Optimizing Generic Taint Analysis

John Galea (PhD Student) and Daniel Kroening

Department of Computer Science
University of Oxford

Abstract. Dynamic taint analysis [11] is a popular technique in software security for tracking suspicious data, e.g., user input, as it flows during execution of a target application. The analysis associates taint tags with data that stems from introductory sources, such as reads from files, and propagates these tags to other memory locations according to data movements. At particular execution points, the tags of memory locations are checked to determine whether certain runtime properties hold, e.g., whether or not any control data, such as function pointers, are tainted and potentially under the control of a malicious attacker. Overall, taint analysis has been widely employed for a number of applications including vulnerability discovery [10, 3, 4], malware analysis [13, 8, 1], and runtime attack detection [9, 6, 12].
However, despite its usefulness, taint analysis is known to incur severely high runtime overheads. Essentially, the bottleneck stems from additional code that is instrumented into the target application in order to propagate taint. The performance problem is even more severe when generic taint analysis [5] is performed, as this flexible variant allows analysts to implement their own custom propagation logic, instead of using efficient bitwise OR operations to merge taint [7].
Therefore, we are investigating optimizations aimed at reducing the overhead of generic taint analysis on x86 binary applications. In particular, we propose a just-in-time approach that generates fast paths for taint propagation at runtime. The approach monitors for frequent taint states related to the inputs and outputs of basic blocks and accordingly generates fast paths that has instrumentation inserted only for those instructions that deal with taint. This is in contrast to traditional taint trackers which always execute basic blocks that are fully instrumented. Results show that the proposed optimization reduces the slowdown of 36x down to 17x over native execution on the SpecCPU 2017 benchmarks [2]. Moreover, our taint tracker also outperforms existing state-of-the-art implementations of generic taint analysis.

References

1. Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. Scalable, behavior-based malware clustering. In *Network and Distributed System Security Symposium*, pages 8–11. Internet Society, 2009.
2. James Bucek, Klaus-Dieter Lange, et al. SPEC CPU2017: Next-Generation Compute Benchmark. In *International Conference on Performance Engineering*, pages 41–42. ACM, 2018.

3. Juan Caballero, Gustavo Grieco, Mark Marron, and Antonio Nappa. Undangle: Early detection of dangling pointers in use-after-free and double-free vulnerabilities. In *International Symposium on Software Testing and Analysis*, pages 133–143. ACM, 2012.
4. Peng Chen and Hao Chen. Angora: Efficient fuzzing by principled search. In *Symposium on Security and Privacy*, pages 711–725. IEEE, 2018.
5. James Clause, Wanchun Li, and Alessandro Orso. Dytan: A generic dynamic taint analysis framework. In *International Symposium on Software Testing and Analysis*, pages 196–206. ACM, 2007.
6. Dongseok Jang, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. An empirical study of privacy-violating information flows in JavaScript web applications. In *Computer and Communications Security*, volume 10, pages 270–283. ACM, 2010.
7. Vasileios P Kemerlis, Georgios Portokalidis, Kangkook Jee, and Angelos D Keromytis. LibDFT: Practical dynamic data flow tracking for commodity systems. In *ACM Sigplan Notices*, volume 47, pages 121–132. ACM, 2012.
8. David Korczynski and Heng Yin. Capturing malware propagations with code injections and code-reuse attacks. In *Computer and Communications Security*, pages 1691–1708. ACM, 2017.
9. James Newsome and Dawn Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Network and Distributed System Security Symposium*, pages 3–4. Internet Society, 2005.
10. Sanjay Rawat, Vivek Jain, Ashish Kumar, Lucian Cojocar, Cristiano Giuffrida, and Herbert Bos. VUzzer: Application-aware evolutionary fuzzing. In *Network and Distributed System Security Symposium*, pages 1–14. Internet Society, 2017.
11. Edward J Schwartz, Thanassis Avgerinos, and David Brumley. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *Symposium on Security and Privacy*, pages 317–331. IEEE, 2010.
12. Philipp Vogt, Florian Nentwich, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Cross site scripting prevention with dynamic data tainting and static analysis. In *Network and Distributed System Security Symposium*, volume 2007, page 12. Internet Society, 2007.
13. Heng Yin, Dawn Song, Manuel Egele, Christopher Kruegel, and Engin Kirda. Panorama: Capturing system-wide information flow for malware detection and analysis. In *Computer and Communications Security*, pages 116–127. ACM, 2007.