CRASH/SAFE

Benjamin C. Pierce

March 11, 2011

Present-day computing platforms are distressingly insecure!

BAE SYSTEMS

Penn University of Pennsylvania

One culprit: legacy requirements

complex instruction sets
complex, monolithic operating systems
insecure, low-level programming languages

Patch?

Reboot!

CRASH

Clean-Slate Design of Resilient, Adaptive, Secure Hosts

SAFE Team

BAE SYSTEMS

BAE Systems


Bryan Loyall


Greg Sullivan


Howard Reubenstein


Basil Krikeles


Greg Frazier


Jothy Rosenberg

Also: Tim Anderson, Chris White, …

University of Pennsylvania


André DeHon


Benjamin Pierce


Jonathan Smith

Also: Ben Karel, Benoit Montagu

Consulting


Tom Knight

Northeastern


Olin Shivers

Harvard


Greg Morrisett

Also: Gregory Malecha

Core Principles

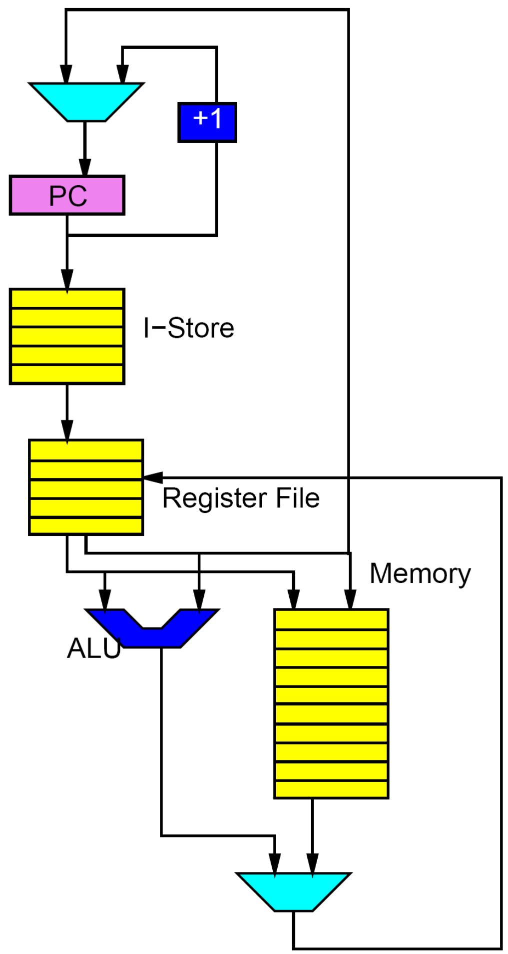
BAE SYSTEMS

- ***Fine-grained compartmentalization:*** supported by hardware, with runtime intents & security interlocks, without compromising performance
  - Tagged data for compartmentalization and intent
  - Programmable Rulesets
  - Hardware Tag Management Unit for complete mediation on cycle-by-cycle basis. Checking performed in parallel to mainline for high performance.
- ***Radical Co-design for pervasive verification:*** define clean semantics and omit complicating features to make verification tractable
- ***Prevention-in-Depth:*** radical decomposition of systems into mutually suspicious components with separated privileges.

