CTSRD

CRASH-worthy Trustworthy Systems Research and Development

Informal methods

A personal search for practical alternatives to 'moral improvement through suffering' in systems research

(lightning-talk version)

Robert N. M. Watson
University of Cambridge
Computer Laboratory

10 October 2014

Approved for public release; distribution is unlimited. This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-10-C-0237. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

SRI International

University of Cambridge

Systems research: the good

- ✓ Working on a widely used, open-source, highly concurrent, multi-million-line, C-language OS kernel (FreeBSD)
- ✓ Developing sandbox techniques that mitigate malware on hundreds of millions of devices from routers to mobile phones / tablets daily
- ✓ Helping to develop a novel CPU with tagged memory and capabilities … and adapting a conventional OS to target it; now in tech transition
- ✓ Contributing to a fielded, state-of-the-art, multicore network stack serving (at least) 34% of US domestic Internet traffic…
- ✓ … then helping to develop a clean-slate research stack that runs > 6x faster on the same hardware
- ✓ Having a personal 'top-three' favorite instructions in the OS boot
- ✓ **Crushing the souls of many subtle concurrency bugs after many weeks of effort**

Systems research: the bad

- X Years of effort invested in projects without knowing if they will succeed … or get thrown away … or get scooped
- X Realizing that the interesting problems span layers of abstraction … but that analysis and debugging tools rarely do
- X Discovering that the debugging data I need is always discarded milliseconds (or weeks) before I realize that I need it
- X Solving the same hard problems (many) hundreds of times
- X Describing reproducible problems but not having tools that can use those descriptions to automatically detect the problems
- X **Having my soul crushed by subtle concurrency bugs despite many weeks of effort**

Systems research requires systems development

- Systems research tests hypotheses through the production and evaluation of concrete artifacts – this requires a lot of work.
- Also concerned with root-cause analyses (e., performance.)
- Systems researchers (often) develop fire-and-forget prototypes
  - … and omit key 'hard bits' of systems development:
  - E.g., testable and maintainable artifacts that can last decades.
- Systems researchers should therefore take a *research interest* in techniques/tools to improve productivity…
  - … but tools/techniques jibe poorly with common systems-research narratives.
- **What?**

Tools vs. systems research

So, I created this new tool that …

Tools are not research.

So, I created these new techniques embodied in tools that …

How do you evaluate that?

Well, there is this corpus of …

This is systems: we study things you can measure, and that sounds anecdotal.

**[We create systems, not tools.
That's all fuzzy and human facing.
You wouldn't want to do a *user study*, would you?]**

Techniques and tools as systems

- Not a strict rule: several key systems papers on techniques/tools.
- But it is the surprising advice to many new PhD students.
- We can treat techniques/tools as systems themselves:

  What problems do they solve?
  How are they better than the state-of-the-art?
  Do they appear to work on sensible corpuses?
  Do they have significant false positives/negatives?
  Are they sound? Complete? Generalizable?
  How will they scale with growing corpus size?
  How will they affect development-cycle performance?
  How will they affect run-time performance?
  Can we optimize them for important cases?
  Are there quantitative or qualitative measures of complexity that can stand in for a user study?

How about systems research for the techniques/tools communities

- The flip-side question:

  What value can systems research offer to communities that develop new techniques, and tools embodying them?

- Possible answer:

  The systems community builds, maintains, and even better, *understands*, systems that can act as non-trivial corpora…

  … and has strong ties to industrial communities that will use practical/effective tools targeting systems artifacts

CASE STUDY: BERI PROCESSOR

DARPA CRASH

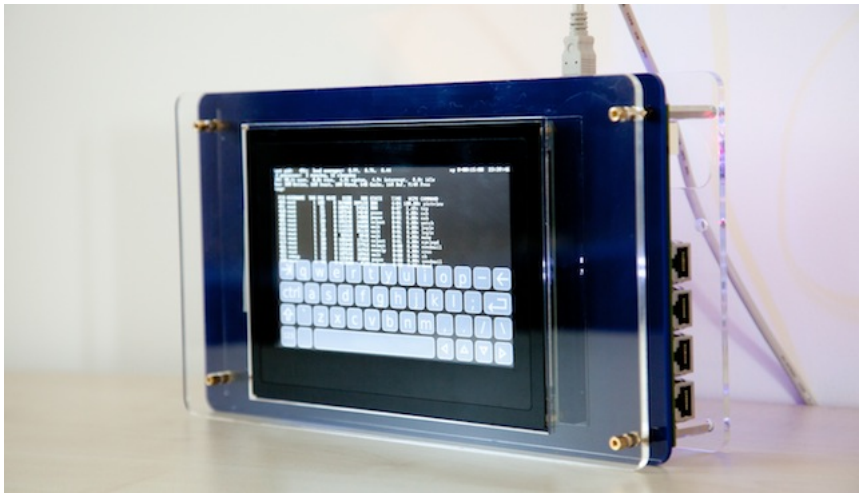
If you could revise the fundamental principles of computer system design to improve security…

…what would you change?

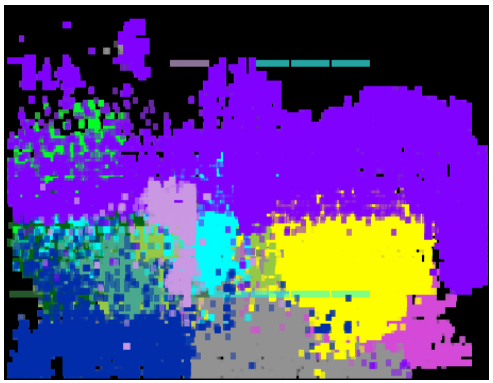
So, we want to do research into the hardware software interface…

…where do we begin?

