Story

• Non-news: Computers systems are insecure
• Processor architecture contributes
  – Blindly run code
  – Make secure/safe thing expensive
• Software Defined Metadata Processing
  – Extensible architecture for concurrent metadata
  – Makes safety inexpensive and programmable
  – Learning, customization, rapid response to threats
• DOVER = RISC-V + PUMP (SDMP)
  – Programmable Unit for Metadata Processing (PUMP)
Outline

• Review SDMP
  – Software Defined Metadata Processing
• Data/Metadata Separation
• RISC-V Integration
• DRAPER and RISC-V Community
Parallel Metadata
Programmable Metadata

• 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 PUMP “rule” cache
Abstract Function

• Every word may have arbitrary metadata
• PUMP is a function from:
  – Opcode, PC\text{tag}, Instr\text{tag}, RS1\text{tag}, RS2\text{tag}, MR\text{tag}
• To:
  – Allowed?
  – PC\text{tag}
  – Result\text{tag} (RD, memory result)
Policies

• What operations are allowed and how metadata is updated

• Examples:
  – Access Control (fine-grained)
    • Mandatory Access Control
  – Types (including application-defined)
  – Fine-grained instruction permission
  – Memory Safety
  – Control Flow Integrity
  – Taint tracking / Information Flow Control
Micro-Policies

- Type Safety
- Memory Safety
- Control-Flow Integrity
- Stack Safety
- Unforgeable resource identifiers
- Abstract Types
- Immutability
- Linearity
- Software Architecture Enforcement
- Units
- Signing
- Sealing
- Endorsement
- Taint
- Confidentiality
- Integrity
- Mandatory Access Control
- Classification levels
- Lightweight compartmentalization
- Software Fault Isolation
- Sandboxing
- Access control
- Capabilities
- Provenance
- Full/Empty Bits
- Concurrency: Race Detection
- Debugging
- Data tracing
- Introspection
- Audit
- Reference monitors
- Garbage collection
- Bignums
PUMP
Uninterpreted

- PUMP provides uninterpreted functional mapping (bits-to-bits)
  - Doesn’t assign meaning, only caches
- Leaves meaning up to software (SDMP)
  - Rules installed by software on PUMP misses
- Demand metadata structures be immutable
  - So meaning of a tag (address to structure) never changes
Abstracting Hardware

• HW level
  – Metadata bits attached to word
  – PUMP to resolve

• Programmer
  – Shouldn’t have to worry about limits
    • Number of bits
    • Complexity of rule logic

• Metadata tag as pointer
  – can point to data structure of arbitrary size
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
DATA/METADATA SEPARATION
Separation

- Data and Metadata to 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 PUMP
Metadata Separation

- Could process SDMP rules (PUMP misses) on separate processor
- Could store metadata structures (what tags point to) in separate memory
- Metadata processor not need access to data memory --- operates only on metadata
Metadata on Processor

• To avoid a second processor, typically want to process miss on same processor (in isolated subsystem).

• The metadata tags become “data” to the metadata processing system
  – E.g. pointers into metadata memory space
Mechanism

• PUMP CSRs for Rule inputs outputs
  – Store PUMP tag inputs into PUMP CSRs
  – Here tags become data

• Metadata subsystem
  – Read these PUMP CSR and process
  – Write to tag results (if allowed) to PUMP CSRs
  – Triggers rule insertion for inputs ➔ outputs

• All tag updates done through Rules in PUMP
  – Controlled by metadata system

• Only metadata system can insert Rules in PUMP
  – Limited by access to PUMP CSRs
Compare

• LowRISC
  – Limited number of tag bits
  – Tags accessible to user code
  – Good for self-protection safety
  – Not adequate to enforce policies on potentially malicious code

• Oracle M7 SSM/ADI
  – Limited number of colors → good for safety
  – Fixed policy
Additions to RISC-V Architecture

• No instructions
• CSR
  – PUMP inputs
  – PUMP outputs (writing one triggers PUMP load)
  – PUMP Flush (write to trigger)
  – Tagmode, bootstrap tag, default tag
• PUMP Miss cause
Parameterization

• Tag width
  — Like RV32 vs. RV64

• Typically – same width as RISC-V Word
  — E.g. 64b on RV64
System Integration

• Global tags and rules
  – vs. per process
  – Support protection across applications that share memory

• Metadata subsystem
  – Place at hypervisor or machine mode
  – Needs to be below what it protects
    • Want to protect OS, maybe Hypervisors
RISC-V
Idiosyncrasies

• One instruction uses RS3!
  – Forces another PUMP input

• Sparse opcode sprawl
  – Opcode + funct3 + funct12 ... 22 bits!
  – From 128 entry memory to 4M?

• Multiple instructions per machine word
  – Policies want tagged instructions

• RF vs. IRF instruction dependent
  – Specification hidden in the instruction .h file
instruction_properties.cc

• **Problem:** need to know about instructions outside of just executing
  – Reads which registers?, write result?, FP or Int?
  – What address does it access? (how calculate)
    • Read or write?

