DRAPER™

The Dover Architecture

Hardware Enforcement of
Software-Defined Security Policies

Team: Greg Sullivan (Draper) (presenting),
André DeHon (UPenn),
Eli Boling, Marco Ciaffi, Steve Milburn,
Nirmal Nepal, Jothy Rosenberg,
Andrew Sutherland (all Draper)

Date: November 28, 2016

New England Security Day Fall 2016
At Worcester Polytechnic Institute,
Worcester, MA

The Charles Stark Draper Laboratory, Inc.
555 Technology Square, Cambridge Mass. 02139-3563
CAGE Code: 51993

1. Brief history lesson and motivation
2. Brief overview of Dover hardware architecture.
3. Introduction to policies, as enforced on Dover
4. Motivation for discussion: more uses for policies.

Dover pre-history

2010-2015 – DARPA CRASH program – Clean Slate Security

- CRASH SAFE project (prime = BAE Systems) included U. Penn (DeHon, Pierce, Smith), Harvard (Morrisett), Northeastern (Wand, Shivers)
- Clean slate hardware, ISA, programming languages, runtime
- Tagged architecture – every word has metadata, every instruction vetted by software-defined policies
- Formal verification of security policies, with a focus on information flow control (IFC)
- ASPLOS 2015: Can we add tags and "PUMP" (Programmable Unit for Metadata Policies) to conventional RISC processor?
- papers at http://www.crash-safe.org/
- Lots of earlier history: TIARA project (Knight, Shrobe, DeHon), other tagged architectures (Intel 432, IBM System 38, Lisp Machines, etc.), information flow PLs and Oses.

Motivation – Software Security Problem

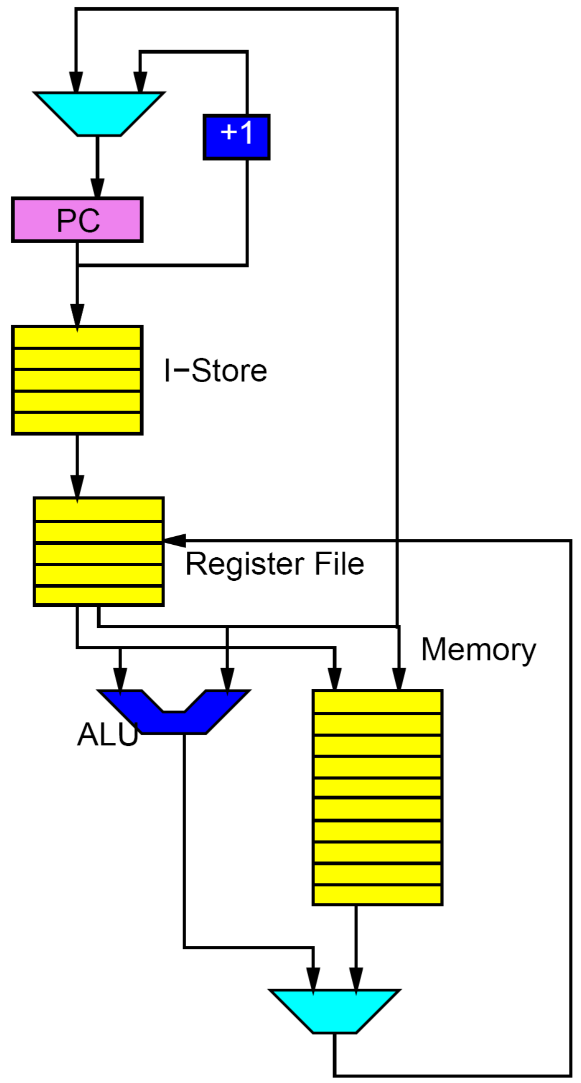
Virtually impossible to write C/C++ code without vulnerabilities

