Zagreb, OSZUR, FER

Debugging and bug detection tools for C

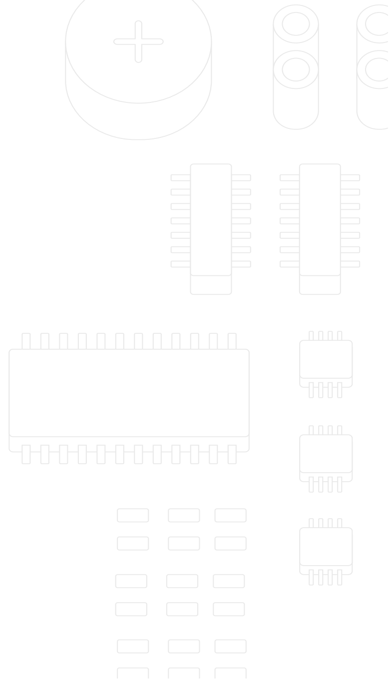
Juraj Vijtiuk

sartura

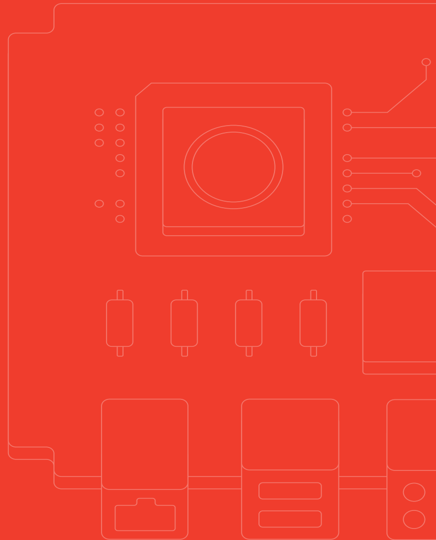
June 4, 2020

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



Debugging



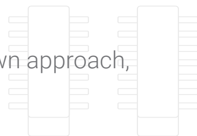
sartura

- The process of fixing and finding the root cause of bugs
- Shouldn’t be confused with troubleshooting
- ***Troubleshooting*** - assumes a good design, and fixess 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
  - 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

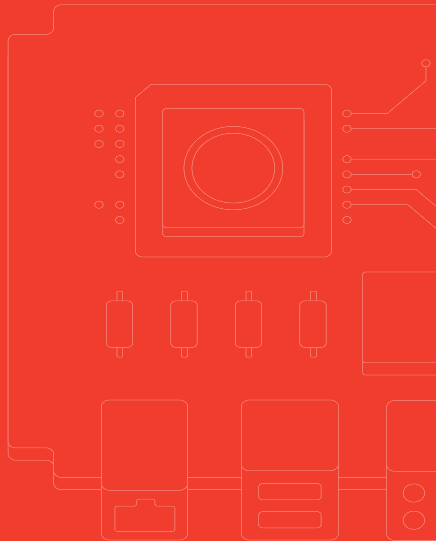
The debugging process cont.



- The process of debugging should be approached systematically, using a top down approach, with some of the following steps:
  - 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
  - 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

