Zagreb, FER

Debugging and bug detection tools for C

Juraj Vijtiuk

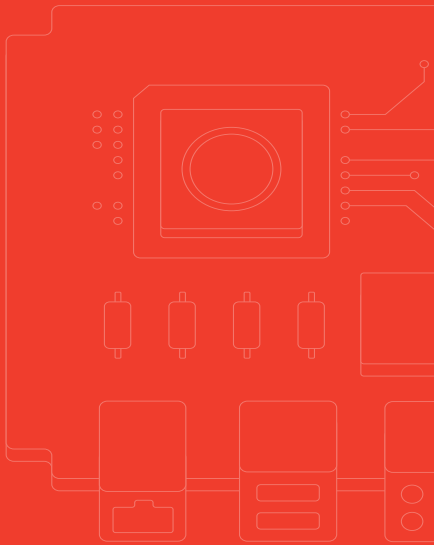
sartura

January 12, 2021

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

Debugging



sartura

- The process of fixing and finding the root cause of bugs
- Shouldn't be confused with troubleshooting
- *Troubleshooting* - assumes a good design, and fixes issues with the use of the design
- *Debugging* - a superset of troubleshooting, includes fixes to the design
- This presentation will focus on UNIX based system, with an emphasis on Linux

The debugging process

- The process of debugging should be approached systematically, using a top down approach, with some of the following steps.
- Get to know the system - read the manuals, source code, examples, previous issues and bug reports
- Make the bug reproducible, document and automate the steps
  - Nondeterministic bugs are problematic

The debugging process cont.

- Collection of information about the problem
  - What triggers the bug (e.g. does the bug still appear after manual changes to the input)
  - What environments does the bug appear in
  - When was the bug introduced
  - Track program state surrounding the bug

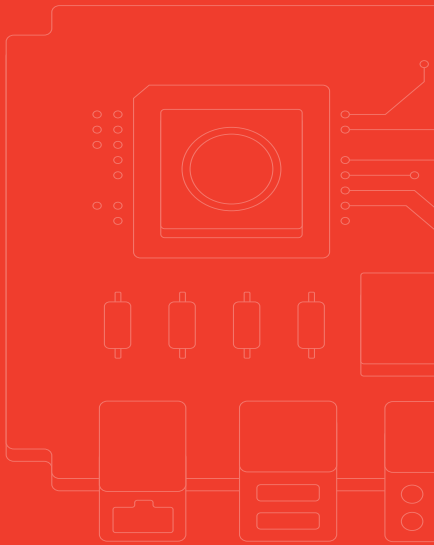
The debugging process

- Divide and conquer while searching for the cause
  - Binary search
  - Use easy to recognise input data patterns
  - Start from the source of the crash/bug and move bottom up

The debugging process

- Check assumptions about the system
  - Check that the tools actually work
- Confirm that the bug really is fixed, and can't be triggered with similar conditions
  - Keeping track of surrounding state helps here

Debugging tools



sartura

Diagnostic tools

- Tools used to collect information about the target at a higher level:
  - strace - used to view system calls to the kernel
  - ltrace - intercepts dynamic library calls
  - dstat - unifies iostat, vmstat and ifstat
  - lsof - show a list of open file descriptors

Diagnostic tools

- Tools used to collect information about the target at a higher level:
  - Network tools
    - Packet capture tools (tcpdump and Wireshark)
    - netcat, ngrep, netstat/ss, socat
  - eBPF tools, BCC
  - perf, flame graphs
- Additional resources at http://www.brendangregg.com/

Debuggers

- gdb, lldb, GUI frontends
- The most common way to use a debugger is by stepping and using breakpoints
- Works with assembly instructions, CPU and memory state
- Debug symbols enabled, optimizations disabled!
- Remote debugging, useful for embedded development
- command line interface and TUI

Debuggers

- Some advanced features are also useful:
  - Automatic expression display, .gdbinit
  - Watchpoints, catchpoints, temporary breakpoints and hardware breakpoints
  - Conditional breakpoints
  - Tracepoints
  - Altering program execution
  - GDB scripting

Timeless Debuggers

- Classic debuggers - ineffective for time sensitive and nondeterministic programs and bugs
- Timeless debuggers can record program execution and then replay it
- Enables reverse stepping, and following execution backwards

Timeless Debuggers

- As gdb has only basic support, other popular tools exist:
  - rr - developed at Mozilla, uses gdb as a backend
  - PANDA, QIRA, radare2 - various frameworks, primarily aimed at reverse engineering and binary analysis
- rr is the most suitable for developer needs

