Verification Challenges of Pervasive Information Flow

Benjamin C. Pierce
University of Pennsylvania

Programming Languages Meets Program Verification
PLPV, January 2012
COMPUTER systems are insecure!
Major contributing factor:

Legacy design decisions, now deeply embedded in HW/SW ecosystem
What’s changed?

1. Huge increases in hardware resources
   ➔ Reconsider traditional sources of complexity
   ➔ Spend hardware to increase security

2. Huge advances in formal methods
   ➔ Machine-checked correctness proofs for significant programs becoming practical
Clean-slate design of Resilient, Adaptive, Secure Hosts

SAFE
Shown: Sumit Ray, Howard Reubenstein, Andrew Sutherland, Tom Knight, Olin Shivers, Benjamin Pierce, Ben Karel, Benoit Montagu, Jonathan Smith, Catalin Hritcu, 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
Outline

1. Overview of CRASH/SAFE
2. Verification challenges

Many challenges!
Questions welcome!

(any time)
Vision
• Clean-slate redesign of the HW / OS / PL stack

• Support at all levels for
  • Memory safety
  • Strong dynamic typing
  • Information flow and access control

• Co-design for verifiability
Low-level view
Fat pointers

Every pointer includes base and bounds:

<table>
<thead>
<tr>
<th>Base address</th>
<th>Offset</th>
<th>Bound</th>
</tr>
</thead>
</table>

(Logarithmic encoding scheme ➔ compact representation)

[Brown et al, 2000]
Strong typing

Every data value is annotated with its atomic group

int64
double
pointer
instruction
...

...
Rich tagging

<table>
<thead>
<tr>
<th>59 bits</th>
<th>5 bits</th>
<th>64 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tag</strong></td>
<td><strong>Atomic Group</strong></td>
<td><strong>Payload</strong></td>
</tr>
</tbody>
</table>

uninterpreted at HW level

int64, double, pointer, instruction, etc.

data
Tag interpretation

- “This pointer can only by followed by the scheduler”
- “This instruction can only be executed by the memory allocator”
- “This integer can only be read by user-defined principal P”
- “The document at the other end of this pointer has been endorsed by principal P”
- “This string came directly off the network and has not been sanitized yet”
- etc.
Processor

PC

+1

I-Store

Register File

ALU

Memory
SAFE Processor

Tag Management Unit

PC

ALU

Memory

I−Store

Register File

Authority

Tag Management Unit

Combine Tags

Result tag

New PC tag

Tag data

Security violation

TMU
(Eliding PC tag...)

TMU
(Tag Management Unit)

Rule Cache

Instruction
Operands
Authority

Result

OK

???

trap to software handler
TMU
(Tag Management Unit)

Rule Cache

Instruction
anybody Add

Operands
"only P" 5
anybody 7

Authority
P

Result
"only P" 12

OK

???

trap to software handler
TMU
(Tag Management Unit)

Rule Cache

anybody | Add

Operands

"only P" | 4 1
"only P" | 1

Authority

P

Result

OK

???

trap to software handler

TRAP!
TMU (Tag Management Unit)

Rule Cache

anybody: Add

Operands:
- "only P": 41
- "only P": 1

Authority: P

Result:
- "only P": 42

OK

???

trap to software handler
High-level view
Breeze

A high-level, security-oriented programming language

Summary:

• ML-like (CBV, mostly functional)
• Channel-based communication
  • à la CML / Pict
• Dynamically typed
  • maybe statically, later
  • for now: rich contract system
• Information flow and access control
Principals

- Breeze execution state include a set of principals
- New principals can be created dynamically
Authority

- Creating a principal also creates an **authority**, representing the capability to act as that principal.
- Abstract machine maintains a **current authority**
  - and offers primitives for raising authority (adding known capability to current authority) and dropping authority.
- Attempting an operation not permitted by the current authority aborts the running thread.
Labels

