Bug detection in embedded environments by fuzzing and symbolic execution

Juraj Vijtiuk · Luka Perkov · Antonio Krog

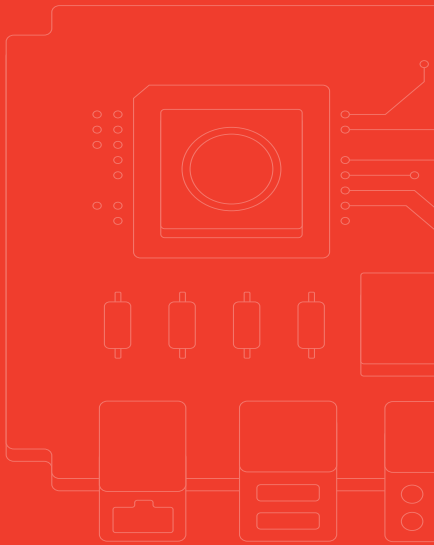
sartura

October 1, 2020

About us

- Embedded Linux development and integration
- Delivering solutions based on Linux, OpenWrt and Yocto
  - Focused on software in network edge and CPEs
- Continuous participation in Open Source projects
- www.sartura.hr

Introduction



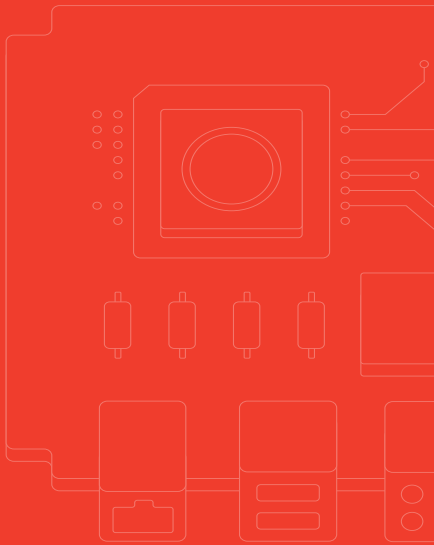
Introduction

- OpenWrt - GNU/Linux distribution aimed at network embedded devices
- RIOT OS - Open Source real-time multithreaded OS for IoT
- Fuzzing - software testing process, uses random inputs and tracks unusual behavior
- Symbolic execution - executes a program with symbolic instead of concrete values

Motivation

- Current state of embedded and IoT security
- Existing work mostly focused on proprietary software
- We previously used both RIOT and OpenWrt
- Compare with other Open Source software and general IoT/embedded security

Methods used



sartura

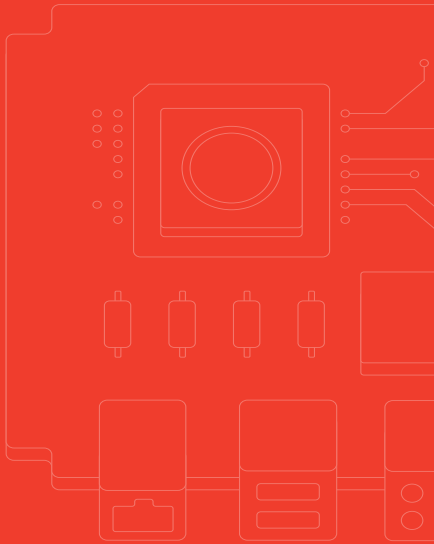
Fuzzing

- Black-box and gray-box based fuzzing
- Mutational and generative input generation
- Can sometimes get ”stuck” on very specific value checks
- E.g. if (var == 0xdeadbeef)
- Often used with ASAN/UBSAN/MSAN

Symbolic execution

- Executes the program with symbolic instead of concrete values, which allows path constraints to be solved easily
- Often used alongside classic fuzzing
- Disadvantages
  - Harder to set up and use
  - State explosion
  - Interaction with the program environment

