AVRS: Emulating AVR Microcontrollers for Reverse Engineering and Security Testing

Michael Pucher, Christian Kudera, Georg Merzdovnik

Problem & Motivation

Background

AVR 8-bit microcontrollers are used in automotive applications, sensor nodes and IoT devices, while also being popular with hobbyists. They are the basis for most boards of the Arduino product line, which in turn is used in a variety of projects, like data logging [5] or control and measurement systems [3].

Due to weak processing power, security can become an afterthought:
- Data received from peripherals is untrusted user input.
- Simple vulnerabilities persist, e.g., stack buffer overflows.
- AVR devices cannot be monitored efficiently during execution.

Methodology

- Analysis of Existing Emulators. Analyze existing open-source AVR emulators and document shortcomings
- AVRS. Implement a new emulator, compensating existing shortcomings
- Evaluation of all Emulators. Compare emulators regarding performance and completeness, examine AVRS improvements
- Fuzzing AVR Firmware. Build fuzzing support on top of AVRS

AVRS

Designed to reuse existing approaches of other emulators, improve on lacking monitoring features and performance

- Instruction Decode. Decode all instructions up front, use internal intermediate representation (IR)
- Instruction Emulation. Emulating IR optimizes to jump table
- Device Differences. Architectural differences are decided at compile-time, using procedural Rust macros
- Peripherals and Monitoring. Provide Rust and RPC API for peripheral implementation, API can hook MMU for monitoring memory access.

Fuzzing

- Implementation. Use boofuzz as input generator, AVRS as emulator
- Crash Detection. Use heuristics presented in [2] to detect crashes
- Example Programs. Test fuzzing using serial protocol on ATtiny104 and ATmega328p, providing fuzzing cores and peripheral implementations

Conclusions & Outlook

- Existing emulators lack completeness with respect to implemented instructions, even in mature, actively-developed emulators.
- Open-source toolchain support for ATtiny and ATxmega is error prone.
- AVRS can achieve competitive performance, as highlighted by figures 2 and 3, while also providing support for missing instructions.
- Fuzzing example shows ease of implementing analysis on top of AVRS.
- Use AVRS in future projects to evaluate symbolic execution/taint tracking on AVR devices, decode to IR allows exploring static analysis on AVR as well.

Figure 1: Overview of AVRS Architecture

Figure 2: Achieved emulator frequencies across all benchmarks

Figure 3: Performance of AES operations in emulators, slower than baseline omitted