• Every value comes with a label describing its security policy
• Labels form a lattice
Information Flow

Labels are propagated during evaluation

\[ 40\@P + 2\@Q \Downarrow 42\@(P\&Q) \]

PC label tracks implicit flows

\[
\text{if secret-belonging-to-P} \\
\text{then } 5\@\bot \text{ else } 6\@\bot \Downarrow 5\@P
\]
Verifying the HW / SW Stack
System structure

- **BREEZE**: source-level operational semantics
  *(more layers for compiler passes)*
- **CW**: TM plus inter-process communication
- **TM**: MM plus tag management
- **MM**: SCHED plus memory management
- **SCHED**: ISA plus scheduler
- **ISA**: bare hardware (in Coq)
- **ISA-BS**: bare hardware (in BlueSpec)

A stack of abstract machines...
Relating Abstract Machines
An abstract machine

machine configurations

external event traces

step relation

\[ M \xrightarrow{T} M' \]
Nondeterminism

Specification doesn’t want to nail down some aspects of machine’s behavior

• “By how many cycles does the countdown timer decrease when each instruction is executed...?”

Loose specification permits any outcome

• “An instruction can take any number of cycles”

However...
Nondeterminism makes reasoning hard!
Oracles

A nice trick:

\[ M = MO \times MS \]

Oracle captures nondeterminism

- “Each instruction takes some particular number of cycles in a given run, but the step function doesn’t know how many; it consults the given oracle to find out.”

Step relation now becomes a function

\[ T \quad (MO, MS) \quad \rightarrow \quad (MO', MS') \]
Relating machines

Given a **concrete** machine C and an **abstract** machine A, suppose we want to argue that “C is a correct implementation of A.”
A is implemented by C if there is some correspondence relation (written ~) between abstract and concrete machine configurations such that

Wait... any relation ~?

Need to require that ~ be “total”...

First try...
A is implemented by C if there is some correspondence relation ~ such that

1. \( \forall A \exists C \text{ with } A \sim C \)
2. this diagram commutes:

\[
\begin{array}{ccc}
A & \xrightarrow{T} & A' \\
\sim & & \sim \\
\sim & & \sim \\
C & \xrightarrow{T} & C'
\end{array}
\]

Wait... is this the right order of quantifiers for the oracles?

No: The abstract oracle’s choices should depend on the concrete one’s!
A is implemented by C if there is some correspondence relation ~ such that

1. \( \forall AS \ni CS \) such that \( \forall CO \ni AO \) with \((AO,AS) \sim (CO,CS)\)

2. this diagram commutes:

\[
\begin{array}{c}
(AO,AS) \xrightarrow{T} (AO',AS') \\
\sim \\
(CO,CS) \xrightarrow{T} (CO',CS')
\end{array}
\]

But we can streamline it a little...
A is implemented by C if there is some relation \( \sim \) between abstract and concrete states and a total function \( O : (CO,CS) \to AO \) such that

1. \( \forall AS \exists CS \text{ such that } AS \sim CS \)
2. this diagram commutes:

\[
\begin{array}{c}
\text{(AO,AS)} \xrightarrow{T} \text{(AO′,AS′)} \\
\downarrow \sim \quad \quad \quad \downarrow \sim \\
\text{(CO,CS)} \xrightarrow{T} \text{*(CO′,CS′)}
\end{array}
\]

(What’s this called?)
Example

(Suppose we were specifying the TM running directly on the bare ISA...)