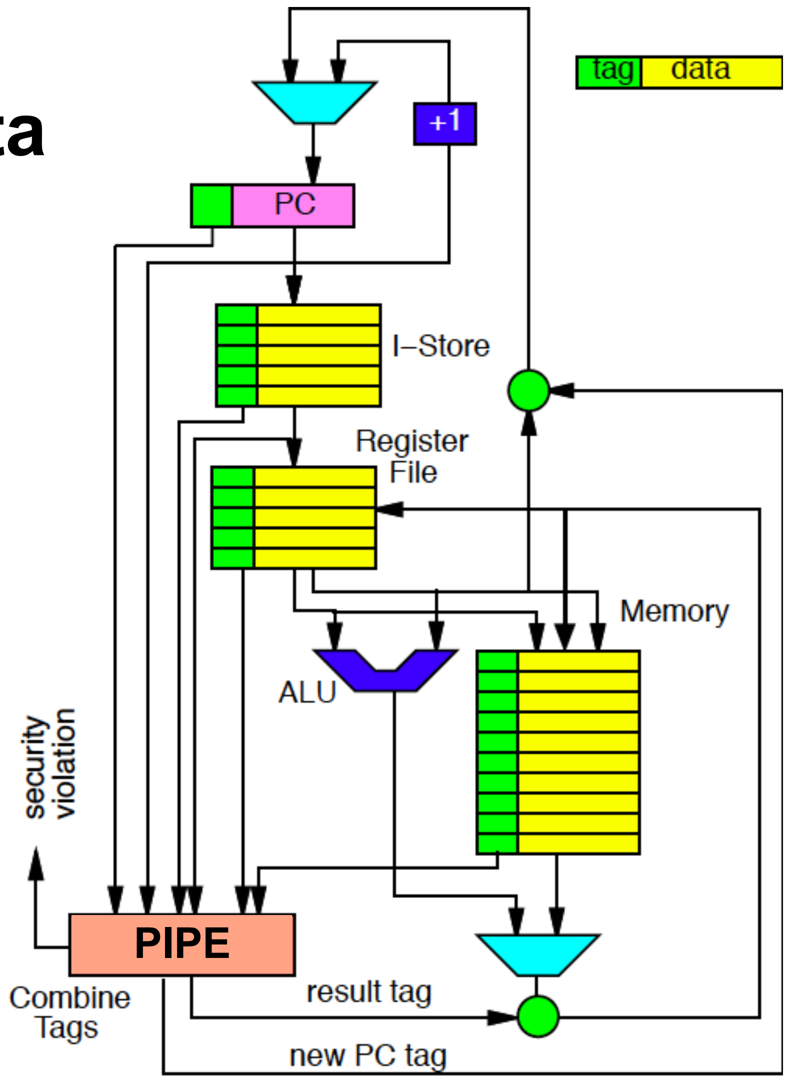
- Static analysis, formal verification: gets you part way.
- Testing: gets you part way.
- Software-based runtime security monitors: hopeless
  - Signature-based: useless, by definition, for 0-days
  - IRMs, stack canaries, ASLR, etc. – subvertible
  - "Eternal war on memory"

You can't fix buggy software with more (buggy) software.

➢ Need hardware as root of trust.

