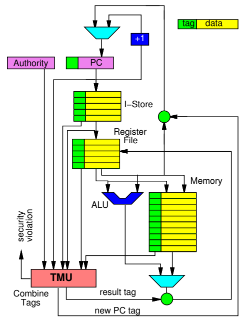
Principles, Meet Practice

An Early Retrospective on SAFE

^ (incomplete, personal...)

Benjamin C. Pierce
University of Pennsylvania

Principles in Practice Workshop
January, 2014



How do we make computers more **secure?**

Single protection boundary
(user/kernel)

expensive crossing between protection domains

Memory protection at process/page granularity

monolithic kernel design

Don’t start from here!

macro instead of micro-kernels

unchecked address arithmetic

manual memory management

"root" user

raw seething bits

“uni-typed” semantics
(pointer = integer = instruction)

Crash/
SAFE

Penn
University of Pennsylvania

BAE SYSTEMS

Principles

- Clean-slate co-design
- Push security into hardware
- Pervasive verification
- Defense in depth

Clean-slate design

Clean-slate design: Principles

- No requirement to be compatible with any legacy standards or artifacts
- Build a completely new system stack, oriented around security from bottom to top
  - New ISA (the *SAFE Machine*)
  - New hardware implementation (on FPGAs)
  - New operating system / runtime services
  - New programming languages
    - Breeze (for applications)
    - Tempest (for low-level system programming)
  - New applications
    - Focused on web services
- Intensive co-design across traditional system layers

Clean-slate design: Practice

- Lots of fun!
- Way to explore new parts of the design space
  - Not just relax one dimension at a time
  - Plant a flag someplace completely different
- Ambitious co-design ⇒ many meetings (some pretty interesting :-)
- Main challenge: Evaluation / validation
  - Need some basis for choosing which way to move
  - General principles helpful up to a point
  - After that, need *pull* from applications
- Result: Parts with clearer vision at the beginning made most progress by the end
  - e.g., hardware security mechanisms

Security in Hardware

Security in Hardware: Principles

- Why hardware?
  - Use abundant silicon to improve security (not just more cores)
  - Make higher-level security mechanisms available to low-level code
  - Hardware more trustworthy than software? (Immutable...)
- Mechanisms:
  - Hardware typing
  - Memory safety
  - Metadata tags and fast tag checking / propagation

    ⇒ fine-grained dynamic information-flow control (IFC)
  - Other innovations: linear capabilities, lightweight transactions, stream support, fast context switching, ...

Security in Hardware: Practice

- Hardware typing :-)
  - integer <> pointer <> instruction <> closure
- Memory safety :-)
  - “low-fat pointers” [CCS 2013]
  - efficient encoding of base, bounds, and offset into 64-bit words
    - logarithmic encoding uses few bits
    - surprisingly little fragmentation
  - bounds checked on *every* pointer access
- IFC... :-|

Information-Flow Control

IFC: Principles

- Apply well-established theory of IFC from PL community

IFC: Practice

- Many significant challenges remaining in “well-developed theory”
- $\Rightarrow$ We became IFC researchers!
- Good progress on some issues...

- Mapping the jungle of label models [CSF 2013]

$$\begin{tikzcd}
\text{Endorsement} & & \sqsubseteq = \sqsupseteq & \sqsubseteq = \subseteq & & \text{Distrust} \\
\mathbf{E} \arrow[r] & \mathbf{CE} \arrow[r] \arrow[d, dotted, leftrightarrow] & \mathbf{UE} \arrow[r, leftrightarrow] \arrow[d, dotted, leftrightarrow] & \mathbf{UD} \arrow[d, dotted, leftrightarrow] & \mathbf{CD} \arrow[l] \arrow[d, dotted, leftrightarrow, "L^{def} = \top"', "L^{def} = \bot"] & \\
 & \mathbf{CR} \arrow[r] & \mathbf{UR} \arrow[r, leftrightarrow] & \mathbf{UT} & \mathbf{CT} \arrow[l] & \mathbf{T} \arrow[l] \\
\text{Readers} & & & & & \text{Taint}
\end{tikzcd}$$

$$\begin{tikzcd}
\mathbf{CR^0} \arrow[r] \arrow[dd, bend right] & \mathbf{CR} \arrow[d, bend right, "\times" marking, "?"'] \arrow[drr, leftrightarrow, "\times" marking] & & \mathbf{T} \arrow[dll, bend right] \arrow[d, bend right] & \mathbf{T^0} \arrow[l] \arrow[dl, bend left] \\
 & \mathbf{DC_S} \arrow[u, bend right, "\times" marking] \arrow[urr, "\times" marking] \arrow[rr, shift left, "\times" marking] & & \mathbf{DLM_S} \arrow[u, bend right, "\times" marking] \arrow[ll, shift left, "\times" marking, "?"] & \\
 & \mathbf{DC_S^0} \arrow[u] \arrow[rr, leftrightarrow] & & \mathbf{DLM_S^0} \arrow[u] &
\end{tikzcd}$$

Immunity from poison pills [Oakland 2013]