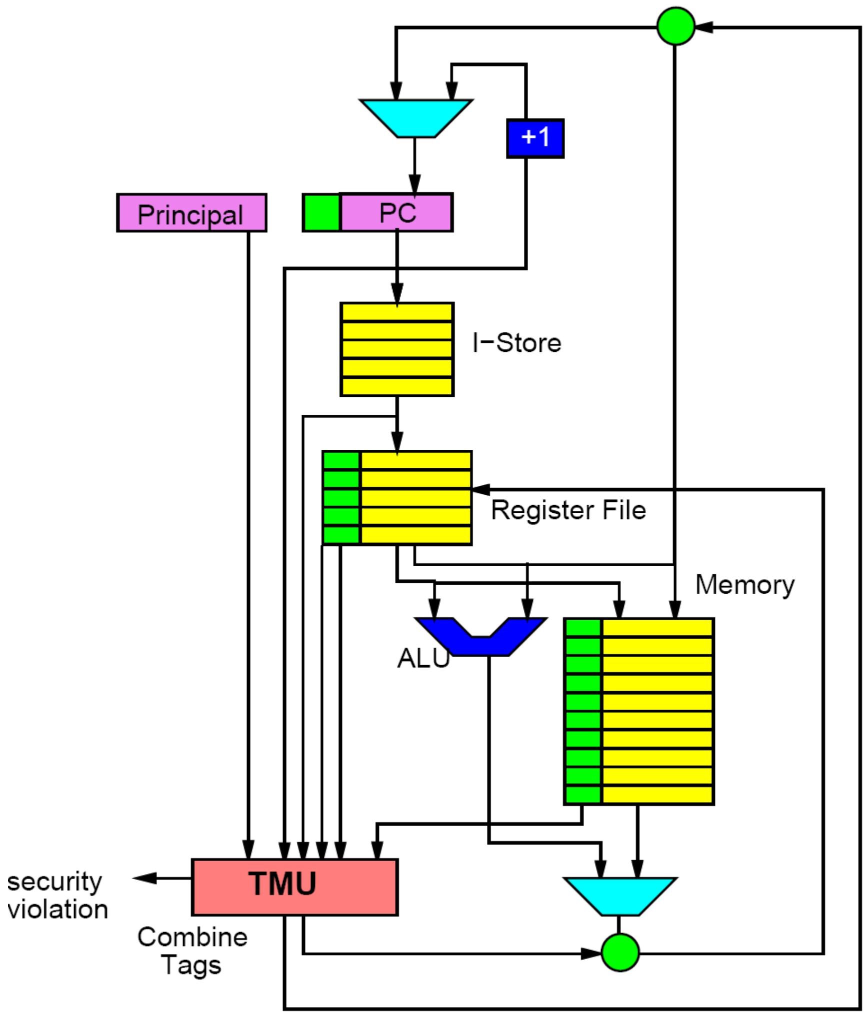
Topic Areas

1. Tagged Processor Architectures
2. "Zero-Kernel" Operating Systems
    1. Strong compartmentalization
    2. Mutual suspicion
3. Programming Languages
    1. Tempest – low-level systems programming (C-like)
    2. Breeze – high-level applications programming (ML/ Haskell-like)
4. System-wide application of Formal Methods
    1. Design for verifiability

