CRASH/SAFE: Clean-slate Co-design of a Secure Host Architecture

Cătălin Hrițcu
CRASH/SAFE project

• Academic partners (16):
  – University of Pennsylvania (11)
  – Harvard University (4)
  – Northeastern University (1)

• Industrial partners (24):
  – BAE systems (21) + Clozure (3)

• Funded by DARPA
  – Clean-Slate Design of Resilient, Adaptive, Secure Hosts
Clean-slate co-design of net host

Primary goal: design and implement a significantly more secure architecture, without backwards compatibility concerns

Secondary goal: verify that it’s secure (whatever that means)

New stack:
- language
- runtime
- hardware
Grandpa! Why are computers so insecure?

Transistors were precious back then, my boy ...
Hardware is now abundant
Formal methods are now better

• random testing
  – QuickCheck [Claessen & Hughes, ICFP’00]

• automatic theorem provers & SMT solvers

• machine-checked proofs
  – CompCert [Leroy, POPL’06]
  – seL4 [Klein et al, SOSP’09]
  – CertiCrypt [Barthe et al., POPL’09]
  – ZKCrypt [Almeida et al, CCS’12]
Security is much more important
Time for a redesign!

language
runtime
hardware

Information flow
Access control
Type safety
Memory safety

Verification

fine-grained
Language (Breeze)

- testing ground for ideas we port to lower levels
- **type and memory safe** high-level language
  - dynamically typed + dynamically-checked contracts
- **functional core** ($\lambda$) + state(!) + concurrency ($\Pi$)
  - message-passing communication (channels)
- built-in **fine-grained protection mechanisms**:  
  - values are attached **security labels** (e.g. public/secret)  
  - dynamic information flow control (IFC)  
  - discretionary access control (clearance)
Runtime system

• manages:
  – **time**: scheduler
  – **memory**: allocator, garbage collector
  – **communication and resources**: channels
  – **protection**: principals, authorities, and tags (PAT)

• small trusted computing base

• comparimentalized
  – a dozen mutually distrustful servers (least privilege)
Hardware

• all instructions have well-defined semantics
  – abstractions strictly enforced
• low-fat pointers
  – can’t access/write out of frame bounds
• dynamic types
  – can’t turn ints into pointers (unforgeable capabilities)
• authority + closures/gates (λ) + protected stack
  – fine-grained privilege separation
• programmable tag management unit (TMU)
Tag management

• **every word tagged** with arbitrary pointer
  – only runtime system interprets these pointers
• **on each instruction** TMU looks up tags of operands in a **hardware rule cache**
  – found → rule provides tags on results (no delay)
  – not found → trap to software (PAT server)
• **access control + IFC** enforced at lowest level
Project status (2/4 years)

• **language:**
  – stable interpreter, work-in-progress compiler
  – applications: e.g. web server running wiki
  – Coq proofs for various core calculi (non-interference)

• **runtime:**
  – detailed design, some prototype servers
  – work on testing+/verifying simplified PAT server

• **hardware:**
  – full-fledged un-optimized FPGA prototype
  – novel instruction set, simulators, debugger, ...
  – executable instruction set semantics in Coq
Research outcomes

• position papers / talks
  – PLOS’11: Preliminary Design of the SAFE Platform
  – PLPV’12: Verification Challenges of Pervasive Information Flow
  – AHNS’12: Hardware Support for Safety Interlocks and Introspection

• language-based security
  – under review at Oakland: All Your IFCException Are Belong To Us
  – likely CSF submission: A Theory of IFC Labels

• hardware mechanisms
  – FPGA’13: Area-Efficient Near-Associative Memories on FPGAs
  – under review at Oakland: Low-Fat Pointers
MY RESEARCH
Pre-SAFE work

• crypto protocols
  – tools aiding design, analysis, and implementation
  – more expressive type systems (e.g. first one for ZK) [CCS’08, CSF’09, TOSCA’11, PhD thesis]
  – remote electronic voting [CSF’08]
  – code generation (Expi2Java) [NFM’12]

• data processing language (Microsoft “M”)
  – semantic subtyping [ICFP’10, JFP’12]
  – verification condition generation [CPP’11]
SAFE work

All Your IFCException Are Belong To Us

Robust Exception Handling for Sound Fine-Grained Dynamic Dynamic IFC

joint work with Michael Greenberg, Ben Karel, Benjamin Pierce, and Greg Morrisett
Exception handling

• we wanted all Breeze errors to be recoverable
  – including IFC violations
• however, existing work assumes errors are fatal
  – makes some things easier ... at the expense of others
    +secrecy     +integrity     –availability
Problem #1: IFC exceptions reveal information about labels