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BREEZE</td>
<td>source-level operational semantics</td>
</tr>
<tr>
<td>CW</td>
<td>TM plus inter-process communication</td>
</tr>
<tr>
<td>TM</td>
<td>MM plus tag management</td>
</tr>
<tr>
<td>MM</td>
<td>SCHED plus memory management</td>
</tr>
<tr>
<td>SCHED</td>
<td>ISA plus scheduler</td>
</tr>
<tr>
<td>ISA</td>
<td>bare hardware (in Coq)</td>
</tr>
<tr>
<td>ISA-BS</td>
<td>bare hardware (in BlueSpec)</td>
</tr>
</tbody>
</table>
ISA Spec

machine state:
- memory, registers
- countdown timer (cycle counter)
- hardware TMU rule cache

oracle:
- how much does timer change on each instruction

step function:
if timer = 0, then save PC and fault to interrupt handler entry point,
else if hardware TMU cache has a rule allowing next instruction
then ask oracle how much to decrement timer
and execute instruction
else fault to TMU handler entry point
Tag Manager Spec

machine state:
- memory, registers, countdown timer as before
- no hardware TMU rule cache
- security state: set of principals, with associated lattice of labels, ...

oracle:
- same as ISA

step function:
if timer = 0 then fault to interrupt handler,
else if next instruction is “call allocate-principal function”, then
  • allocate a principle (in one step)
  • and put its name in result register
else ... (similarly for other TM entry points) ...
else if security state says next instruction is legal
  then execute it, using security state to determine tags on results
else halt machine
Metatheorems
Beyond non-interference?

Vanilla non-interference is not enough...

- concurrent threads weaken it
- declassification breaks it

(...though better than nothing!)
Possible approaches

Methodological:

• Minimize number of audit points requiring ad hoc inspection:
  • e.g., declassification, process creation

• Make user-level code as deterministic as possible

Structural:

• Could user code be completely determinized??
  • cf. Determinator [Ford et al.]
Poison Pills

How to prevent one component from “poisoning” another by sending it an inappropriately secret value...
One approach: Public labels

Fundamental issue:
- In standard formulations of dynamic information flow, the security label on a piece of data can itself carry secret information

Idea:
- Rearrange primitives so that security labels can always be public
- Now, “victim” of a poison pill can look at the label and decide whether it is willing to raise its security level enough to look at the contents
Application-level policies

How do we (formally) connect our language-level security primitives to user-level security policies?
One approach: Policy weaving

Idea [Harris, Farley, Jha, Reps 2011]

- Specify policy separate from application code
- Automatically “weave” them together

Side benefit:

- Might work at ConcreteWare level, reducing the urgency of verifying the compiler!
What is the attack model?
Clear part...

Attacker does **not** have physical access to the machine (either directly or via the supply chain)

Attacker **does** get to run their code on the machine, and it can interact with ours

• e.g., plug-ins
Clear implication

We need to be careful about where secrets can flow on the machine, not just at its external interface (the network)

• If we allow attacker code to see secrets, it can easily exfiltrate them using covert channels
  • No practical way to prevent this!

• ➜ Need access control, not just information-flow tracking
Real attacks often involve sending bad inputs that confuse some trusted component and cause it to behave badly
• e.g., buffer overflow attacks

We *hope* we’ve prevented many of the common cases, but there is no way to be certain.

→ least-privilege design
Challenge!

What is “least privilege,” formally?
Possible definitions:

1. Given a **fixed** set of software components, how do we assign them privileges in a minimal fashion?

2. Given two **alternative designs** satisfying the same specification, which one is “more least privilege”?
Finishing up...
Status

- Breeze v0 design, interpreter, toy apps
- Machine-checked proofs of a few metatheorems for core calculi
- Non-pipelined implementation of most instructions running on FPGA
- Toy versions of key services (allocator, scheduler, tag manager)
- Formal ISA spec under construction now
Related work

Verified operating systems
- Gypsy [1989]
- VeriSoft [2008]
- seL4 [2009]
- Verve [2010]

Verified compilers and runtime systems
- Flint [2008]
- CompCert [2006, 2009] and friends

Language-based operating systems
- Cedar/Mesa, Smalltalk, lisp machine, ...
- SPIN
- House/HASP
- Singularity
- Java OSs
- ...
Join us!

We have a lot of exciting projects for PhD students and postdocs...