Micro-Policies

Formally Verified, Tag-Based Security Monitors

Cătălin Hrițcu

Inria Paris-Rocquencourt, Prosecco team
Micro-Policies collaborators

• **Formal methods & hardware architecture**

• **Current team:**
  – UPenn: Arthur Azevedo de Amorim, André DeHon, Benjamin Pierce, Antal Spector-Zabusky, Udit Dhawan
  – Inria Prosecco: Cătălin Hrițcu, Yannis Juglaret (soon DGA-Inria PhD)
  – Paris Sud (Digiteo) & Portland State: Andrew Tolmach

• **Past: DARPA CRASH/SAFE project**
  – France: Delphine Demange (IRISA Celtique), Maxime Dénès (Inria Gallium), Nick Giannarakis (Inria Prosecco), David Pichardie (IRISA Celtique)
Computer systems are insecure
Computer systems are insecure

- **Today’s computers are mindless bureaucrats**
  - “write past the end of this buffer” ... *yes boss!*
  - “jump to this untrusted integer” ... *right boss!*
  - “return into the middle of this instruction” ... *sure boss!*

- **Software bears most of the burden for security**
  - pervasive security enforcement impractical
  - bad security-performance tradeoff
  - just write secure code ... all of it!

- **Consequence:** vulnerabilities in every system
  - violations of well-studied safety and security policies
Micro-policies

• add **large tag** to each machine word

<table>
<thead>
<tr>
<th>word</th>
<th>tag</th>
<th>tag[0]</th>
<th>tag[1]</th>
<th>tag[2]</th>
</tr>
</thead>
</table>

• words in memory and registers are all tagged

<table>
<thead>
<tr>
<th>pc</th>
<th>tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>tag</td>
</tr>
<tr>
<td>r1</td>
<td>tag</td>
</tr>
<tr>
<td>r2</td>
<td>tag</td>
</tr>
<tr>
<td>mem[0]</td>
<td>tag</td>
</tr>
<tr>
<td>mem[1]</td>
<td>tag</td>
</tr>
<tr>
<td>mem[2]</td>
<td>tag</td>
</tr>
<tr>
<td>mem[3]</td>
<td>tag</td>
</tr>
</tbody>
</table>

*Conceptual model, the hardware implements this efficiently (more later)*
Tag-based instruction-level monitoring

\[
\begin{array}{|c|c|}
\hline
\text{pc} & \text{tpc} \\
\hline
\text{r0} & \text{tr0} \\
\hline
\text{r1} & \text{tr1} \\
\hline
\text{r2} & \text{tr2} \\
\hline
\end{array}
\quad
\begin{array}{|c|c|}
\hline
\text{mem[0]} & \text{tm0} \\
\hline
\text{mem[1]} & \text{tm1} \\
\hline
\text{mem[2]} & \text{tm2} \\
\hline
\text{mem[3]} & \text{tm3} \\
\hline
\end{array}
\]

\[\text{decode}(\text{mem}[1]) = \text{add} \ \text{r0} \ \text{r1} \ \text{r2}\]

\[\text{monitor} \quad \text{tpc}' \quad \text{tr0}' \quad \text{allow}\]
Tag-based instruction-level monitoring

```
<table>
<thead>
<tr>
<th>pc</th>
<th>tpc</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>tr0</td>
</tr>
<tr>
<td>r1</td>
<td>tr1</td>
</tr>
<tr>
<td>r2</td>
<td>tr2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mem[0]</th>
<th>tm0</th>
</tr>
</thead>
<tbody>
<tr>
<td>mem[1]</td>
<td>tm1</td>
</tr>
<tr>
<td>mem[2]</td>
<td>tm2</td>
</tr>
<tr>
<td>mem[3]</td>
<td>tm3</td>
</tr>
</tbody>
</table>
```

decode(mem[1]) = store r0 r1

store

monitor

bad action stopped!
Micro-policies are cool!

• **low level + fine grained**: unbounded per-word metadata, checked & propagated on each instruction

• **expressive**: can enforce large number of policies

• **flexible**: tags and monitor defined by software

• **efficient**: accelerated using hardware caching

• **secure**: formally verified to provide security
Expressiveness

• Micro-policy mechanism can efficiently enforce:
  – memory safety
  – code-data separation
  – control-flow integrity
  – compartment isolation
  – taint tracking
  – information flow control
  – monitor self-protection
  – dynamic sealing

and a lot more!