IFC labels as a denial-of-service vector...

```
fun process_max (x,y) = if x <= y then y else x

fun rec max_server_loop () =
  send out (process_max (recv in));
  max_server_loop ()
```

Good discussions on other issues :-)

IFC: Major challenges

give example for concurrency?
emphasize that there is a lot of great work going on in all these areas -- just not finished

- Weak attack model
  - Assumes attacker cannot observe nontermination, timing, power, cache effects, disk head movement, ...
  - Pragmatic solution in SAFE: Supplement with access control
- Doesn't play well with concurrency

Possible solution: "never lower PC label" *à la* LIO

- Declassification a perennial problem

```
P0:                 P1:                 P3:
while H do          while ¬H do         while ¬L0∧¬L1 do
  skip;               skip;               skip;
L0 := true          L1 := true          L := L0;
                                        ...
```

Pervasive Verification

Pervasive Verification: Principles

- Keep design clean and simple
- Formally *specify* major interfaces
  - Hardware ISA
  - IFC abstract machine
  - High-level programming language
- Formally *verify* critical software components
  - Rule cache management
  - Scheduling, process management, and IPC
  - Garbage collection
  - (Maybe compiler)

Pervasive Verification: Practice

- Hard to convey difficulty of formal verification to people who haven't done it
- Hard to do formal verification on a fast-moving design
- Bad interaction with "defense in depth"...

warnings like "100x verification cost" are j
words to the hardware and OS people

Defense in Depth

Defense-in-Depth: Principles

- Increase attacker work factor by forcing them to overcome multiple lines of defense
- Avoid brittleness of "strong single layer"
    - faulty devices
    - bit flips
    - unverified components
    - proof rot

Defense-in-Depth: Practice

- Concepts like “degree of redundancy” and “attacker work factor” very difficult to measure
- “n+1 mechanisms better than n”
- More mechanism ⇒ more complex design
- Difficult to combine with pervasive verification

Not-So-Pervasive Verification

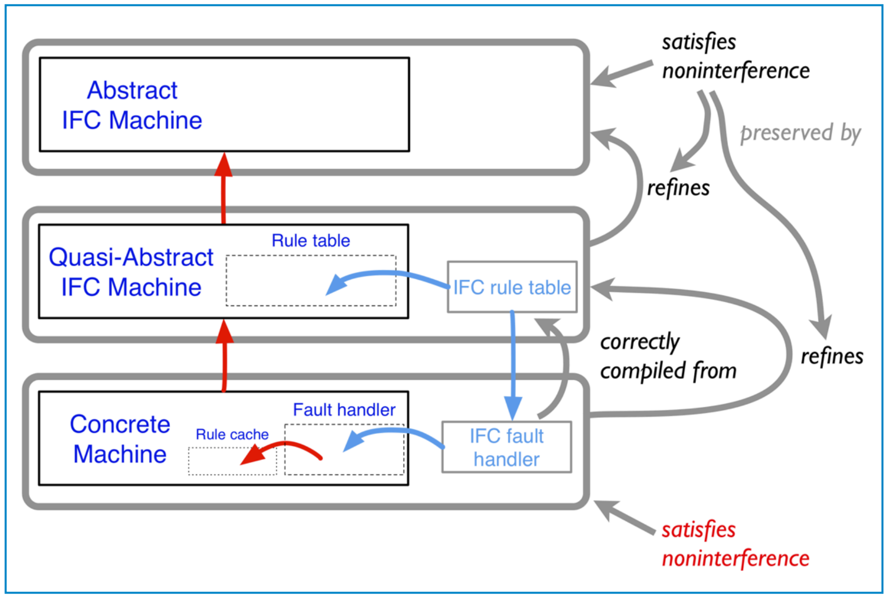
Modest Verification: Principles

- Specify critical interfaces
- Verify key properties of simplified models

Modest Verification: Practice

- Small proofs of noninterference for *many* IFC calculi and abstract machines

- Biggest achievement so far:
  - Formalization of core tag cache hardware, software handler, IFC abstract machine, and proof of correctness [POPL 2014]



(but effective!)

Another Modest Idea...

Random Testing
for Security Properties

Random Testing: Principles

- Specify critical interfaces
- Use property-based random testing (*à la* QuickCheck) to debug key properties such as noninterference

talk about ICFP experiment and the methodology it gives rise to...

Random Testing: Practice

- Question: Can we effectively test security properties (noninterference) by generating random inputs (pairs of low-equivalent machine states)?
- Progress so far [ICFP 2013]:
  - Simple IFC abstract machine, known correct
  - Manually inject IFC bugs
  - Measure how quickly they can be found by random testing

If two machine states are low-equivalent then the <u>next low states</u> reached by both

Prope

If two machine states are <u>related</u> then the next steps of the two machines are <u>similar</u> and the <u>next states</u> (whether low or high) are again related (cf. unwinding conditions)

nc

Low-lo ckstep

If two initial machine states are low-equivalent and both machines terminate in low states, then these states are also low-equivalent

Single-step

Methodology for testing NI

- (Skip end-to-end property: too slow)
- Start with low-lockstep property (LLNI)
    - Generic
    - Finds bugs reasonably fast
- When LLNI stops finding bugs, try to find single-step invariant
    - Requires considerable insight
    - Random testing can help debug it!
- Test single-step property (SSNI) until no more counterexamples are found
    - Gives confidence in both invariant and artifact
- (Optional) Use invariant to prove SSNI
    - Conclude that original NI property holds

(How well) does it scale up?

