A Scalable and Accurate Hybrid Vulnerability Analysis Framework

Julian Thomé
SnT Centre for Security, Reliability and Trust
University of Luxembourg, Luxembourg
Email: julian.thome@uni.lu

***Abstract*—Software security assurance is an important process in software development that protects the sensitive data and resources contained in and controlled by the software. Addressing security vulnerabilities at an early phase could decrease the cost of addressing them in later stages by two orders of magnitude. In order to detect vulnerabilities in Web services and Web applications in a scalable and accurate manner, we aim at developing a hybrid vulnerability analysis framework which combines program analysis, symbolic execution and machine learning. We use program analysis to identify potential vulnerable execution branches within the source code for the purpose of guiding the symbolic execution along the potentially vulnerable execution paths. We also propose scalable constraint solving techniques for vulnerability analysis. To further enhance scalability and accuracy, we also apply machine learning by incorporating predictors for identifying potentially vulnerable paths of the program based on known vulnerable cases.**

***Keywords*—Software Security Assurance, Vulnerability Analysis, Program Analysis, Symbolic Execution, Constraint Solving, Machine Learning**

I. Introduction and Research Hypothesis

Web security vulnerabilities such as cross-site scripting (XSS), SQL injection (SQLi), XPath injection (XPathi), XML injection (XMLi) and LDAP injection (LDAPi) could lead to serious attacks that are threats to the continuity of business operations. Addressing them at an early phase could decrease the cost of addressing them in later stages by two orders of magnitude [1].

Common Web security vulnerabilities arise from inadequate sanitisation of user inputs that flow to security-sensitive program operations (sensitive sinks). To address these issues, many approaches such as static and dynamic taint analysis, model checking, symbolic and concolic testing, and machine learning-based vulnerability detection approaches have been proposed.

Static taint analysis approaches [2] track the flow of user inputs and check whether any input data is used in sinks without passing through sanity checks. These approaches often generate too many false alarms because they cannot reason about the correctness and the adequacy of those sanity checks [3], or they do not take into consideration control-flow information [4]. Thus, these approaches are not accurate in general.

Dynamic taint analysis [5], model checking [6], symbolic [7] and concolic [3] testing techniques reason about various paths in the program that lead to sensitive sinks. Although accurate, these techniques typically involve scalability issues [8], [9]. Several approaches have been proposed [10], [11], [12], [13], [8] to address them, but they have yet to be successfully applied to security analysis.

Machine learning systems are systems that learn from data. By learning on a set of code attributes that reflect different program characteristics, they can be used to predict future vulnerabilities [14], [15]. However, as most of these existing approaches identify vulnerabilities at coarse-grain levels (software components, programs), they are not scalable in terms of the manual effort required to identify the vulnerable code locations.

A recent literature study [16] has made two important observations: (1) none of the above-mentioned security techniques alone is efficient enough to address vulnerabilities in Web applications/services; (2) some of the more sophisticated techniques are hard to implement in practice, or they do not scale to real-world systems. Hence, an approach that is scalable and accurate is required to adequately address Web security issues. We propose a hybrid approach that combines and adapts the best of the above-mentioned approaches making it possible to overcome their limitation and to capitalise their strengths.

II. Contributions

The novelty of our research lies in the effective and seamless combination of program analysis, symbolic execution together with machine learning in order to achieve scalability and accuracy in vulnerability analysis. Much of the literature that exists on symbolic execution focuses on solving its scalability without taking the analysis context into account. Our contribution is to investigate this and to develop techniques that allow symbolic execution to scale in the context of Web security vulnerability analysis.

A prototype tool that implements our proposed approach is expected at the end of the project. We expect the tool to be scalable and accurate in analysing common and serious Web security vulnerabilities such as XSS, SQL, XML, XPath, and LDAP injection issues.

III. Research Approach and Results

As the initial steps of the project, we have completed the literature study on vulnerability analysis approaches. From the study, we devise that to develop a scalable and accurate vulnerability analysis approach, the following three work items have to be completed: (1) program slicing of sensitive sinks, (2) symbolic execution of the slices, and (3) machine learning-

978-1-5090-0406-5/15/$31.00 ©2015 IEEE

based vulnerability prediction. Each work item, its status and the results (if present) are explained below.

The first work item is the application of static program analysis to partially address the scalability problem of symbolic execution. Based on program slicing, we develop techniques to extract minimal slices relevant to security. This item has been completed and the results are published [4]. Our program-slicing based approach extracts sound and accurate security slices that are on average 80% smaller than the ones generated by a state-of-the art slicing tool.

In the second work item, we develop techniques to symbolically execute security slices. Since the effectiveness of symbolic execution strongly depends on the underlying constraint solver, we have investigated different solving techniques [17], [3], [18], [19], [20], [21] and concluded that none of them is suitable in our context due to insufficient support of Java language features, the high complexity of usage and the lack of scalability or expandability. We are currently working on an approach for solving numeric and string constraints that does not suffer from these limitations, and which we will combine with our existing work [4] to proof the presence/absence of vulnerabilities in security slices.

For those cases where symbolic execution does not arrive at a conclusion, we shall investigate the application of machine learning. A recent approach [22] addressed this problem by combining program analysis with machine learning. However, being a probabilistic approach, its accuracy will not match that of symbolic execution-based approaches. To address this, we shall adapt this approach to utilise the program behavioural and structural information produced by our symbolic execution step.

