditional static analysis and burgeoning machine learning field. Given a segment of source code, the proposed machine learning-based static analysis tool should be able to identify the likelihood and locate the occurrences of major security issues in this segment.

3 Methodology

For this project, I will mainly apply machine learning techniques and improve existing models from different aspects. Machine learning techniques include graph neural networks and natural language processing models. Furthermore, different possible vulnerabilities for detection and choices of programming languages are introduced. In addition, the solution to the lack of training datasets for machine learning approaches is proposed.

3.1 Machine Learning Techniques

3.1.1 Graph Neural Networks(GNNs)

There is a number of work proposing machine learning techniques to analyse software security issues. Among all of the most recent approaches, for detecting vulnerabilities - variable misuse and variable naming, GNN is proved that it outperforms the current baselines BiRNN and LBL around 20%[5]. While GNN related approaches have not been widely applied in more complex problems in software security area, it shows a great potential of application of GNN in software security analysis.

With respect to the current work of GNN, there has been some research addressing graph characteristics of AST and control flow graph generated from source code with program security analysis. For example, the VGDetector, proposed by Cheng et al. in 2019[3], they applied graph embedding technique to transfer nodes and edges in a control flow graph to vectors. However, transforming into vectors still lead to a loss in hierarchical information which can lead to imprecise propagation of information between nodes. Furthermore, another fatal weakness of graph embedding is that locations of occurrences of security issues are lost once transformed to vector form. In this project, instead of converting the graphs into embeddings, I will design novel methods which apply appropriate deep learning technique, for example, Gated-GNN, to keep the original structural information of the source code.

Gated-GNN is a variance of GNN with the same basic idea as RNN, proposed by Li et al. in 2016[10]. RNN is the most commonly used deep learning approach in natural language processing because it takes sequence information into account. The main idea of Gated-GNN is not only considering the sequence relationship but all neighbours connected to a specific node. In addition, different neural networks are constructed for different types of edges. This results in better-structured relationships between nodes and edges because different types of neighbours for a node contribute accordingly. Hence, my model should be able to propagate information between nodes more precise by taking different types of edges and relationships into consideration. In addition, calculations of my model should all be done on the graph, thus would be able to return locations of security issues by addressing the nodes with top possibilities of occurrence of errors. Moreover, by applying regression, the output can give information about possibilities rather than if a specific security issue exists or not.

For this project, I will provide a new end-to-end solution to users that can detect various software security vulnerabilities with gated-GNN. The structure of gated-GNN and how the program should be fitted into the neural network need to be decided. Using this approach, the report of error locations could be possible.

3.1.2 Natural Language Processing(NLP)

As to the combination of NLP and static code analysis, existing approaches mainly consider source code only and discard the comments. While taking mismatch between the developer's implicit specification and implementation into consideration can also help spot the vulnerabilities of software better.

3.2 Vulnerabilities

In regard to vulnerabilities, the research above mentioned only detect control flow related vulnerabilities, while we can apply graph embedding to do detect security issues that require pattern matching, such as sensitive data exposure and using components with known vulnerabilities[1]. Whereas most other existing research only detects trivial errors(e.g. variable misuse[5], binary expressions[6]). In this project, I can work on more diverse and complex vulnerabilities, for instance, bugs not fixed by replacing a variable name but require some additions or deletions of statements.

3.3 Programming Languages

There is a various number of programming languages designed for different purposes. The languages can be divided into interpreted languages and compiled languages. Different languages have different characteristics which have different possible vulnerabilities, for example, memory corruptions and use after free pointer related issues in C/C++. Currently, researches mainly focus on Javascript[4] among interpreted languages and C/C++ among compiled languages. Since C/C++ is one of the most widely used programming languages, and GNN related approaches have not been addressed to most security issues in C/C++, this project would mainly focus on detecting security issues in C/C++.

