JAINT: A Framework for User-Defined Dynamic Taint-Analyses based on Dynamic Symbolic Execution of Java Programs

Malte Mues[1], Till Schallau[1], Falk Howar[1]

Abstract: We summarize the paper „Jaint: A Framework for User-Defined Dynamic Taint-Analyses Based on Dynamic Symbolic Execution of Java Programs“, published at the sixteenth international conference on integrated formal methods in November 2020 [MSH20]. Reliable and scalable methods for security analyses of JAVA applications are an important enabler for a secure digital infrastructure. In this paper, we present a security analysis that integrates dynamic symbolic execution and dynamic multi-colored taint analysis of JAVA programs, combining the precision of dynamic analysis with the exhaustive exploration of symbolic execution. We implement the approach in the JAINT tool, based on JDART [Lu16], a dynamic symbolic execution engine for JAVA PathFinder, and evaluate its performance by comparing precision and runtimes to other research tools on the OWASP benchmark set. The paper also presents a domain-specific language for taint analyses that is more expressive than the source and sink specifications found in publicly available tools and enables precise, CWE-specific specification of undesired data flows. This summary presents JAINT's language and the evaluation.

Keywords: Dynamic Symbolic Execution; Domain Specific Languages; Java Bytecode Analysis; Dynamic Taint Analysis

Specification of Taint Analyses

JAINT provides a domain-specific language for specifying undesired data flows from tainted sources to protected sinks that may be interrupted by flow through sanitization methods. The paper presents the grammar of the language along with usage examples. Here, we only present one small example: command injection attacks (CWE 78[2]) use parameters of a HTTP request as executable commands in a shell, i.e., in a command that is executed as a new process. Methods that match patterns `Runtime.exec(*)` and `ProcessBuilder.*(command)` are considered protected sinks:

$$Src ::= cmdi \quad \leftarrow (\_ : *\texttt{HttpServletRequest}).\texttt{get}*()$$
$$Sink ::= cmdi \quad \rightarrow (\_ : \texttt{java.lang.Runtime}).\texttt{exec}(*), (\_ : \texttt{java.lang.ProcessBuilder}).*(\texttt{command})$$

The zenodo archive accompanying the paper contains a version of JAINT with taint specifications for the eleven classes of CWEs in the OWASP benchmark set.[3]

[1] TU Dortmund, LS XIV Software Engineering - Automated Quality Assurance Group, Otto-Hahn-Str. 12, 44227 Dortmund, Deutschland {malte.mues,till.schallau,falk.howar}@tu-dortmund.de

[2] https://cwe.mitre.org/data/definitions/78.html

[3] `http://doi.org/10.5281/zenodo.4060244`

doi:10.18420/SE2021_27

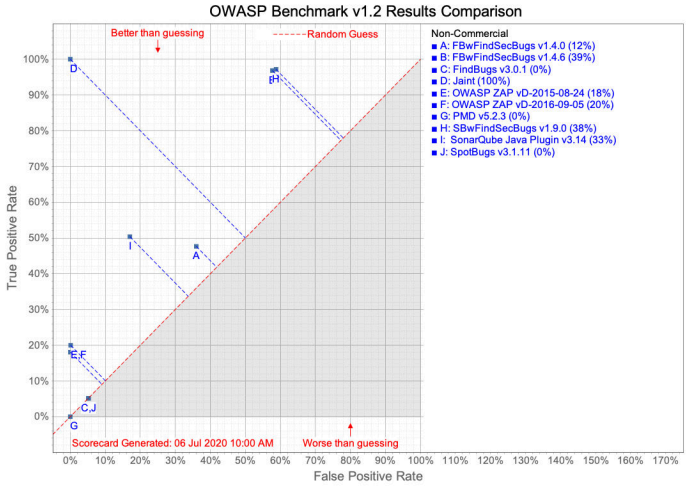

Fig. 1: Precision of JAINT on the OWASP benchmark set [MSH20].

Performance of JAINT on the OWASP benchmark set

Figure 1 shows the precision of JAINT on the OWASP benchmark set. The paper contains a detailed discussion and more obtained results, including runtimes. Most of the compared tools fall in one of two categories: dynamic analyses are precise but miss many vulnerabilities (lower left corner in the plot). Static analyses discover many vulnerabilities but suffer from high false positive rates (upper right corner of the plot). JAINT, in contrast, combines the exhaustive exploration of symbolic execution with the precision of dynamic analysis (upper left corner of the plot).

In the paper, we showed that JAINT beats the OWASP benchmark. This, of course, can only serve as initial validation of the approach: most benchmark instances consist only of few, easy to hit execution paths. We plan future work in two directions: (1) validation of JAINT on real-world examples, and (2) development of a more realistic benchmark set.

Bibliography

[Lu16] Luckow, Kasper Søe; Dimjasevic, Marko; Giannakopoulou, Dimitra; Howar, Falk; Isberner, Malte; Kahsai, Temesghen; Rakamaric, Zvonimir; Raman, Vishwanath: JDart: A Dynamic Symbolic Analysis Framework. In (Chechik, Marsha; Raskin, Jean-François, eds): Proceedings of TACAS 2016. volume 9636 of LNCS. Springer, pp. 442–459, 2016.

[MSH20] Mues, Malte; Schallau, Till; Howar, Falk: Jaint: A Framework for User-Defined Dynamic Taint-Analyses Based on Dynamic Symbolic Execution of Java Programs. In (Dongol, Brijesh; Troubitsyna, Elena, eds): Proceedings of IFM 2020. volume 12546 of LNCS. Springer, pp. 123–140, 2020.