IV. Evaluation and Dissemination Plan

To evaluate the practical utility of our proposed approach, case studies and experiments on both open source and industrial software systems shall be conducted. We aim to achieve scalability and accuracy in vulnerability analysis. As a quantitative objective, we aim to detect at least 75% of the vulnerabilities with a false alarm rate $\leq 20\%$. We consider this to be useful in practice, as the approach would then detect most of the vulnerabilities at a reasonably low cost [14], [23].

We plan to collaborate with an industrial partner. Given that security is important for any software company, it would be useful to disseminate our research technology to our partners. Research results will be presented at reputable international software engineering and security conferences (*e.g.* ICSE, FSE, ISSTA, CCS, S&P, USENIX). We also plan to publish in journals such as ACM and IEEE Transaction on Software Engineering.

Acknowledgment

This work is supported by the National Research Fund, Luxembourg (FNR/P10/03 and FNR9132112). I gratefully acknowledge my supervisor Lionel C. Briand (lionel.briand@uni.lu) and my cosupervisor Lwin Khin Shar (lwinkhin.shar@uni.lu).

References

[1] A. Arora and C. Mellon, "Estimating Benefits from Investing in Secure Software Development," vol. 265, pp. 1–12, 2005.

[2] P. M. Pérez, J. Filipiak, and J. M. Sierra, "LAPSE+ static analysis security software: Vulnerabilities detection in Java EE Applications," *Communications in Computer and Information Science*, vol. 184 CCIS, no. PART 1, pp. 148–156, 2011.

[3] A. Kieżun, V. Ganesh, P. J. Guo, P. Hooimeijer, and M. D. Ernst, "HAMPI: A solver for string constraints," in *ISSTA 2009, Proceedings of the 2009 International Symposium on Software Testing and Analysis*, Chicago, IL, USA, July 21–23, 2009, pp. 105–116.

[4] J. Thomé, L. K. Shar, and L. C. Briand, "Security Slicing for Auditing XML, XPath, and SQL Injection Vulnerabilities," *ISSRE*, 2015.

[5] "Saner: Composing static and dynamic analysis to validate sanitization in web applications," *Proceedings - IEEE Symposium on Security and Privacy*, pp. 387–401, 2008.

[6] "Automatic Generation of XSS and SQL Injection Attacks with Goal-Directed Model Checking," *Symposium A Quarterly Journal In Modern Foreign Literatures*, pp. 31–43, 2008.

[7] X. Fu and C.-c. Li, "A String Constraint Solver for Detecting Web Application Vulnerability," *Proceedings of the 22nd International Conference on Software Engineering & Knowledge Engineering - SEKE'2010*, pp. 535–542, 2010.

[8] "Directed symbolic execution," *Lecture Notes in Computer Science*, vol. 6887 LNCS, pp. 95–111, 2011.

[9] M. Borges, M. D'Amorim, S. Anand, D. Bushnell, and C. S. Pasareanu, "Symbolic execution with interval solving and meta-heuristic search," *Proceedings - IEEE 5th International Conference on Software Testing, Verification and Validation, ICST 2012*, no. 1, pp. 111–120, 2012.

[10] C. S. Pasareanu and W. Visser, "Verification of Java Programs using Symbolic Execution and Invariant Generation."

[11] S. Anand, C. S. Psreanu, and W. Visser, "Symbolic execution with abstract subsumption checking," *Lecture Notes in Computer Science*, vol. 3925 LNCS, pp. 163–181, 2006.

[12] P. Godefroid, "Compositional dynamic test generation," *ACM SIGPLAN Notices*, vol. 42, no. 1, p. 47, 2007.

[13] S. Anand, P. Godefroid, and N. Tillmann, "Demand-driven compositional symbolic execution," *Lecture Notes in Computer Science*, vol. 4963 LNCS, pp. 367–381, 2008.

[14] Y. Shin, A. Meneely, L. Williams, and J. a. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE Transactions on Software Engineering*, vol. 37, no. 6, pp. 772–787, 2011.

[15] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," *Proceedings of the 14th ACM conference on Computer and communications security CCS 07*, p. 529, 2007.

[16] L. K. Shar and H. B. K. Tan, "Defeating SQL injection," *Computer*, vol. 46, no. 3, pp. 69–77, 2013.

[17] X. Fu, M. C. Powell, M. Bantegui, and C. C. Li, *Simple linear string constraints*, 2013, vol. 25, no. 6.

[18] Y. Zheng, "Z3-str : A Z3-Based String Solver for Web Application Analysis," *Fse 2013*, 29.

[19] T. Liang, A. Reynolds, C. Tinelli, C. Barrett, and M. Deters, "A DPLL(T) theory solver for a theory of strings and regular expressions," *Lecture Notes in Computer Science*, vol. 8559 LNCS, pp. 646–662, 2014.

[20] P. Saxena, D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song, "A symbolic execution framework for javascript."

[21] G. Redelinghuys, W. Visser, and J. Geldenhuys, "Symbolic execution of programs with strings," in *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference*, ser. SAICSIT '12. New York, NY, USA: ACM, 2012, pp. 139–148.

[22] L. K. Shar, H. Beng Kuan Tan, and L. C. Briand, "Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis," *Proceedings - International Conference on Software Engineering*, pp. 642–651, 2013.

[23] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," vol. 33, no. 1, pp. 2–14, 2007.