HARDWARE

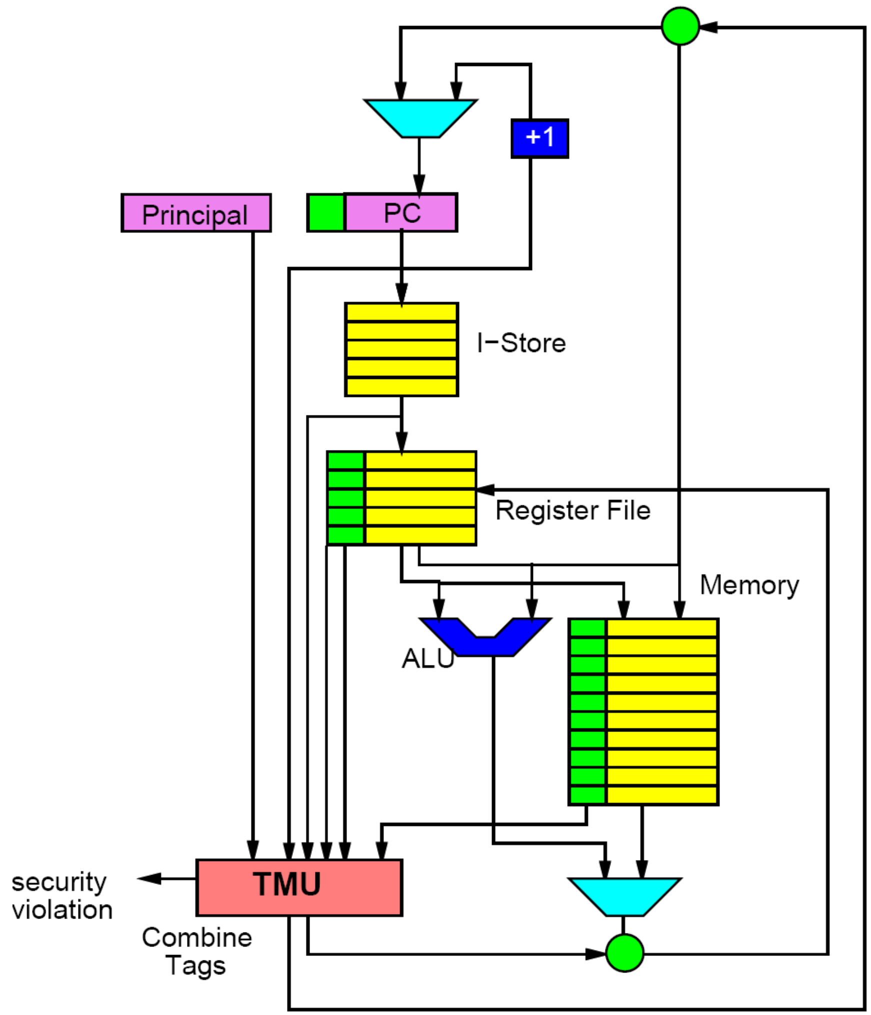


+1
PC
I−Store
Register File
Memory
ALU



+1
Principal
PC
I−Store
Register File
Memory
ALU
security violation
TMU
Combine Tags

- Process tags in parallel with datapath
  - No impact on cycle time
- Leverage existing speculation/in-order exception and retirement hardware
- Implement with fast, small Tag Management Unit
  - Similar in size/complexity to TLB



A taste of μBreeze

μBreeze overview

BAE SYSTEMS

- Straw-man design – just to gain experience
- An untyped, CBV lambda-calculus with
  - information-flow tracking *a la* JIF/JFlow
    - every value is tagged with a *label* specifying who may read (eliminate) it
  - communication channels (elided for today) and threads (soon)

Syntax

e	$::=$			expressions
	$\mid$	$const$		constant
	$\mid$	x		variable
	$\mid$	$\lambda x.e$	bind x in e	abstraction
	$\mid$	$e_1\ e_2$		application
	$\mid$	(e_1, e_2)		pairing
	$\mid$	$e.1$		first projection
	$\mid$	$e.2$		second projection
	$\mid$	$e \vee L$		raise label
	$\mid$	$e \wedge L$		lower label
	$\mid$	$e < L$		check label
	$\mid$	auth A in e		change authorization
	$\mid$	block L in e		locally join pc label

Evaluation

current PC label

current environment (binding variables to their values)

current authority

result (atom = value+label)

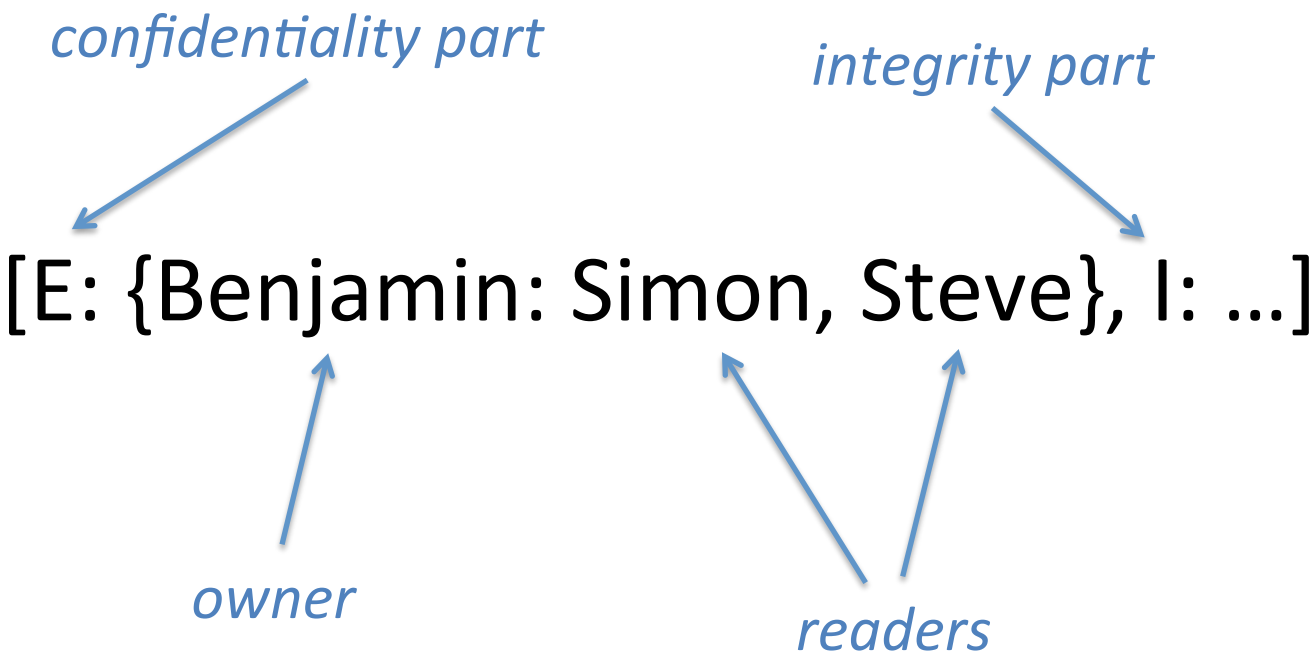
$$A, pc, \rho \vdash \sigma, e \Downarrow \sigma', a$$

current store

new store

expression being evaluated

Labels



confidentiality part
integrity part
[E: {Benjamin: Simon, Steve}, I: ...]
owner
readers