• Want to derive from single-canonical source
  – Rather than code separately and change

• **Solution:** “parse” riscv/insn/* .h files and generate methods to answer
Pipeline Integration

- Bluespec 6-stage, in-order core
DRAPER AND RISC-V COMMUNITY
Open Source plans

Draper plans to make available:
- Bluespec RISC-V + Metadata changes + PUMP
- Set of basic $\mu$-policies
- Runtime support & tools

Draper is a RISC-V Foundation Platinum Sponsor:
- Will participate in standards work and IP sharing

WE ARE HIRING
- Architect, HW designers, SW engineers & Biz Dev
- Contact jothy@draper.com 617-306-8121
Draper’s Business Model

• Traditional focus on embedded systems
  – Military systems
  – Commercial (industrial controllers, IoT, medical)

• After a “design win” with a customer
  – Establish requirements for a Dover SOC
  – And a policy (and safety?) protection suite

• Inherently Secure Processing Hive
  – An ecosystem building up around Dover
  – Membership has many benefits
  – Contact jothy@draper.com 617-306-8121
**Dover Feature Roadmap**

**Dover v0.9**
- 2H 2015
  - FPGA based
  - First tool chain
  - Vanilla RISC-V Linux
  - Heartbleed demo
  - First eval boards
  - IP planning complete
  - I/O (Ethernet and flash)

**Dover v1.0**
- 1H 2016
  - FPGA based
  - MIT red team
  - HW and policy fixes
  - Address red team findings
  - More mature tools
  - Debugged RISC-V Linux

**Dover v1.1**
- 2H 2016
  - FPGA based
  - Add TPM (keys, encrypt/decrypt)
  - Port VxWorks (RTOS)
  - First ASIC customer design win
  - Full control flow integrity protection (compiler)
  - Open source D'over RISC-V
  - Professional red team (BAE, LL)
  - Initial formal verification of PUMP and policies
  - Optimize for low power

**Dover v2.0**
- 1H 2017
  - FPGA and ASIC
  - First ASIC tape out
  - Protected Linux (portions)
  - Full information flow control and MLS (language)

**Dover v2.1**
- 2H 2017
  - Add anti-tamper
  - Add trusted supply chain (MicroSemi)
  - Support Java and Javascript programming
  - Add RAD hardening (via wafer fab process)

**Dover v3.0**
- 1H 2018
  - Multicore implementation
  - Support virtualization (hypervisor)
  - Extensive formal verification complete

**Dover v3.1**
- 2H 2018
  - Support Haskel programming
  - Other (TBD)
Story

• DOVER = RISC-V + PUMP (SDMP)
  – Programmable Unit for Metadata Processing (PUMP)
• Software Defined Metadata Processing
  – Extensible architecture for concurrent metadata
  – Makes safety inexpensive and programmable
  – Learning, customization, rapid response to threats
• Computers systems are insecure
• Processor architecture contributes
  – Blindly run code
  – Make secure/safe thing expensive
More Information

• “Architectural Support for Software-Defined Metadata Processing” – ASPLOS 2015
• http://www.draper.com/solution/inherently-secure-processor
• https://vimeo.com/142503315
Backup
## Related Work

<table>
<thead>
<tr>
<th>Tag Bits</th>
<th>Propagate?</th>
<th>Outputs allow?</th>
<th>R (result)</th>
<th>PC</th>
<th>CI</th>
<th>OP1</th>
<th>OP2</th>
<th>MR</th>
<th>Usage (Example)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>x</td>
<td>soft</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>memory protection (Mondrian)</td>
</tr>
<tr>
<td>word</td>
<td>x</td>
<td>limited prog.</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>memory hygiene, stack, isolation (SECTAG)</td>
</tr>
<tr>
<td>32</td>
<td>x</td>
<td>limited prog.</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>unforgeable data, isolation (Loki)</td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td>fixed</td>
<td>fixed</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>fine-grained synchronization (HEP)</td>
</tr>
<tr>
<td>1</td>
<td>✓</td>
<td>fixed</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>capabilities (IBM System/38, Cheri)</td>
</tr>
<tr>
<td>2–8</td>
<td>✓</td>
<td>fixed</td>
<td>fixed</td>
<td>x</td>
<td></td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>types (Burroughs B5000, B6500/7500 LISP Machine, SPUR)</td>
</tr>
<tr>
<td>128</td>
<td>✓</td>
<td>fixed</td>
<td>copy</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>memory safety (HardBound, Watchdog)</td>
</tr>
<tr>
<td>0</td>
<td>✓</td>
<td>software defined</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>propagate only one</td>
<td>invariant checking (LBA)</td>
</tr>
<tr>
<td>1</td>
<td>✓</td>
<td>fixed</td>
<td>fixed</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>taint (DIFT (Devadas), Minos)</td>
</tr>
<tr>
<td>4</td>
<td>✓</td>
<td>limited programmability</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>taint, interposition, fault isolation (Raksha)</td>
</tr>
<tr>
<td>10</td>
<td>✓</td>
<td>limited prog.</td>
<td>fixed</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>taint, isolation (DataSafe)</td>
</tr>
<tr>
<td>unspec.</td>
<td>✓</td>
<td>software defined</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>flexible taint (FlexiTaint)</td>
</tr>
<tr>
<td>32</td>
<td>✓</td>
<td>software defined</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>programmable, taint, memory checking, reference counting (Harmoni)</td>
</tr>
<tr>
<td>0–64</td>
<td>✓</td>
<td>software defined</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>information flow, types (Aries)</td>
</tr>
<tr>
<td>Unbounded</td>
<td>✓</td>
<td>software defined</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>fully programmable, pointer-sized tags (PUMP)</td>
</tr>
</tbody>
</table>

Dover SoC