Machine-Learning-Guided Program Analysis For Detecting Software Vulnerabilities

Tengjiao Zhu

1. Background

Software is often subject to external attacks that aim to control its behaviour. The majority of the attacks rely on some form of control hijacking to redirect program execution caused by memory errors. For instance, a buffer overflow in an application may result in a call to a sensitive system function, possibly a function that the application was never intended to use. Use-after-free (UAF) vulnerabilities, i.e., dangling pointer dereferences (accessing objects that have already been freed) in programs can also cause data corruption, information leaks, denial-of-service attacks (via program crashes), and control-flow hijacking attacks [5].

Attacks exploiting software vulnerabilities can cause significant social and economic impact. On one hand, government and industry have paid great amount of effort to prevent or mitigate the damage caused by vulnerabilities, on the other hand, the vulnerability assessment market in 2017 is predicted to generate close to 1.5 billion U.S. dollars in revenue according to statista.com, while it was 672.9 million in 2009.

2. Problem statement

Although the extensive efforts [1-3] have been made to develop precise and efficient software analysis techniques for detecting vulnerabilities. The state-of-the-art techniques are still insufficient for analysing large-scale industrial-sized software.

In the big data era, the increasing demand on data-driven business has lead to data proliferation and also significantly advanced data analytics and machine-learning research. However, the existing program analysis techniques relying on conventional analysis theory (e.g., data-flow, abstract interpretation), ignore a large amount of data generated during the software development (e.g., source code, test programs, execution traces, code commits history). Thus, they do not benefit from the state-of-the-art data analytics.

3. Aims and objectives

From program analysis point of view, data refers not only program source or binary code, but also huge amount of testing inputs, program specifications, execution traces, defect code patterns, and existing programming experiences.

This project aims to systematically investigate how to apply learning-based program analysis in detecting software bugs by leveraging the massive amount of available data. My plan is to design and implement a new method to accurately locate potential software vulnerabilities in C++ and/or Java programs based on machine learning and deep learning techniques, such as CNN, RNN and LSTM, which could detect security vulnerabilities, such as memory leaks, UAFs and race conditions.

4. Research Methodology

My previous research focus on how to systematically apply machine and deep learning-based algorithms in reality by building neural network including Image Creation with GAN, Style Transfer with CNN, Articles Creation with LSTM, Image Caption with LSTM and ResNet, which can provide new methods for detecting both critical security related vulnerabilities and normal potential defects such as memory leak, UAF and race condition.

In order to predict the vulnerability, the project designs and implements a new tool to make full use of machine leaning like Support Vector Machine(SVM) Or Logistic Regression (LR) and ADAboost and deep learning like DNN to determine the likelihood of a use-after-free bug or within C source files. It will automatically detect critical security bugs such as use of previously freed memory, for C/C++ source files. The steps of the algorithms includes two aspects.

First, the method will adapt a static analysis method [4] to avoid complexity caused by dynamic approach. It classifies different defects via a neural network including RNN, LSTM which have excellent ability to distinguish different features of errors.

Second, it will resolve some kinds of defects - for example, memory leak - automatically based on ResNet or VGG that deep the layers of neural network and could take the richer meaning of defects. The performance and accuracy of the project will be addressed against some open source libraries, and the testing result will be compared with existing approaches.The implementation of the project might be based on RandomForest, XGboost.

5. Our pilot studies and existing open-source tools

I will work on the proposed topic by following our existing works in developing program analysis tools [1] and our pilot studies in applying machine-learning techniques to detect software vulnerabilities, e.g., use-after-free bugs [2] . The existing open-source tool SVF [6] (developed by Dr. Sui) and its datasets can be a good base for my forthcoming research project. SVF is publicly available and hosted on Github. It is a fundamental program analysis infrastructure for program understanding and software bug detection. The tool and its related publications has been used and cited by many developers and researchers world-wide.

6. Resources required

Nothing extensive is required aside from access to powerful computing re- sources to run complicated static analysis for analysing large-scale programs .

7. Research plan and timetable

Months 0 - 6	1.Formalising existing work, expanding literature review. 2.Conduct continuous, through literature review to identify gaps in knowledge and experts 3.Identify specific aims of project based on research vision,plan, data results literature review results
Months 7 - 12	1.Design and implement program analysis based on existing open-source tools, 2.Investigate the existing implementation and how they relate to my specific project
Months 13 - 24	1.Obtain advice/guidance from colleagues and sponsor sources(e.g., research supervisors) 2.Know potential reviewers(my audience) 3.Compiler implementation and performance improvements (in both design and implementation). 4. Start writing research papers
Months 25 - 30	1.Write research papers 2.Put research paper draft aside for a time,then edit 3.Ask external reviewers to review draft and provide comments

Months 31 - 36	1.Try to put everything together and investigating other directions which may appear. 2.Rewrite research paper based on external reviewers comments(this process may continue until close to proposal submission) 3.Submit Ph.d paper to UTS graduate school and review

8. Expected outcomes

The expected outcomes of the project are 1) an open-source tool for automatically detecting different types of software defects of the C++ and/or Java source files based on neural networks; 2) high-quality publications in the area of software engineering, security and artificial intelligence.

References

1. Hua Yan, Yulei Sui, Shiping Chen and Jingling Xue. Spatio-Temporal Context Reduction: A Pointer-Analysis-Based Static Approach for Detecting Use-After-Free Vulnerabilities. 40th International Conference on Software Engineering
2. Hua Yan, Yulei Sui, Shiping Chen and Jingling Xue. Machine-Learning-Guided Typestate Analysis for Use-After-Free Detection, 33th Annual Computer Security Applications Conference
3. Yulei Sui, Ding Ye, Yu Su and Jingling Xue. Eliminating Redundant Bounds Checks in Dynamic Buffer Overflow Detection Using Weakest Preconditions , IEEE Transactions on Reliability
4. Yulei Sui and Jingling Xue. On-Demand Strong Update Analysis via Value-Flow Refinement , [slide] ACM SIGSOFT International Symposium on the Foundation of Software Engineering.
5. Xiaokang Fan, Yulei Sui and Jingling Xue. Boosting the Precision of Virtual Call Integrity Protection with Partial Pointer Analysis for C++, International Symposium on Software Testing and Analysis (Artifact)
6. Yulei Sui and Jingling Xue. SVF: Interprocedural Static Value-Flow Analysis in LLVM , 25th International Conference on Compiler Construction