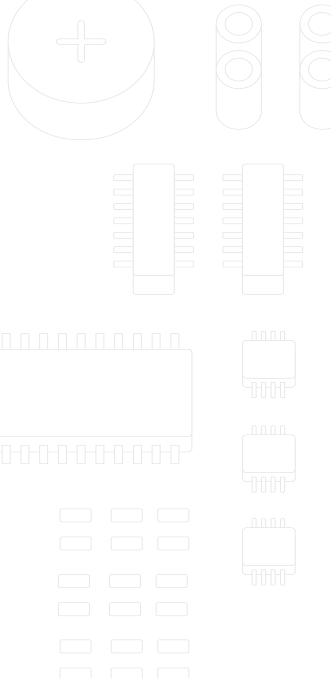
Debugging tools

sartura



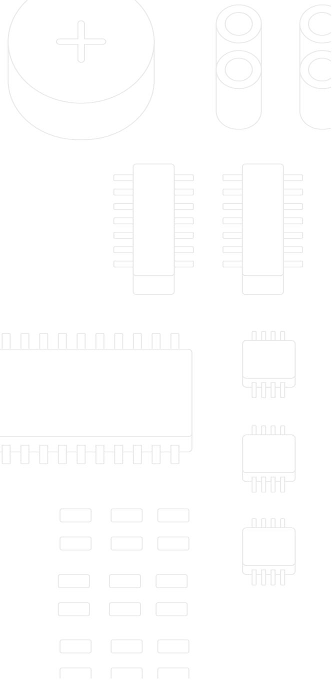
Diagnostic tools

- 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



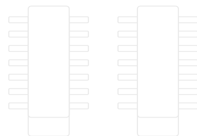
Diagnostic tools

- 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/



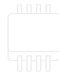
Debuggers

- gdb, lldb and various GUI frontends for both
- The most common way to use a debugger is by stepping and using breakpoints
- However, gdb can also be used to work with assembly instructions, CPU and memory state
- Debug symbols should be available, and optimization disabled!
- Most debuggers support remote debugging, which is useful for embedded development
- Two variations of UI for the command line are available



Debuggers

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

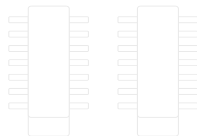


Timeless Debuggers

- Classic debuggers are ineffective when debugging time sensitive and nondeterministic programs and bugs
- Timeless debuggers can record program execution and then replay it
- Another benefit is the ability to reverse step, and follow the program's execution backwards
- As gdb has only basic support, other popular tools exist:
  - rr - developed at Mozilla
  - PANDA - a whole framework for dynamic binary analysis, which also includes record/replay
  - QIRA - also aimed at reverse engineering

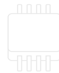
Memory debugging

- 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



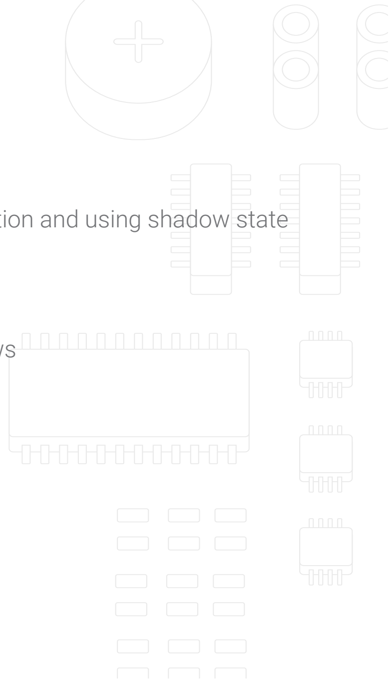
Valgrind

- 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
- The idea was to build a DBI framework based on emulation with a VM and shadow values
- Valgrind papers are available at https://www.valgrind.org/docs/pubs.html

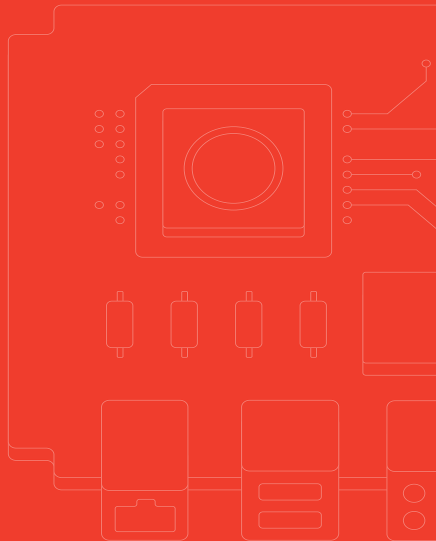


Sanitizers

- At a high level works similarly to valgrind, by adding instrumentation and using shadow state
- However sanitizers have to be added at compile and link time
- 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
- Some of these have corresponding variants in the Linux kernel

