The Integration of Multi-Color Taint-Analysis with Dynamic Symbolic Execution for Java Web Application Security Analysis

Malte Mues, TU Dortmund University, Dortmund, Germany

Reference
Malte Mues. *The Integration of Multi-Color Taint-Analysis with Dynamic Symbolic Execution for Java Web Application Security Analysis.* Dissertation, TU Dortmund University, 2023, DOI: 10.17877/DE290R-23694

Supervisor: Prof. Dr. Falk Howar

Date of Defense: 02.03.2023

Abstract

The view on IT security in today's software development processes is changing. While IT security used to be seen mainly as a risk that had to be managed during the operation of IT systems, a class of security weaknesses is seen today as measurable quality aspects of IT system implementations, e.g., the number of paths allowing SQL injection attacks. In consequence, we need tools that can measure and assess the quality of an IT system regarding the presence of security weaknesses before shipping the final software product. Literature traditionally categorizes such tools into dynamic and static security analyzers with hybrid solutions in between that are static analyses incorporating dynamic information or vice versa.

In my thesis, I present the design of a dynamic security analyzer called JAINT that combines dynamic tainting as a pathwise security policy enforcing technique with dynamic symbolic execution as a path enumeration technique. More specifically, the thesis looks into SMT meta-solving, extending dynamic symbolic execution on Java programs with string operations, and the configuration problem of multi-color taint analysis in greater detail to enable JAINT for the analysis of Java web applications. The evaluation in Figure 1 demonstrates that the resulting framework is the best research tool on the OWASP JAVA Benchmark. JDART, one of the two dynamic symbolic execution engines that I worked on as part of the thesis has won gold in the Java track of SV-COMP 2022. GDART, the other dynamic symbolic execution engine, demonstrates that it is possible to lift the implementation design from the research-specific JAVA PATHFINDER VM to the industry grade GRAALVM, paving the way for the future scaling of Jaint.

Keywords— Dynamic Symbolic Execution, Automated Software Testing, SMT Solving, IT Security, Multi-Color Taint Analysis, Software Engineering, Software Verification

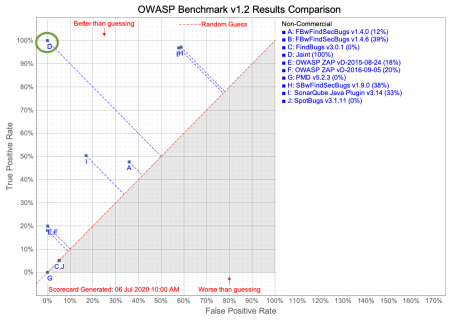

Figure 1: JAINT (D) on the OWASP Java Benchmark

1 Introduction

The log4shell vulnerability[1] has demonstrated in 2021 that injection based attacks are still a significant IT security risk. The knowledge about these weaknesses has existed for many years now, however, we are still missing tools to precisely detect them. From a technical viewpoint, detecting paths vulnerable to injection attacks is a sweet point for dynamic taint analysis (e.g., as proposed by Haldar et al. [4]). If a security weakness exists along the executed path, dynamic taint analysis is guaranteed to find it and is precise. Otherwise, it proves the security for the currently executed path. Multi-color taint analysis tracks different taint colors dynamically and maps them to different security policies. This allows analyzing the same path for multiple security weaknesses in parallel. Applying dynamic tainting in a software verifier requires a path enumerator for establishing guarantees covering all reachable paths.

JAINT [11] combines dynamic symbolic execution as a path enumerator with multi-color dynamic tainting resulting in a security analyzer for JAVA web application. The implementation is based on the directed automated random testing idea by Godefroid et al. [3] and started from the JDART tool introduced by Luckow et al. [5] implementing dynamic symbolic execution for the JVM. As long as the search component in dynamic symbolic execution terminates, JAINT guarantees the absence of security weaknesses within the search space if no weaknesses are found. The following two sections will briefly discuss my results and contributions.

2 Contributions

The first proof of concept of JAINT [11] has been built using the JAVA PATHFINDER VM [14] (JPF-VM) and JDART [5]. To enable JAINT for web applications, the thesis makes three contributions briefly highlighted here: introduction of symbolic strings, moving to the GRAALVM, and the design of SMT meta-solving strategies.

[1] `https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2021-44228`

Symbolic Strings. JDART used to implement dynamic symbolic execution for the primitive types in JAVA. Analyzing web applications requires a representation of strings. The thesis proposes the integration of SMTLib string encoding into the symbolic constraintstree making constraints simpler to handle than using a bitvector-based encoding. We have reported first experiments as part of SV-COMP [6, 8] and summarized the overall result with the shift to SPouT [10]. SMTLib strings are an integral part of JDART winning the SV-COMP 2022 JAVA track.