History:
• SAFE machine had separate HW mechanisms for many of these
• micro-policies were only used for IFC [Oakland’13, POPL’14]
• ... we only realized later how expressive they are [ASPLOS’15, Oakland’15]
Flexibility by example: memory safety

• Our memory safety micro-policy prevents
  – spatial violations: reading/writing out of bounds
  – temporal violations: use after free, invalid free
  – for heap-allocated data (for simplicity)

• Pointers become unforgeable capabilities
  – can only obtain a valid pointer to a memory region
    • by allocating that region or
    • by copying/offsetting an existing pointer to that region
Memory safety micro-policy

\[ p \leftarrow \text{malloc } k \]

fresh \( c \) (e.g. \( ++c \))

\[ p = \text{A8F0} @ \text{ptr}(c) \]

\[ q \leftarrow p + k \]

\[ q \leftarrow p + k \]

\[ c \neq c' \]

out of bounds

\[ !q \leftarrow 42 \]

\[ \text{free } p \]

\[ T_v ::= i \mid \text{ptr}(c) \quad \text{tags on values} \]

\[ T_m ::= M(c,T_v) \mid F \quad \text{tags on memory} \]
Memory safety micro-policy

```
p = A8F0@ptr(c)
A8FK@ptr(c) = q
q ← p + k
!q ≤ 42
```

- **free p**
- **use after free**
Efficiently executing micro-policies

<table>
<thead>
<tr>
<th>op</th>
<th>tpc</th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>tci</th>
</tr>
</thead>
</table>

lookup ➔ zero overhead hits!

found ➔

<table>
<thead>
<tr>
<th>op</th>
<th>tpc</th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>tci</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>tpc</td>
<td>t1</td>
<td>t2</td>
<td>t3</td>
<td>tci</td>
</tr>
<tr>
<td>op</td>
<td>tpc</td>
<td>t1</td>
<td>t2</td>
<td>t3</td>
<td>tci</td>
</tr>
<tr>
<td>op</td>
<td>tpc</td>
<td>t1</td>
<td>t2</td>
<td>t3</td>
<td>tci</td>
</tr>
<tr>
<td>op</td>
<td>tpc</td>
<td>t1</td>
<td>t2</td>
<td>t3</td>
<td>tci</td>
</tr>
</tbody>
</table>

hardware cache

```plaintext
<table>
<thead>
<tr>
<th>tpc'</th>
<th>tr</th>
</tr>
</thead>
<tbody>
<tr>
<td>tpc'</td>
<td>tr</td>
</tr>
<tr>
<td>tpc'</td>
<td>tr</td>
</tr>
<tr>
<td>tpc'</td>
<td>tr</td>
</tr>
</tbody>
</table>
```
Efficiently executing micro-policies

lookup → misses trap to software produced “rule” cached

hardware cache
Simulations for **naive** implementation

memory safety + code-data separation + taint tracking + control-flow integrity

simple RISC processor: single-core 5-stage in-order Alpha

---

**mean 50%**

**mean 220%**
Targeted [micro-]architectural optimizations

[ASPLOS’15]

• grouping opcodes and ignoring unused tags
  – increases effective rule cache capacity

• transferring only unique tags to/from DRAM
  – reduces runtime and energy overhead

• using much shorter tags for on-chip data caches
  – reduces runtime, energy, and area overhead

• caching composite policies separately
  – makes rule cache misses much cheaper
Simulations for **optimized** implementation

memory safety + code-data separation + taint tracking + control-flow integrity

simple RISC processor: single-core 5-stage in-order Alpha

**no free lunch**

50% -> 7%

220% -> 60%

17
FORMAL VERIFICATION IN COQ

[POPL’14, Oakland’15]
Memory safe abstract machine

Symbolic machine

Concrete machine

Micro-policy

Correctly implements memory safety

Correctly implements micro-policy

Correctly implements memory safety monitor

*Only proved for IFC [POPL 2014]
Abstract machine for P

Symbolic machine

Correctly implements

Concrete machine

Rule cache

Micro-policy

Monitor

Correctly implements

Monitor for P

P in \{IFC, CFI\}

Secure

(e.g. noninterference)
1. Sets of tags

\[ T_v ::= i \mid \text{ptr}(c) \]
\[ T_m ::= M(c,T_v) \mid F \]
\[ T_{pc} ::= T_v \]

