RESEARCH PROPOSAL

Applicant: Yuxiang Lei

Supervisor: Dr. Yulei Sui

Title: Software Bug Detection via Deep Learning

PROJECT DESCRIPTION

Introduction

Program Analysis is widely recognized as a fundamental tool for program verification, bug detection, compiler optimization, program understanding, and software maintenance. Nowadays, with the massive growth of software, analyzing large-scale programs is inevitable. It has become vital yet enormously difficult to develop effective techniques to significantly reduce the analysis overhead while maintaining sufficient precision for analyzing large code bases in practice. However, because of the undecidability of program analysis, it is hard to make the analysis approach fully algorithmic; and checking methods usually fail to detect certain bug or inconsistency from codes when facing complex logic of software [1].

Software vulnerability checking is a kind of model checking, and it mainly focuses on detect the implicit vulnerability of codes which may cause the partial error or even a systemic breakdown, mainly because of the memory error, such as the Use-after-free (UAF) vulnerabilities. UAF Use-after-free (UAF) vulnerabilities, i.e., dangling pointer dereferences can cause data corruption [2], information leaks [3], denial-of-service attacks (via program crashes) [4], and controlflow hijacking attacks [5]. UAF recently attracts lots of researchers' study and it has made considerable progress.

Literature Review

In recent year, a number of methods were proposed to detect UAF vulnerabilities. Those methods can be generally categorized into two groups: dynamic analysis and static analysis. Dynamic analysis [6,7,8], although has high precise rates, usually suffer

long runtime drawback and incompatibility issues due to code instrumentation used. Static analysis [9] does not suffer these problems, but it require precise pointer analysis to reduce false alarm and avoid too low precise rates. Typestate analysis [10] is a widely used program analysis method. It focuses on examining the state of variables and can be used to determine temporal memory safety vulnerabilities. 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 deep learning research.

Research Objectives

My research will mainly focus on:

1) Combination of traditional static analysis method with machine learning methods;

2) Application of artificial neural network into software vulnerability analysis. Study the combination of static analysis and artificial neural network;

3) Application of deep learning model in software vulnerability checking, study the feasibility or performance of using deep learning solely to handle the problem.

Methodology

The following points can effectively facilitate my research:

1) Machine learning [11] learns a subject by training a specific model using statistical learning methods via formalized data (usually by digitizing features of certain subjects), hence realize the abilities of classification, regression and prediction. Since the features of software vulnerabilities is categorizable and formalizable, machine learning can be applied in software vulnerability checking. [12] proposed a novel SVM guided typestate analysis method to detect UAF vulnerability and obtained much better performance than traditional typestate analysis approaches.

2) Similarly artificial neural network (ANN) can be used in software vulnerability checking. Moreover, it has the advantages of strong capacity, computational power and robustness over machine learning methods. It also has the characteristic of non-sensitivity to the form of certain problem [13,14,15].

3) Since codes are string based data, the natural language processing [16,17] techniques can play their roles in software vulnerability detection. The CNN+RNN based model can effectively excavate the semantic feature of natural language sentences. Analogously, these models [18,19,20] can be tune for software vulnerability checking.

Since I have done researches on machine learning (e.g., ICA, knn, SVM, adaBoost, etc.) and artificial neural network (e.g., CNN, RNN, etc. and Tensorflow, Pytorch, etc.) and have the experiences of applying them in real-world problems. The algorithms themselves can be applied in software bug detection. Moreover, sometimes I have had to spend lots of time in debugging. So I think the codes I have written can always be as sources/objects for software bug detection.

My research will start from study of the features of software vulnerabilities, by analyzing the control-flow and data-flow information of a program to extract useful program features. Then I will study the usage of ANN to improve the precision and efficiency of program analysis (e.g., typestate analysis). I will also try to discover the DNN-only model as an alternative of typestate analysis.

I will work on the proposed topic by following our existing works in developing program analysis tools [21] and our pilot studies in applying machine-learning techniques to detect software vulnerabilities, e.g., use-after-free bugs [22]. The existing open-source tool SVF [23] (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.

Schedule

Month 0 - 6	Preparation: literature survey; Data collection and comparative study;
Month 7 - 12	Specify my research target and plan; Collect methods that could aid my specific project;
Month 13 - 24	Propose idea and design the corresponding method; Validate the proposed method;
Month 25 - 31	Write research papers while continue complete my proposed method; Update my research progress into the research paper; Ask external reviewers to review the paper;
Month 32 - 36	Summarize my research; Write my PhD thesis; Submit the PhD thesis to UTS graduate school and review.