Dover: Parallel Metadata

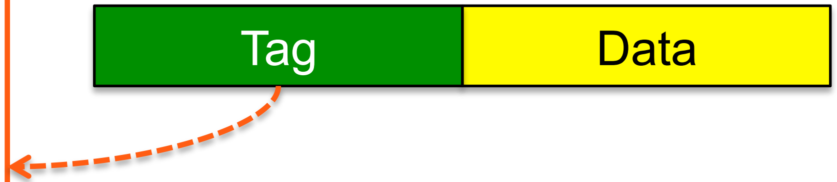


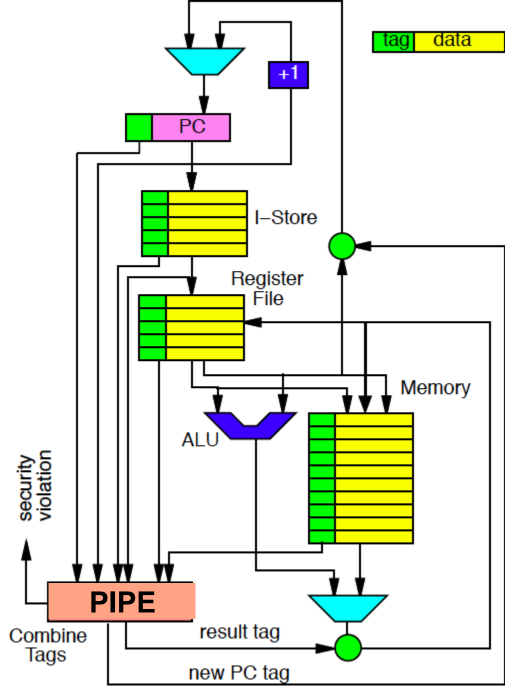


PIPE: Processor Interlocks for Policy Enforcement

Programmable Metadata

Metadata
- Provenance
- Classification
- Pointer?
- Instruction?
- Return address?
- etc.

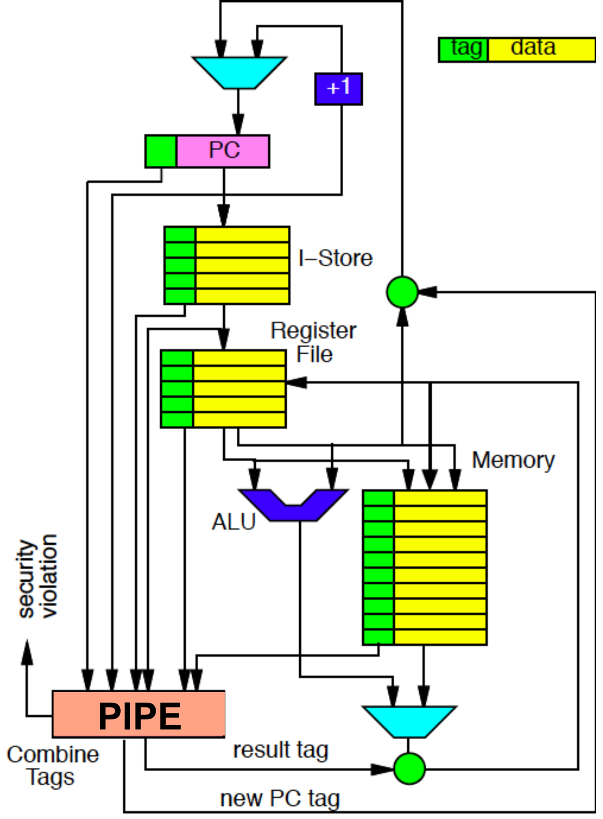




- Give each word a programmable tag
  - Indivisible from word
  - Uninterpreted by hardware
  - Software can use as pointer to data structure
- Tags checked and updated on every operation
  - Common case in parallel by PIPE "rule" cache

Abstract Function

- Every word may have arbitrary metadata
- PIPE is a function from:
  - Opcode, PC_{tag}, $Instr_{tag}$, $RS1_{tag}$, $RS2_{tag}$, MR_{tag}
- To:
  - Allowed?
  - PC_{tag}
  - $Result_{tag}$ (RD, memory result)