- We're applying it to significantly larger fragments of SAFE
- Have not run out of steam yet :-)
- Main challenge: Generating well-distributed test data

Where (Else) We’re Going

Where we are...

- Working hardware
  - Running on FPGAs, implemented in Bluespec
- Core OS / RTS components
  - Simple scheduler, GC, tag hardware management
- Low-level systems programming language: Tempest
  - Compiler working
  - Growing body of systems code
  - One simple "end to end" application
  - (talk to Jesse Tov)
- High-level applications language: Breeze
  - Running interpreter, but no compiler
  - A few larger demo applications

Near-Term Goals

- SAFE processor
  - Pipelining
  - Energy and area optimization
- Low-level software
  - Better tag management, more interesting label model
- High-level software
  - More interesting demo application(s?)
- Formalization
  - Full (or at least full-er) specification of ISA
  - Validated by random testing

An Interesting Spin-Off

- Transplant (just) tag-management hardware to a stock processor
- Explore other "micro-policies" that can be implemented using hardware tags
  - memory safety
  - CFI
  - linearity
  - dynamic typing
  - race detection
  - ...

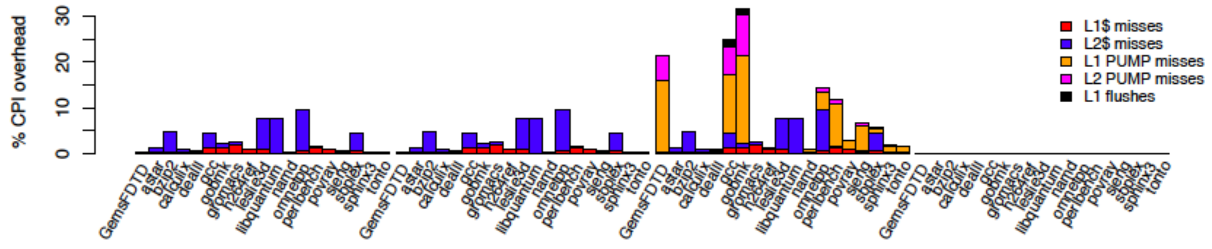


Figure 12: Overhead CPI (types, libtaint, complete-cfi, none) with a 1200 cyc miss handler latency

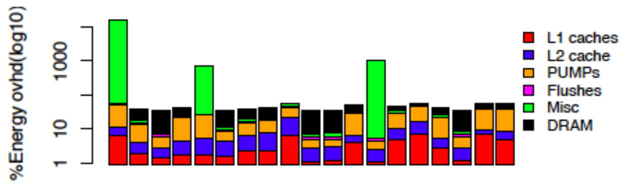
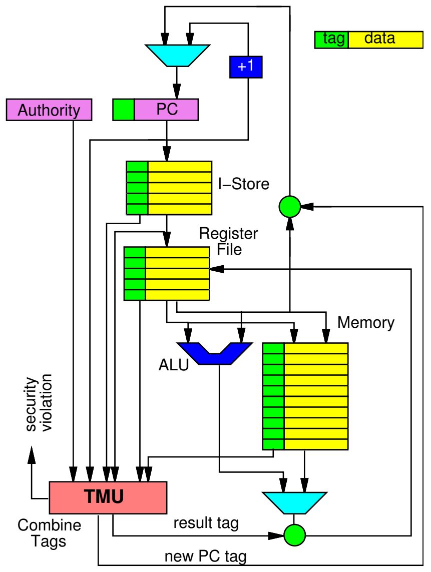


Figure 21: Overhead energy for memory safety policy with a 1200 miss handler latency (gcc, perlbench missing)

Thank you!


tag
data
+1
Authority
PC
I–Store
Register File
Memory
ALU
security violation
TMU
Combine Tags
result tag
new PC tag

Shown: Sumit Ray, Howard Reubenstein, Andrew Sutherland, Tom Knight, Olin Shivers, Benjamin Pierce, Ben Karel, Benoit Montagu, Jonathan Smith, Cătălin Hriţcu, Randy Pollack, André DeHon, Gregory Malecha, Basil Krikeles, Greg Sullivan, Greg Frazier, Tim Anderson, Bryan Loyall

Not shown: Greg Morrisett, Peter Trei, David Wittenberg, Amanda Strnad, Justin Slepak, David Darais, Robin Morisset, Chris White, Anna Gommerstadt, Marty Fahey, Tom Hawkins, Karl Fischer, Hillary Holloway, Andrew Kaluzniacki, Michael Greenberg, Andrew Tolmach, Antal Spector-Zabuski, Leonidas Lampropoulos, Tyler Brown, Ian Nightingale, Udit Dhawan, Albert Kwon, Jesse Tov, Arthur Azevedo de Amorim, Nathan Collins, Arun Thomas, Shannon Spires, Mitch Wand, ...