Main References

1 I. Grobelna, M. Grobelny, M. Adamski, "Model Checking of UML Activity Diagrams in Logic Controllers Design", Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX, Advances in Intelligent Systems and Computing Volume 286, Springer International Publishing Switzerland, pp. 233-242, 2014

2 Mauro Conti, Stephen Crane, Lucas Davi, Michael Franz, Per Larsen, Marco Negro, Christopher Liebchen, Mohaned Qunaibit, and Ahmad-Reza Sadeghi. 2015. Losing control: on the effectiveness of control-flow integrity under stack attacks. In CCS'15. 952–963.

3 Sangho Lee, Changhee Jung, and Santosh Pande. 2014. Detecting memory leaks through introspective dynamic behavior modelling using machine learning. In ICSE'14. 814–824.

4 Haogang Chen, Yandong Mao, Xi Wang, Dong Zhou, Nickolai Zeldovich, and M Frans Kaashoek. 2011. Linux kernel vulnerabilities: State-of-the-art defenses and open problems. In APSYS'11. Article No.5.

5 Haogang Chen, Yandong Mao, Xi Wang, Dong Zhou, Nickolai Zeldovich, and M Frans Kaashoek. 2011. Linux kernel vulnerabilities: State-of-the-art defenses and open problems. In APSYS'11. Article No.5.

6 Santosh Nagarakatte, Jianzhou Zhao, Milo MK Martin, and Steve Zdancewic. 2010. CETS: compiler enforced temporal safety for C. In ISMM'10. 31–40.

7 Byoungyoung Lee, Chengyu Song, Yeongjin Jang, Tielei Wang, Taesoo Kim, Long Lu, and Wenke Lee. 2015. Preventing use-after-free with dangling pointers nullification.. In NDSS'15.

8 Laszlo Szekeres, Mathias Payer, Tao Wei, and Dawn Song. 2013. Sok: eternal war in memory. In SP'13. 48–62.

9 Byoungyoung Lee, Chengyu Song, Yeongjin Jang, Tielei Wang, Taesoo Kim, Long Lu, and

Wenke Lee. 2015. Preventing use-after-free with dangling pointers nullification.. In NDSS'15.

10 Stephen J Fink, Eran Yahav, Nurit Dor, G Ramalingam, and Emmanuel Geay. 2008. Effective typestate verification in the presence of aliasing. ACM Transactions on Software Engineering and Methodology (TOSEM) 17, 2 (2008), 9.

11 Koza J.R., Bennett F.H., Andre D., Keane M.A. (1996) Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming. In: Gero J.S., Sudweeks F. (eds) Artificial Intelligence in Design '96. Springer, Dordrecht

12 Yan H, Sui Y, Chen S, et al. Machine-Learning-Guided Typestate Analysis for Static Use-After-Free Detection[C]// Computer Security Applications Conference. ACM, 2017:42-54.

13 Ting Qin, et al. "A learning algorithm of CMAC based on RLS." Neural Processing Letters 19.1 (2004): 49-61.

14 Siegelmann, H.T.; Sontag, E.D. (1991). "Turing computability with neural nets" (PDF). Appl. Math. Lett. 4 (6): 77–80. doi:10.1016/0893-9659(91)90080-F

15 Balcázar, José (Jul 1997). "Computational Power of Neural Networks: A Kolmogorov Complexity Characterization". Information Theory, IEEE Transactions on. 43 (4): 1175–1183. CiteSeerX 10.1.1.411.7782 Freely accessible. doi:10.1109/18.605580. Retrieved 3 November 2014.

16 Cambria E, White B. Jumping NLP curves: A review of natural language processing research[J]. IEEE Computational intelligence magazine, 2014, 9(2): 48-57.

17 Hirschberg J, Manning C D. Advances in natural language processing[J]. Science, 2015, 349(6245):261.

18 Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate[J]. arXiv preprint arXiv:1409.0473, 2014.

19 Zhou J, Cao Y, Wang X, et al. Deep recurrent models with fast-forward connections for neural machine translation[J]. arXiv preprint arXiv:1606.04199, 2016.

20 Gehring J, Auli M, Grangier D, et al. Convolutional Sequence to Sequence Learning[J]. arXiv preprint arXiv:1705.03122, 2017.

21 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

22 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

23 Yulei Sui and Jingling Xue. SVF: Interprocedural Static Value-Flow Analysis in LLVM , 25th International Conference on Compiler Construction