Moving to the GraalVM. The JPF-VM is a great model checker for JAVA. It enabled many research projects. However, neither dynamic symbolic execution nor dynamic tainting needs any of the model-checking capabilities of the JPF-VM. Therefore, the thesis introduced GDART [9, 10], a dynamic symbolic execution engine built on top of the industry-grade GRAALVM. Until today, the implementation has been extended with dynamic tainting so that JAINT runs on top of the GRAALVM infrastructure. As a result, JAINT can be applied to arbitrary JVM code today, allowing to analyze, e.g., Spring Boot applications. Moreover, this transition paves the way for analyzing any other JVM language, e.g., Kotlin or Scala.

SMT Meta-Solving Strategies. Participating with JDART [8] at SV-COMP, we demonstrated the performance benefits of using more than one SMT solver. While the SMT solver community established with Z3 [2] and cvc5 [1] the current state-of-the-art for bitvector and floating-point theory solvers, there is a higher variety of string theory solvers. We proposed SMT meta-solving strategies [7] to demonstrate a data-driven approach for designing an SMT solver backend with string solvers. The paper explains the tradeoffs between using a single SMT solver and using multiple SMT solvers. It was awarded as an ASE ACM Sigsoft distinguished paper award.

3 Discussion and Outlook

While the theoretical foundations for dynamic symbolic execution and dynamic tainting have been formalized by Schwartz et al. [12] in 2010, it took over a decade before we introduced with JAINT [11] the first tool that combines both techniques for the analysis of modern JAVA web applications. The closest approach I am aware of for other languages is BITBLAZE by Song et al. [13] for x86 assembly. In direct comparison with BITBLAZE, JAINT benefits from the stricter memory access rules in the JVM than in x86 assembly. From my point of view, the JVM is better suited for security analysis with tainting as tainting can enforce security properties without having to guard the memory stack. This makes the taint space more focused than for x86.

JAINT is very precise but its scalability is affected by existing problems in symbolic execution resulting from unbounded state spaces. In the long run, I expect the approach is only able to unfold its full potential in combination with a static analysis that prunes path not influencing the security property from the search space, potentially using overapproximations. JAINT checks the smaller state space as a second step with high precision filtering potential false positives in the static analysis. This allows static analysis to leverage its broader perspective for finding vulnerable paths and dynamic analysis to run finer checks along these paths ensuring precision.

References

[1] H. Barbosa, C. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, et al. cvc5: A versatile and industrial-strength smt solver. In *TACAS*, pages 415–442. Springer, 2022.

[2] L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *TACAS*, pages 337–340. Springer, 2008.

[3] P. Godefroid, N. Klarlund, and K. Sen. Dart: Directed automated random testing. In *PLDI*, PLDI '05, pages 213–223. ACM, 2005.

[4] V. Haldar, D. Chandra, and M. Franz. Dynamic taint propagation for Java. In *21st Annual Computer Security Applications Conference (ACSAC'05)*, pages 9–pp. IEEE, 2005.

[5] K. Luckow, M. Dimjašević, D. Giannakopoulou, F. Howar, M. Isberner, T. Kahsai, Z. Rakamarić, and V. Raman. Jdart: A dynamic symbolic analysis framework. In *TACAS*. Springer, 2016.

[6] M. Mues and F. Howar. JDart: Dynamic symbolic execution for Java bytecode (competition contribution). In *TACAS*, pages 398–402. Springer, 2020.

[7] M. Mues and F. Howar. Data-driven design and evaluation of smt meta-solving strategies: Balancing performance, accuracy, and cost. In *ASE*, pages 179–190, 2021.

[8] M. Mues and F. Howar. Jdart: Portfolio solving, breadth-first search and smt-lib strings. In *TACAS*. Springer, 2021.

[9] M. Mues and F. Howar. Gdart: An ensemble of tools for dynamic symbolic execution on the java virtual machine (competition contribution). In *TACAS*. Springer, 2022.

[10] M. Mues, F. Howar, and S. Dierl. Spout: Symbolic path recording during testing - a concolic executor for the jvm. In *SEFM*, pages 91–107. Springer, 2022.

[11] M. Mues, T. Schallau, and F. Howar. Jaint: A framework for user-defined dynamic taint-analyses based on dynamic symbolic execution of Java programs. In B. Dongol and E. Troubitsyna, editors, *IFM*. Springer, 2020.

[12] E. J. Schwartz, T. Avgerinos, and D. Brumley. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *2010 IEEE S&P*, pages 317–331. IEEE, 2010.

[13] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena. Bitblaze: A new approach to computer security via binary analysis. In R. Sekar and A. K. Pujari, editors, *Information Systems Security*, pages 1–25, Berlin, Heidelberg, 2008. Springer.

[14] W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda. Model checking programs. *ASE*, 10(2):203–232, 2003.