Memory debugging

- Currently two memory debugging tool suites are popular for C/C++ programs
  - *Valgrind* - runtime debugging using a VM and dynamic recompilation, requires no target program modification
  - *The sanitizers project*- ASAN, MSAN, UBSAN, TSAN, which are added at build time

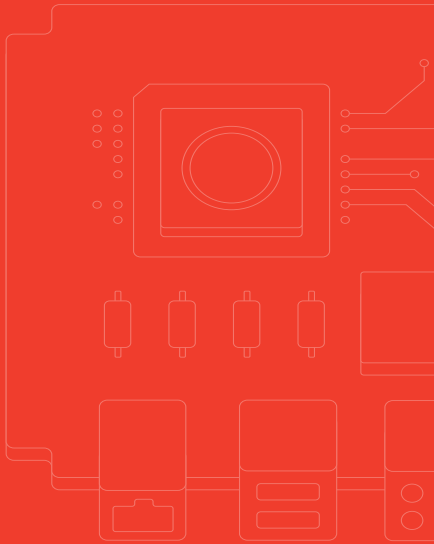
Valgrind

- Valgrind is actually a collection of tools:
  - memchek - a memory error detector
  - cachegrind - cache and branch prediction profiler
  - callgrind - call-graph based cache and branch prediction profiler
  - Helgrind and DRD - thread error detectors
  - Massif and DHAT - heap profilers and analyzers
- Valgrind papers are available at https://www.valgrind.org/docs/pubs.html

Sanitizers

- Similar to valgrind, uses instrumentation and shadow state
- Has to be compiled and linked
- Also a collection of tools:
  - ASAN - memory error detection: leaks, UAFs, buffer overflows
  - MSAN - detects the use of uninitialized memory
  - TSAN - detects data races
  - UBSAN - detects undefined behaviour
- Corresponding variants in the Linux kernel

Proactive bug detection



sartura

Proactive bug detection tools

- Complex software will probably never be completely bug free
  - Halting problem
- Tools and methods can help detect bugs early:
  - Testing and Continuous Integration
  - Non default compiler flags and warnings
  - Detailed debug logging
    - Can be toggled at build time, run time or during program execution
  - Fuzzing
  - Static analysis

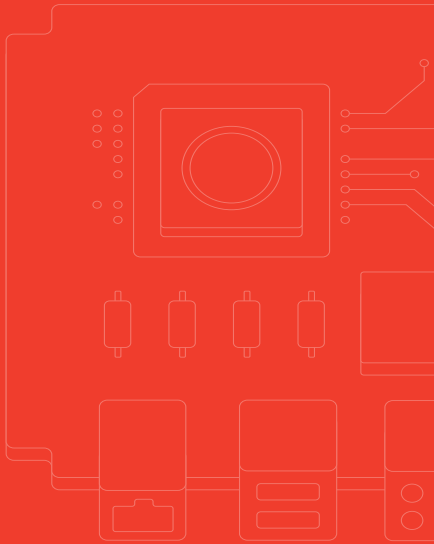
Fuzzing

- Automated software testing by providing unexpected and random data to a program
- The program is watched for any unexpected behaviours:
  - Crashes
  - Hangs
  - Memory errors

Fuzzing

- Mostly used to find security bugs
- Useful for proactive bug finding
- Can be integrated into CI
- LLVM's LibFuzzer most appropriate for developers
- Works even better with sanitizers

Kernel debugging



sartura

Kernel debugging

- Similar to userspace debugging
- Kernels are debugged by:
  - Attaching to a running kernel in a VM
  - Attaching to a running kernel via hardware (JTAG/serial ports)
  - A special kernel configuration is needed
- System map, stack traces
- -O2 by default
- Kernel panics and oops

Kernel tools

- printk, dmesg, systemd tools
- kernel probes, tracepoints - similar to userspace breakpoints
- Ftrace - kernel function tracer
- kgdb, kdb, gdb
- eBPF again
- KASAN, KMSAN, KCSAN - kernel sanitizers
- syzkaller - kernel system call fuzzer

Additional resources

- https://jvns.ca/blog/2017/07/05/linux-tracing-systems/
- http://www.brendangregg.com/blog/2015-07-08/choosing-a-linux-tracer.html
- https://llvm.org/docs/LibFuzzer.html
- https://www.youtube.com/watch?v=PorfLSr3DDI

Debugging and bug detection tools for C

juraj.vijtiuk@sartura.hr

Feedback form: https://forms.gle/YncUDrUJZ89JD1TZA

info@sartura.hr · www.sartura.hr