Testing setup



sartura

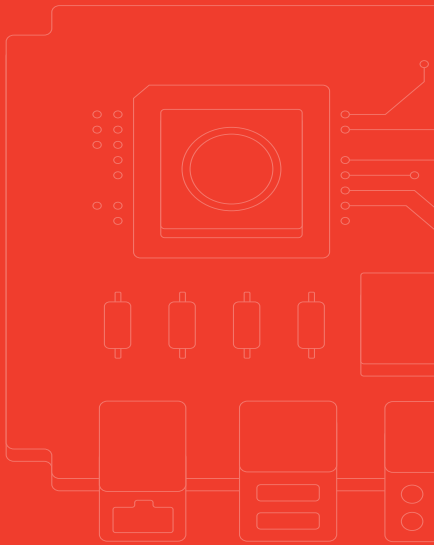
Tools used

- Fuzzers
  - black-box - Radamsa
  - gray-box - AFL, AFLPlusPlus, LibFuzzer, honggfuzz, Angora
  - generational - dharma, gramfuzz
- Symbolic Execution
  - KLEE

Targets

- Openwrt
  - UCI
  - Ubus
  - libubox
- libjson-c
- RIOT-OS

Results



UCI

- AFL
- Existing OpenWrt UCI files used as starting corpus
  - Trial one : three hours, 5 crashes, one bug
  - Trial two : one hour, 3 crashes, one bug
  - Trial three : 12 hours, no crashes
  - Trial four : 12 hours, gramfuzz and dharma starting inputs, no crashes
- Radamsa - no crashes

libubox

- OpenWrt community started fuzzing libubox around the same time
- Several surface bugs were found and fixed
- LibFuzzer
  - Trial one : 12 hours, 0 crashes
  - Trial two : 12 hours, 0 crashes
  - Trial three, ASAN + UBSAN: 12 hours, 0 crashes
  - Trial four, MSAN : 12 hours, 1 crash
- KLEE immediately found the issue
- Some time later, CVE-2020-7248 was found and fixed by Petr Štetiar and Jo-Philipp Wich.

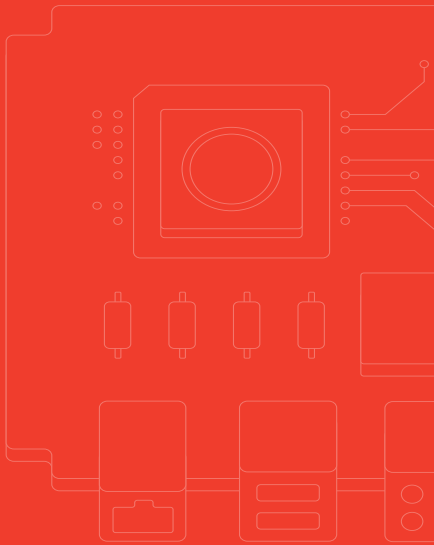
libjson-c

- Fuzzed as part of OSS-Fuzz
- Fuzzed for around 3 days with various fuzzers and KLEE, no crashes
- Continued fuzzing after paper submission
  - In total around two weeks spent on fuzzing, with no crashes found

RIOT

- Previous existing work fuzzed COAP and the TCP stack
  - Multiple bugs were found and fixed
- We started with a custom application that uses the default GNRC network functionality
- Encountered various issues
- Motivated by OpenWrt results, we switched to fuzzing smaller functions with LibFuzzer
- No crashes were found
- Simultaneously, another contributor opened a PR for GNRC AFL fuzzing
- Multiple crashes were found and fixed

Conclusion



Conclusion

- Projects would benefit from simple, elementary fuzz testing
- Surface bugs can be discovered within hours with little effort
- Sanitizers shouldn't be ignored
- Symbolic execution is hard to set up
  - However, elementary testing can reveal surface bugs quicker than fuzzing

Bug detection in embedded environments by fuzzing and symbolic execution

juraj.vijtiuk@sartura.hr