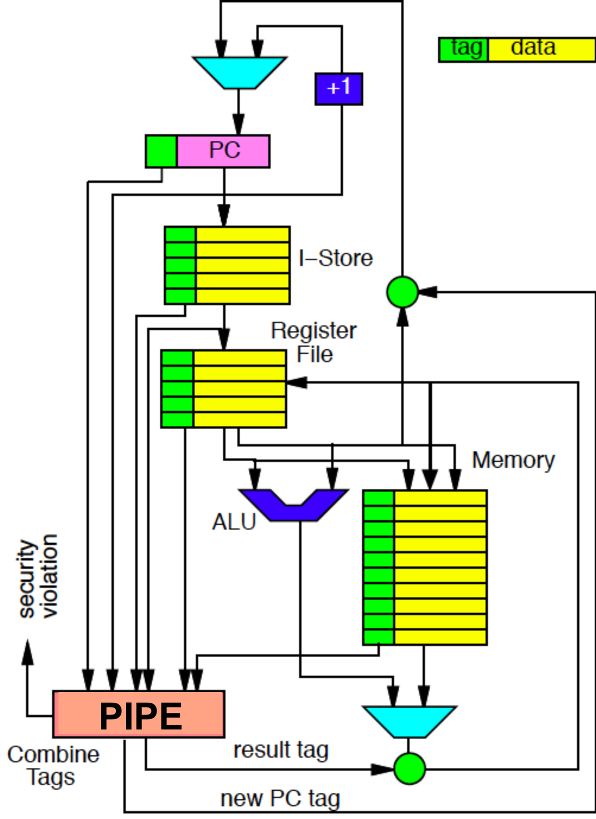


Policies

What operations are allowed and how metadata is updated

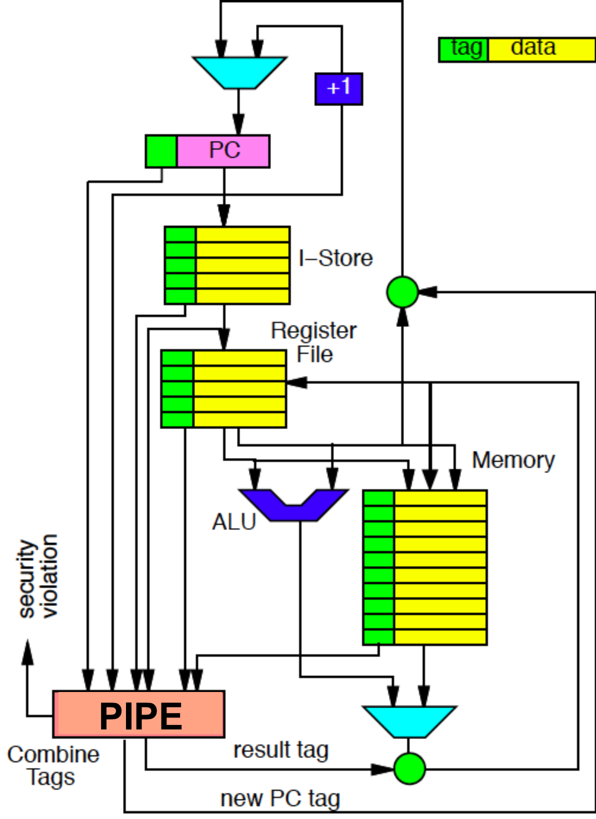
Examples:

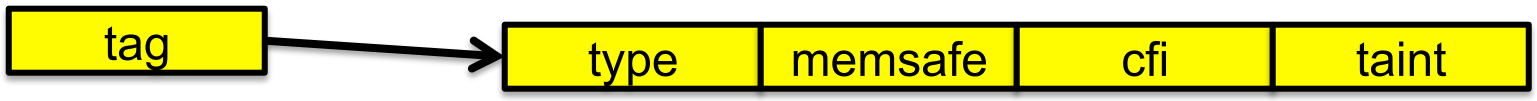
- Memory Safety
- Control Flow Integrity
- Taint tracking / Information Flow Control
- Access Control (fine-grained)
  - Mandatory Access Control
- Types (including application-defined)
- Fine-grained instruction permission



Composite Policies

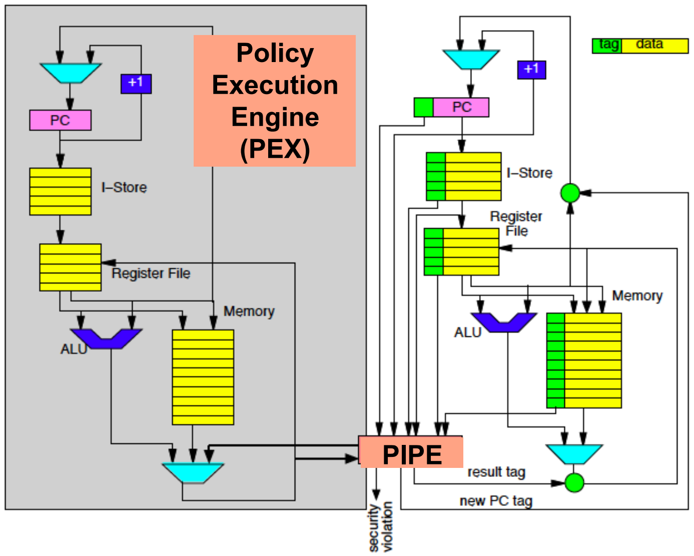
- Limiting if only support one policy at a time
- Use pointer tag to point to tuple of μpolicies
- No hardware limit on number of μpolicies supported
  - Support 0-1-∞ design principle