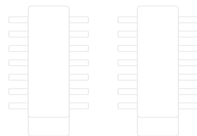


Proactive bug detection

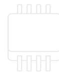


sartura

Proactive bug detection tools

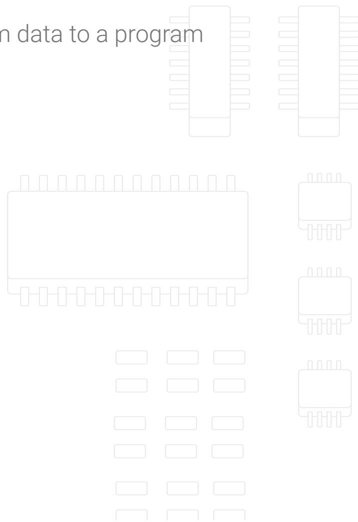


- Complex software will probably never be completely bug free
  - Halting problem
- Tools and methods can help detect bugs early:
  - Testing and Continuous Integration
  - Non default compiler flags and warnings
  - Detailed debug logging
    - Can be toggled at build time, run time or during program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



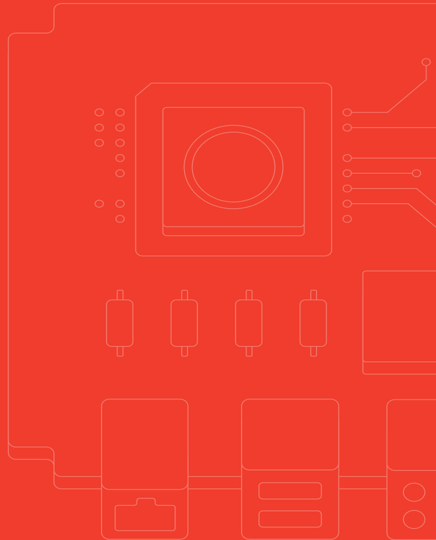
Fuzzing

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



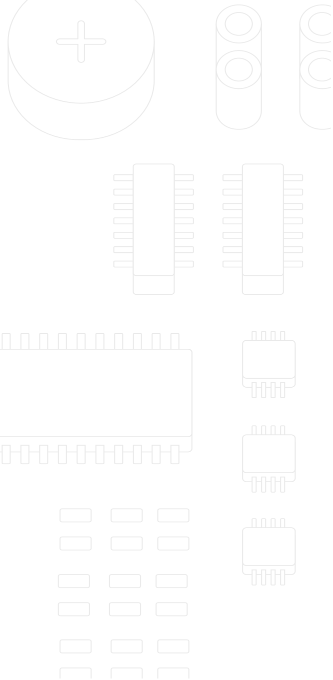
Kernel debugging

sartura

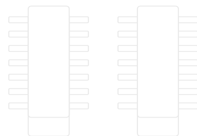


Kernel debugging

- 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



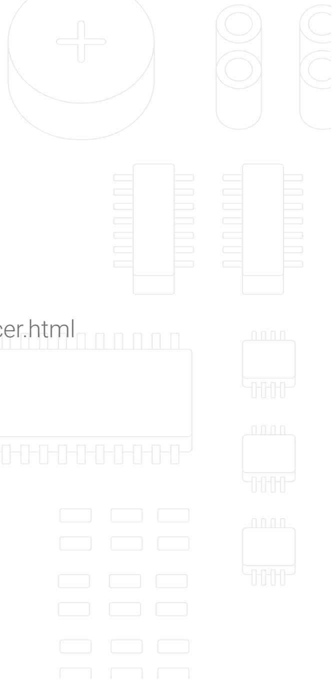
Kernel tools

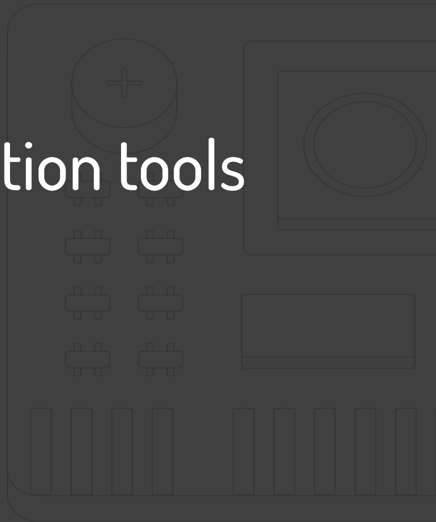


- printk, dmesg, systemd tools
- kernel probes, tracepoints - similar to userspace breakpoints
- Ftrace - kernel function tracer
- kgdb, kdb, gdb
- eBPF again
- KASAN, KMSAN, KCSAN - kernel sanitizers
- syzkaller - kernel system call fuzzer

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/auaBjZgzcg4uoqsS9