BERI processor



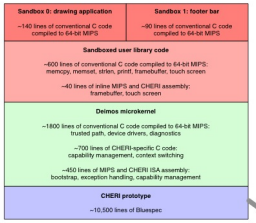
Floating Point	31.8%
BERI Pipeline	18.6%
Capability Unit	14.7%
CPro0 & TLB	7.8%
Level 2 Cache	6.6%
Debug	4.7%
L1 Data Cache	4.6%
Tag Cache	4.0%
Multiply & Divide	2.6%
L1 Instr. Cache	2.4%
Branch Predictor	2.3%



- Research into the hardware-software interface
- Fully pipelined 64-bit FPGA soft core
  - Bluespec System Verilog HDL
  - MIPS ISA + CP0: exceptions, MMU, …
  - Debug unit with GDB target, tracing
  - IEEE Floating Point Unit (FPU)
  - ISA test suite, fuzz tests, ISA model
  - Multi-core, multithreading support
  - Multi-FPGA extensions in progress
- FreeBSD, open-source application stack
  - Drivers for Altera, Xilinx IP cores, Terasic peripherals
  - Clang/LLVM/LLDB toolchain
  - … You can just SSH in!

BERI development timeline

October 2011: Deimos capability-based microkernel runs first sandbox



June 2012: CheriBSD capability context for user threads

July 2012: LLVM generates CHERI ISA code

November 2012: Sandboxed user code runs on CheriBSD; trojan mitigated

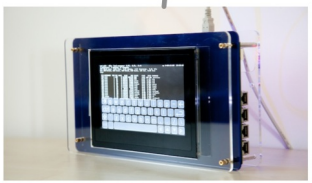
Feburary 2013: CHERI box on the Internet

December 2012: Nested sandboxes and sandbox exception handling

January 2014: CheriBSD now uses CHERI Clang/ LLVM for key sandboxing components

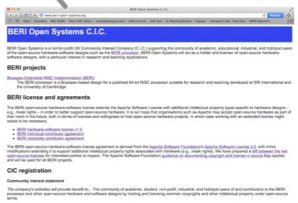
2010 2011 2012 2013 2014

November 2011: tPad tablet demo with Deimos



May 2012: DE4 tablet runs FreeBSD

April 2013: CheriCloud online for SRI/Cam users



December 2013: BERI Open Systems CIC created

Roles for theory, semantics, and tools

- Bluespec, a Haskell-based DSL ⟶ theorem proving/model checking
- ISA model in SRI PVS/SAL
  - Prove ISA expressiveness properties, generate test suite
- ISA model in Cambridge L3
  - Executable simulation that runs ISA, PIC test suites ('gold model')
  - Detects use of undefined hardware behavior by software
  - Analyze software interaction with multi-core memory model
- Extended the C language with strong memory/pointer integrity
- Automated tools to study security/performance/complexity tradeoffs
- Hardware tracing to introspection/analysis tools – zero SW overhead

Upshot: formal techniques used daily in our project.

FURTHER THOUGHTS

Unappealing collaboration: **systems**

We have this complex experimental system…

… that some PhD students threw together moments before the paper deadline…

… and it has some bugs (the same old ones that you've studied to death before).

We wonder if you could quickly find them all for us so we can get the code into production?

Unappealing collaboration: **theory**

We saw your bugs and wrote this tool…

… in Lambda Calculus, Haskell, **and** ML, …

… which we've already published the core paper on, and feel we are done with.

Could you use it so that we can claim in a journal article that it has been used on real systems?

More appealing: **systems**

We built and published this complex system.

In doing so discovered some new kinds of problems that we think are really interesting – and broadly applicable – but don't know how to model them or what to do with that knowledge.

Could we collaborate on exploring and evaluating solutions here – they will apply to our future systems projects?

More appealing: **theory**

We have some theoretical work that seems to describe what you are building, and has practical implications.

We have published the early work involving symbols you can't understand, and have tools written in Haskell – but you don't need to know Haskell to use them.

We are now looking to collaboratively validate them against a real systems corpus and understand their implications.

Trend: system introspection?

- Increasingly, processors, OSes, compilers, etc., offer explicit introspection features:
  - Systemic user-driven tracing and profiling tools that are 'always available' (e.g., DTrace)
  - Hardware performance counters (PMC), tracing, and virtualization techniques
  - Compiler frameworks for analysis and transformation, not just code generation (e.g., LLVM)
  - Techniques that compose all these three
- These are trends that the formal community can not only benefit from, but also contribute to the success of

Sustainability, adoptability, and transition as first-class goals

- Demonstrate easy wins – even if they seem boring
- Use the language of the programmer (or program)
- Avoid one-shot tools: integrate with suites (e.g., LLVM)
- Avoid dubious eval: OMG, our tool found (more) bugs in (just) the Linux kernel (again)
- Target design time – but also continuous integration
- Engage systems developers, not just systems researchers – but we can help you with contacts
- Plan to support transition activities in grant proposals
- Select open-source targets carefully; show generalizability

Conclusions

- My take on the evolving dialog between systems research and techniques/tools targeted at systems
- Taster of a systems research project that attempts to engage with techniques and tools
- Suggestions for engagement with the systems research community, areas of mutual interest
- http://www.cl.cam.ac.uk/research/security/ctsrd/