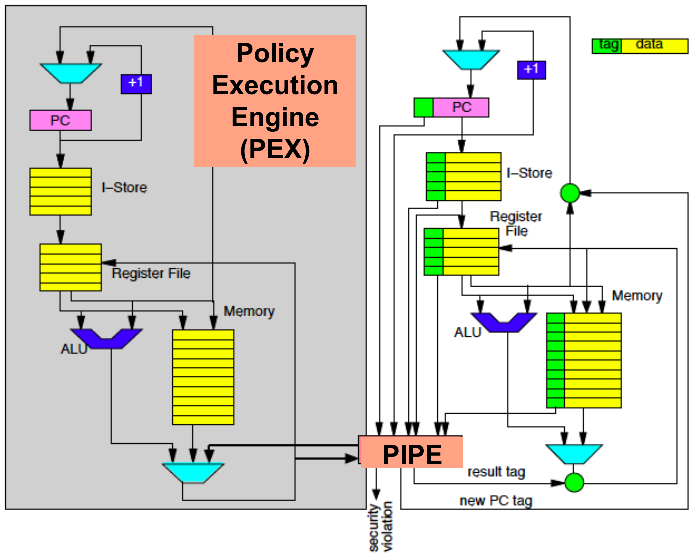
Separation

Policy Execution Engine (PEX) Coprocessor



- Data and Metadata do not mix
- Metadata not addressable
- Datapaths do not cross
- No instructions read or write metadata
  - No set-tag, no read-tag
- All metadata transforms through PIPE

Project Status



Hardware

- Building around RISC-V (open source ISA specification)
  - see https://riscv.org/
- Implemented on FPGA.
  - 1st version used Bluespec/Verilog. Current version uses just Verilog
- Aiming for ASIC tape out June 2017.
  - Both Application Processor (AP) and PEX based on 32b *Rocket* open source RISC-V design

Project Status, continued

Software

- simple “Dover Kernel” – useful for experimenting with policies.
  - Most complicated bits: booting – initializing PEX and AP; loading ELF images and applying tags to instruction words.
- modified GCC RISC-V cross-compiler to generate metadata used by loader for CFI, stack safety policies.
- modified RISC-V software simulator (“spike”) to mimic AP+PEX design
- Domain Specific Language for writing policies.
  - generates C

DRAPER™

Fun With Policies

Memory Safety *μ*-Policy

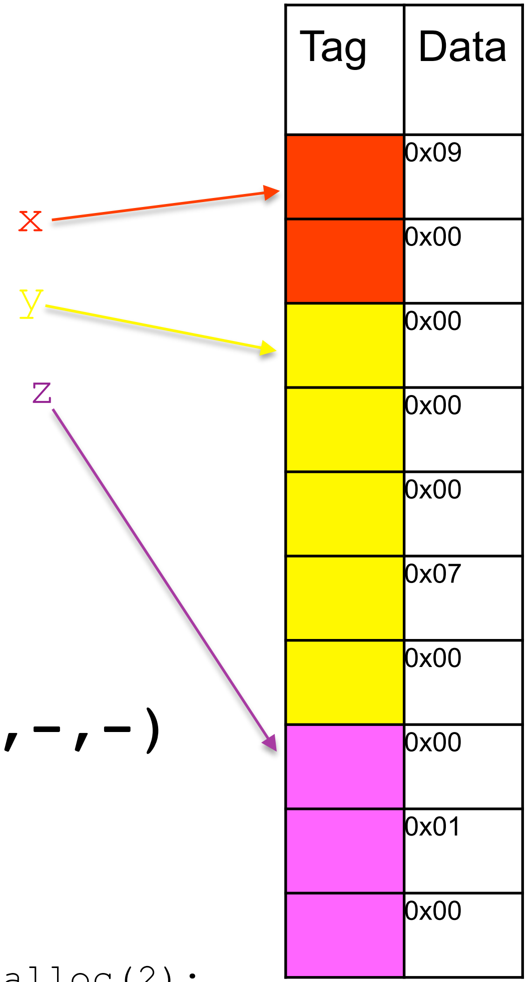
Goal: enforce spatial and temporal safety

