Dataflow-based Binary Fuzzing

Kai Feng, University of Glasgow
k.feng.2@research.gla.ac.uk

Fuzzing is a widely used technique for identifying vulnerabilities by generating test cases that maximize code coverage. Most modern fuzzers evolve a set of seed inputs through small mutations, guided by coverage feedback from the system under test. However, applying fuzzing to embedded systems is challenging because firmware is tightly coupled with specific hardware, making it difficult to execute in a different environment. Firmware is built for a particular System-on-Chip (SoC), compiled for a specific CPU, and depends on dedicated peripherals.

To adapt fuzzing for embedded systems, researchers have explored four main approaches: rehosting [1], para-rehosting [2], hardware-in-the-loop (HITL) [3], and fully on-device execution [4]. However, embedded systems are highly diverse, with each designed to meet unique power, performance, and cost constraints. These constraints affect every stage of system design, from choosing the instruction set architecture to selecting peripherals and developing software, making fuzzing integration complex. As a result, existing techniques face challenges such as performance overhead, high complexity, lack of tracing support at the driver level, and synchronization delays when interacting with physical hardware.

To overcome these challenges, we propose a novel fuzzing framework that operates directly on hardware, leveraging *hardware breakpoints* for feedback-driven coverage collection. Our approach employs static analysis to extract definition-use (def-use) chains and strategically places breakpoints at definition addresses. When a breakpoint is triggered, it dynamically adjusts to track execution at corresponding use addresses, enabling precise execution flow tracking. Records of definitions and their uses enable fine-grained program behavior analysis. Given the limited number of hardware breakpoints available on microcontrollers, we introduce a weighted breakpoint relocation strategy to optimize resource allocation while maximizing execution path exploration.

Our framework enhances the effectiveness of fuzzing by enabling realistic firmware execution and improving driver and embedded software analysis in IoT environments. By addressing the limitations of existing methods, this approach provides a scalable and efficient solution for security testing in embedded systems.

References

[1] Bo Feng, Alejandro Mera, and Long Lu. P2IM: Scalable and hardware-independent firmware testing via automatic peripheral interface modeling. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1237–1254, 2020.

[2] Abraham A Clements, Eric Gustafson, Tobias Scharnowski, Paul Grosen, David Fritz, Christopher Kruegel, Giovanni Vigna, Saurabh Bagchi, and Mathias Payer. HALucinator: Firmware re-hosting through abstraction layer emulation. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1201–1218, 2020.

[3] Marius Muench, Dario Nisi, Aurélien Francillon, and Davide Balzarotti. Avatar 2: A multi-target orchestration platform. In *Proc. Workshop Binary Anal. Res.(Colocated NDSS Symp.)*, volume 18, pages 1–11, 2018.

[4] Alejandro Mera, Changming Liu, Ruimin Sun, Engin Kirda, and Long Lu. SHiFT: Semi-hosted fuzz testing for embedded applications. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 5323–5340, 2024.