2. Transfer function

Record \( \text{IVec} \) := \{ \text{op}:\text{opcode} ; t_{pc}:T_{pc} ; t_i:T_m ; \text{ts}: \ldots \} \\
Record \( \text{OVec} (\text{op}:\text{opcode}) \) := \{ t_{rpc}:T_{pc} ; t_r: \ldots \} \\
\text{transfer} : (\text{iv}:\text{IVec}) \rightarrow \text{option} (\text{OVec} (\text{op} \text{ iv}))

Definition \text{transfer} \ iv :=

match \ iv \ with

| \{ \text{op}=\text{Load}; t_{pc}=\text{ptr}(c_{pc}); t_i=M(c_{pc},i); \text{ts}=[\text{ptr}(c); M(c,T_v)]\} \\
\rightarrow \{ t_{rpc}=\text{ptr}(c_{pc}); t_r=T_v \}

| \{ \text{op}=\text{Store}; t_{pc}=\text{ptr}(c_{pc}); t_i=M(c_{pc},i); \text{ts}=[\text{ptr}(c); T_v; M(c,T_v')]\} \\
\rightarrow \{ t_{rpc}=\text{ptr}(c_{pc}); t_r=M(c,T_v) \}

...
Memory safety micro-policy

1. Sets of tags
\[ T_v ::= i \mid \text{ptr}(c) \]
\[ T_m ::= M(c,T_v) \mid F \]
\[ T_{pc} ::= T_v \]

2. Transfer function
Record \textbf{IVec} ::= \{ op:opcode ; t_{pc}:T_{pc} ; t_i:T_m ; ts: ... \}
Record \textbf{OVec} (op:opcode) ::= \{ t_{rpc} : T_{pc} ; t_r : ... \}
\textbf{transfer} : (iv:IVec) -> option (OVec (op iv))

3. Monitor services
Record \textbf{service} ::= \{ addr : word; sem : state -> option state; ... \}
Definition \textbf{mem\_safety\_services} : list service :=
[\textit{malloc}; \textit{free}; \textit{base}; \textit{size}; \textit{eq}].
Open problems

• Interaction with compiler, loader, linker, OS
• Secure micro-policy composition
• Reduce energy + more adaptive usage
• Modern RISC instruction set (e.g. ARM)
• More realistic processor
  (our-of-order execution, even multi-core)
Fully abstract compilation

- Golden standard for secure compilation
  - $P \approx Q \iff \text{compile}(P) \approx \text{compile}(Q)$
  - $P \approx Q = \forall C. C[P]$ has the same behavior as $C[Q]$
  - *intuition*: low-level machine code contexts can’t do more harm than high-level contexts
    - can securely link compiled and untrusted machine code

- Very strong, but rarely achieved in practice
  - much stronger than compiler correctness
  - need a compiler & runtime that actually enforce high-level abstractions at the low level
    - ... and that’s currently too expensive!
Targeting micro-policy machine

• Micro-policies can **efficiently protect abstractions**

• **Fully abstract compiler to micro-policy machine**
  – Recently started with **Yannis Juglaret**
  – Toy source language: Featherweight Java subset
  – FJ classes protected from native classes they link with

  – Micro-policy combining: protects:
    • compartment isolation classes
    • linear return capabilities stack discipline
    • dynamic typing type safety

• **Long term goal**: functional programming language
Take away

• **Micro-policies**, novel security mechanism
  – low level, fine grained, expressive, flexible, efficient, formally secure

• cool research direction with many interesting open problems for us and others to solve
BACKUP SLIDES
Other highlights in **Prosecco** team

- **Pro**gramming securely with **cryptography**
- **Proverif** and **Cryptoverif** protocol analyzers
- **miTLS**: verified reference implementation
- **F***: program verification system for OCaml/F#
- **QuickChick**: property-based testing for Coq
- Permanent researchers:
  - Karthikeyan Bhargavan, Bruno Blanchet, Cătălin Hriţcu, Graham Steel