3.4 Dataset

Considering the machine learning approaches requires a large number of learning datasets while there only exist limited sample data for some particular security issues, this proposed project will also apply transfer learning-based method [3]. Transfer learning is a method that applies data collected for solving one problem to another for those tightly-knitted problems. Taking Use After Free problem in programming language C/C++ as an example, the UAF datasets of other programming languages, such as C#, Java, can also be applied because the object dispose exception in C# is similar to UAF in C/C++. After data collection and feature extraction, it can use a transfer-learning model which is trained by some other languages to detect UAF in C/C++. Apart from transfer learning, reinforcement learning is also a possible approach to solve the problem of lack of datasets. The key issue is to define effective reward mechanisms.

4 Expected Outcome

The expected outcomes of the project are implementing an open-source tool for automatically detecting software vulnerabilities based on machine learning and producing high-quality publications in the areas of software engineering and artificial intelligence.

5 Timeline

Semester	Aims
S1/2021	1. Review and formalise the existing work in detecting software vulnerabilities. 2. Complete preliminary courses in university. 3. Explore current machine learning approaches applications.
S2/2021	1. Summarise the advantages and disadvantages of existing methods and conduct an experimental study. 2. Submit a survey/experiment paper to a major conference/journal. 3. Based on the limitations of existing approaches, prototyping the new machine learning models for software security analysis.
S1/2022	1. Gathering training and testing dataset from Github, Stackoverflow, or other sources. 2. Conducting preliminary experiments about the proposed prototype model. 3. Improve the model using the findings in the preliminary experiment results.
S2/2022	1. Finalise all insights and findings. 2. Writing an academic paper about proposed static analysis model. 3. Submit a paper to a major conference/journal.
S1/2023	1. Further improve the accuracy of the model. 2. Try to integrated the model with distributed environment to support large-scale software development. 3. Explore and support more programming languages.
S2/2023	1. Submit an extension paper to a major conference/journal. 2. Open source the proposed tool
S1/2024	1. Investigate other relevant directions and do some further work. 2. Writing the PhD thesis

Table 1: Timeline from S1/2021 to S1/2024

References

[1] Top 25 software errors: Sans institute. https://www.sans.org/top25-software-errors/.

[2] software fail watch: 5th edition. https://www.tricentis.com/resources/software-fail-watch-5th-edition/, 2018.

[3] Xiao Cheng, Haoyu Wang, Jiayi Hua, Miao Zhang, Guoai Xu, Li Yi, and Yulei Sui. Static detection of control-flow-related vulnerabilities using graph embedding. *24th International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2019.

[4] Elizabeth Dinella, Hanjun Dai, Ziyang Li, Mayur Naik, Le Song, and Ke Wang. Hoppity: Learning graph transformations to detect and fix bugs in programs, Mar 2020.

[5] Mahmoud Khademi Miltiadis Allamanis, Marc Brockschmidt. Learning to represent programs with graphs. *International Conference on Learning Representations (ICLR)*, 2018.

[6] Michael Pradel and Koushik Sen. Deepbugs: a learning approach to name-based bug detection. *Proceedings of the ACM on Programming Languages*, 2(OOPSLA):1–25, 2018.

[7] F. Scarselli, Sweah Liang Yong, M. Gori, M. Hagenbuchner, Ah Chung Tsoi, and M. Maggini. Graph neural networks for ranking web pages. *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI05)*.

[8] Yao Wan, Jingdong Shu, Yulei Sui, Guandong Xu, Zhou Zhao, Jian Wu, and Philip Yu. Multi-modal attention network learning for semantic source code retrieval. *34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019.

[9] Wenhua Wang, Yuqun Zhang, Yulei Sui, Yao Wan, Zhou Zhao, Jian Wu, Philip Yu, and Guandong Xu. Reinforcement-learning-guided source code summarization via hierarchical attention. *IEEE Transactions on Software Engineering*, 2020.

[10] Marc Brockschmidt Richard Zemel Yujia Li, Daniel Tarlow. Gated graph sequence neural networks. *International Conference on Learning Representations (ICLR)*, 2016.