DifFuzz: Differential Fuzzing for Side-Channel Analysis

Shirin Nilizadeh¹, Yannic Noller², Corina S. Păsăreanu³

Abstract: This summary is based on our research results on “DifFuzz: Differential Fuzzing for Side-Channel Analysis” which was published in the proceedings of the 41st International Conference on Software Engineering [NNP19]. Side-channel analysis aims to investigate the risk that a potential attacker can infer any secret information through observations of the system, such as the execution time or the memory consumption. Side-channel vulnerabilities therefore represent security risks that can cause serious damage and need to be identified and repaired. DifFuzz applies differential fuzzing to identify inputs that trigger such vulnerabilities. Our fuzzing approach analyzes multiple program executions, which vary in their secret information, and uses resource-guided heuristics to identify inputs that maximize the observable cost difference between these executions. Our evaluation shows that such a dynamic analysis approach can find the same side-channel vulnerabilities as state-of-the-art static analysis techniques, and even more vulnerabilities since it does not rely on models for its analysis. Additionally, the advantage of DifFuzz compared to other techniques is not only that it can generate inputs that show a vulnerability, but that the resulting cost difference can also be used to estimate the severity of an identified vulnerability. This enables the comparing of repaired versions of an application.

Keywords: Vulnerability Detection; Side-Channel Analysis; Dynamic Analysis; Fuzzing

Summary

Side-channel vulnerabilities might cause the leakage of sensitive information by observing non-functional characteristics of the program execution, such as the execution time, memory usage, response size, network traffic or power consumption. Popular side-channel attacks like Meltdown and Spectre [Th18] highlight the increased need for tools and techniques that can effectively discover side channels before they are exploited by a malicious user in the field. However, it is difficult to reason about side-channel vulnerabilities as they involve analyzing correlations between resource usage over multiple program paths.

Our approach is similar to the well-known method of self-composition [BDR04], which is used to check that no matter what the secret is, the program yields the same output. If that is the case, the program is said to satisfy non-interference, meaning that the program leaks no information; otherwise, the program is vulnerable. Although satisfying non-interference is a sound guarantee for a system to be secure, this requirement is too strict for the side-channel analysis of most realistic programs. Particularly for timing channels, small differences in computations may be imperceptible to an attacker and can thus not be exploited in practice. This problem was observed before [An17, CFD17] and was formalized as checking

¹ University of Texas at Arlington, USA, shirin.nilizadeh@uta.edu
² Humboldt-Universität zu Berlin, Germany, yannic.noller@hu-berlin.de
³ Carnegie Mellon University Silicon Valley, NASA Ames Research Center, USA, corina.s.pasareanu@nasa.gov

doi:10.18420/SE2020_37
\( \epsilon \)-bounded non-interference: not only programs with zero interference can be accepted as secure, but also programs where the difference between observations is too small (below a threshold \( \epsilon \)) to be exploitable in practice.

**DifFuzz** uses a form of differential fuzzing, in which it analyzes the program with the same public inputs but with different secret values, and computes the difference in the side channel measurements observed over the two executions. Therefore, it guides the exploration to find inputs that maximize the cost difference between two program executions, for which only the secret values are different:

\[
\text{maximize: } \delta = \left| c(P[pub, sec_1]) - c(P[pub, sec_2]) \right|
\]

If the difference is large, then it means that the program has a side-channel vulnerability, which should be remedied by the developer.

We implemented **DifFuzz** on top of **AFL** [Za14], a state-of-the-art, security-oriented grey-box fuzzer. Our evaluation showed that **DifFuzz** can keep up with existing approaches such as **Blazer** [An17] and **Themis** [CFD17], two state-of-the-art analysis tools for finding side channels in **Java** programs. Both tools perform static analysis and can in principle guarantee absence of side channels, but may also give false alarms due to underlying over-approximation. In contrast **DifFuzz** performs a dynamic analysis, but it can not prove absence of vulnerabilities. Furthermore, **DifFuzz** found new vulnerabilities in popular open-source **Java** applications such as Apache FtpServer. In the future, we plan to explore automated repair methods to eliminate the vulnerabilities discovered with **DifFuzz**.

**Bibliography**