Labels

Decentralized Label Model (Liskov/Myers)
Multiple owners, *each* asserting a constraint on who may read

owner *readers*

[E: {Benjamin: Simon, Steve;
Stephanie: Simon, John, Mary}, I: …]

owner *readers*

Authority

A	$::=$		authority
	$\mid$	p	specified principal

Values and Atoms

BAE SYSTEMS

v	$::=$		value
	$\mid$	$const$	constant
	$\mid$	$(A, \rho, \lambda x.e)$	closure
	$\mid$	(a_1, a_2)	pair
	$\mid$	$\&c$	channel identifier
a	$::=$		atom
	$\mid$	$v@L$	labeled value

Evaluation

$$\frac{}{A, pc, \rho \vdash \sigma, c \Downarrow \sigma, c@pc} \quad \text{EVAL\_CONST}$$

$$\frac{\rho(x) = v@L}{A, pc, \rho \vdash \sigma, x \Downarrow \sigma, v@(pc \vee L)} \quad \text{EVAL\_VAR}$$

$$\frac{}{A, pc, \rho \vdash \sigma, (\lambda x.e) \Downarrow \sigma, (A, \rho, \lambda x.e)@pc} \quad \text{EVAL\_ABS}$$

$$\frac{\begin{array}{l} A, pc, \rho \vdash \sigma, e_1 \Downarrow \sigma_1, (A_1, \rho_1, \lambda x.e)@L_1 \\ A \text{ can eliminate } L_1 \\ A, pc, \rho \vdash \sigma_1, e_2 \Downarrow \sigma_2, a_2 \\ A', pc, (\rho_1, x : a_2) \vdash \sigma_2, \mathsf{block}\, L_1 \,\mathsf{in}\, e \Downarrow \sigma_3, a_3 \end{array}}{A, pc, \rho \vdash \sigma, e_1\ e_2 \Downarrow \sigma_3, a_3} \quad \text{EVAL\_APP}$$

$$\frac{A, pc \vee L, \rho \vdash \sigma, e \Downarrow \sigma', a}{A, pc, \rho \vdash \sigma, \mathsf{block}\, L \,\mathsf{in}\, e \Downarrow \sigma', a} \quad \text{EVAL\_BLOCK}$$

Evaluation

$$\frac{A, pc, \rho \vdash \sigma, e \Downarrow \sigma', v@L'}{A, pc, \rho \vdash \sigma, e \vee L \Downarrow \sigma', v@(L' \vee L)} \quad \textsc{Eval\_Raise}$$

$$\frac{\begin{array}{l} A, pc, \rho \vdash \sigma, e \Downarrow \sigma', v@L' \\ L' \setminus p \sqsubseteq (L' \wedge L) \end{array}}{A, pc, \rho \vdash \sigma, e \wedge L \Downarrow \sigma', v@(L' \wedge L)} \quad \textsc{Eval\_Lower}$$

$$\frac{\begin{array}{l} A, pc, \rho \vdash \sigma, e \Downarrow \sigma', v@L' \\ L' \sqsubseteq L \end{array}}{A, pc, \rho \vdash \sigma, e < L \Downarrow \sigma', v@L'} \quad \textsc{Eval\_Check}$$

$$\frac{A', pc, \rho \vdash \sigma, e \Downarrow \sigma', a}{A, pc, \rho \vdash \sigma, \text{auth } A' \text{ in } e \Downarrow \sigma', a} \quad \textsc{Eval\_Auth}$$

Example

```
val bool =
  auth BOOL in
    let label private =
        [ E: BOOL:BOOL | * & I: * : {} ] in
    let label public  =
        [ E: * : * & I: * : {} ] in
    { true   = (\t f. t) \/ private
    ; false  = (\t f. f) \/ private
    ; ifthen = (\b t f.
                 let label L =
                   [ E: BOOL : * | {} & I: * : * ]
                 in (b t f) /\ L)
               \/ public
    } \/ public
```