- **Method**: give each pointer a unique "color"
  - color memory slots with this color on allocation
  - recolor on free
- **Policy**:

```
(LOAD,-,-,R1,-,MR)➡(MR==R1,-,-)
```

Require that tag (color) on pointer (R1) equals tag on pointed-to word (MR)

  - similar for `STORE`



```
x = malloc(2);
x[0]= 0x09;
y = malloc(5);
y[3] = 0x07;
z = malloc(3);
z[1] = 0x01;
x[2] = 0xbad; //FAIL
```

Reminder: (opcode, PC, INST, OP1, OP2, MR) ➡ (allow?, PC, Result)

Control Flow Integrity *μ*-Policy

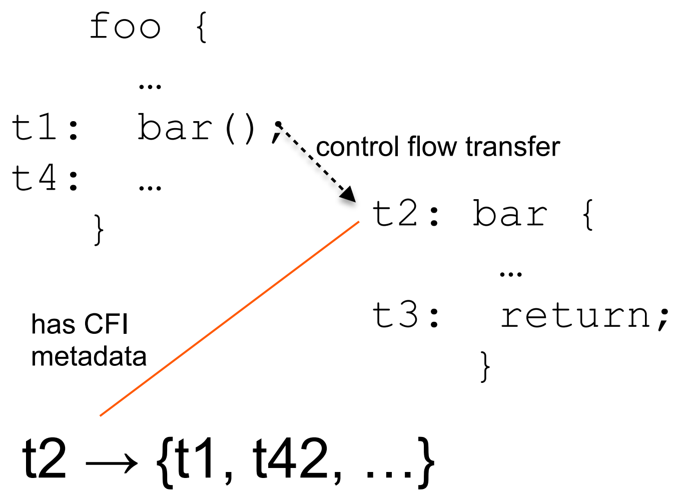
Goal: limit control transfers to those specified by program

- **Policy**:

Copy tag from call instruction to PC tag

```
(CALL,none,t1,R1,-,-) ➡ (true,t1,-)
(CALL,t1,t2,-,-,-) ➡ (t1 in t2,none,-)
```

If not a call instruction, and PC is tagged (e.g. t1), check that tag on PC (t1) is in the list of "legal caller tags" (t2) on current instruction (which must be the target of a call). Also, untag PC back to none.



$t2 \longrightarrow \{t1, t42, \ldots\}$

- Generalize for return

Reminder: (opcode, PC, INST, OP1, OP2, MR) ➡ (allow?, PC, Result)

Taint Tracking *μ*-Policy

- **Goal**: track influences of values
  - prevent untrusted values influencing critical decision
  - limit flow of sensitive data
- **Policy**:

```
(ADDL,PC,INST,OP1,OP2,-)➡
            (true,PC,union(PC,INST,OP1,OP2))
```

Tag (taint) on result is union of taints on operands.

Reminder: (opcode, PC, INST, OP1, OP2, MR) ➡ (allow?, PC, Result)

DRAPER™

Questions for Discussion

Discussion Topics

How to use metadata to implement / enforce:

- Least privilege compartmentalization
- Information flow, à la MLS (multi-level security) or more general
- Linear / Affine types (e.g. use at most once, cannot copy, etc.)
  - Canonical example: A return address should not be copied.
- Stack safety (vs. heap memory safety using colors)
- Intra-structure safety (e.g. two arrays w/in same struct – prevent overflow from one into another).
- Fully abstract compilation (being pursued by Cătălin Hriţcu et al. under ERC SECOMP project)
  - call untrusted `reverse` – restrict access to contents of list elements.
  - call untrusted `sort` – restrict access to calls to <= or compare on elements.

Q&A

Some pointers:

- CRASH SAFE papers: http://www.crash-safe.org/papers.html
- Draper Inherently Secure Processor project: http://www.draper.com/solution/inherently-secure-processor
- RISC-V: https://riscv.org/