- labels are themselves information channels
- get soundness by preventing secrets from leaking either *into* or *out of* label channel
Problem #1: IFC exceptions reveal information about labels

- labels are themselves information channels
- get soundness by preventing secrets from leaking either *into* or *out of* label channel

\[
\text{if } h@\text{secret} \text{ then } ()@\text{secret} \text{ else } ()@\text{top-secret}
\]
Problem #1: IFC exceptions reveal information about labels

- labels are themselves information channels
- get soundness by preventing secrets from leaking either into or out of label channel

allow labels to depend on secrets

labels must be hidden

labels must be hidden!
(and we don’t want that)

**if h@secret then ()@secret else ()@top-secret**
Problem #1: IFC exceptions reveal information about labels

- labels are themselves information channels
- get soundness by preventing secrets from leaking either into or out of label channel

enforce that labels don’t depend on secrets

labels and IFC errors can be observed

\[
\text{if } h@\text{secret} \text{ then } ()@\text{secret} \text{ else } ()@\text{top-secret}
\]
Problem #1: IFC exceptions reveal information about labels

- labels are themselves information channels
- get soundness by preventing secrets from leaking either into or out of label channel

enforce that labels don’t depend on secrets

Solution #1: brackets

top-secret[if h@secret then ()@secret else ()@top-secret]
Problem #2: exceptions destroy control flow join points

• ending brackets need to be control flow join points
  – try
    
    ```
    let _ = secret[if h then throw Ex] in
    false
    catch Ex => true
    ```

• brackets need to delay all exceptions!
  – secret[if true@secret then throw Ex] => “(Error Ex)@secret”
  – secret[if false@secret then throw Ex] => “(Success ())@secret”

• similarly for failed brackets
  – secret[42@top-secret] => “(Error EBracket)@secret”
Solution #2: Delayed exceptions

• delayed exceptions unavoidable
  – still have a choice how to propagate them

• we studied two alternatives for error handling:
  1. mix active and delayed exceptions \((\lambda[^1]_{\text{throw}})\)
  2. only delayed exceptions \((\lambda[^1]_{\text{NaV}})\)

  • delayed exception = not-a-value (NaV)
  • NaVs are first-class replacement for values
  • NaVs propagated solely via data flow
  • NaVs are labeled and pervasive
  • more radical solution; implemented in Breeze
What’s in a NaV?

• error message
  – `EDivisionByZero ("can’t divide %1 by 0", 42)

• stack trace
  – pinpoints error origin
    (not the billion-dollar mistake)

• propagation trace
  – how did the error make it here?
Formal results

- proved termination-insensitive **non-interference** in Coq for $\lambda[]$, $\lambda[\text{Nav}]$, and $\lambda[\text{throw}]$
  - for $\lambda[\text{Nav}]$ even with all debugging aids; **error-sensitive**
- in our setting NaVs and catchable exceptions have **equivalent expressive power**
  - translations validated by QuickChecking extracted code
Summary for IFC exceptions

- reliable error handling possible even for sound fine-grained dynamic IFC systems
- we study two mechanisms ($\lambda^{\text{NAV}}$ and $\lambda^{\text{throw}}$)
  - all errors recoverable, even IFC violations
  - key ingredients: sound public labels (brackets)
    + delayed exceptions
  - quite radical design (not backwards compatible!)
Ongoing SAFE work

- **testing+/verifying PAT server**
  - with Benjamin Pierce, Dimitrios Vytionitis, John Hughes, Andrew Tolmach, Delphine Demange, ...

- **protecting data integrity with signature labels**
  - on the meaning(lessness) of IFC endorsement
  - reviving trademarks [Moris ‘73]
  - beyond data abstraction (dynamic sealing): caching contracts

- **implementing Breeze labels cryptographically**
  - potential collaboration with Deian Stefan / LIO team (DC labels)
Testing+/verifying PAT server

abstract machine

security (non-interference)

already done (most of) this for extremely simplified machines (6 instructions)

concrete machine + PAT server

correctness of PAT server implementation

future work: scale this up as much as possible

future work: scale this up to the real thing

random testing

Coq proving

extraction

future work:

challenge:

challenges:
- smart program generation
- counterexample shrinking

very complex invariants
Some post-SAFE ideas ...

• software-hardware co-design for security-critical high-assurance devices
  – voting machines, automobile subsystems (e.g. driver assistance), medical devices (e.g. pacemakers, insulin pumps), crypto boxes (e.g. TPMs, HSMs, etc.)
    • limited/fixed functionality
    • security more important than backwards compatibility
  – existing devices often blatantly vulnerable
  – goal #1: make security analysis part of design process
  – goal #2: verify security of actual implementations

• fine-grained access control and integrity protection for mobile devices
THE END