Micro-Policies
Hardware-Assisted Tag-Based Security Monitors

Cătălin Hrițcu
Inria Paris-Rocquencourt, Prosecco team
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
HP reinventing the computer

• opportunity to fix this:
  – devise a computer that’s **not just faster**, but that’s also **significantly more secure**

• it’s possible!
  – new security mechanism called **micro-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>
<th>mem[0]</th>
<th>tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>tag</td>
<td>mem[1]</td>
<td>tag</td>
</tr>
<tr>
<td>r1</td>
<td>tag</td>
<td>mem[2]</td>
<td>tag</td>
</tr>
<tr>
<td>r2</td>
<td>tag</td>
<td>mem[3]</td>
<td>tag</td>
</tr>
</tbody>
</table>
```
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]) = add r0 r1 r2

monitor

allow

add
Tag-based instruction-level monitoring

```
decode(mem[1]) = store r0 r1
```

![Diagram showing the process of decoding and storing values in memory with a monitor to disallow bad actions.](image)
Features of micro-policies

- **low-level and fine-grained**: large per-word tags, checked and propagated on each instruction
- **expressive**: can enforce large number of policies
- **flexible**: tags and monitor defined by software
- **efficient**: hardware caching
- **secure**: formally verified to provide security
Expressiveness

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

and probably a lot more!

History:
• DARPA CRASH/SAFE project
• different mechanisms for most of these things
• micro-policies were only used for IFC ... but they are a lot more expressive than we realized at first
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 now)

• 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 ← malloc k
fresh c

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

\( q \leftarrow p + k \)

\( \text{free p} \)

\( \text{c} \neq c' \)

\( \text{out of bounds} \)

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

\( T_m ::= M(c, T_v) \mid F \) tags on memory
Memory safety micro-policy

p = A8F0@ptr(c)

A8FK@ptr(c) = q

c != c'

q ← p + k

!q < 42  
out of bounds

T_v ::= i | ptr(c)  tags on values
T_m ::= M(c,T_v) | F  tags on memory
Efficiently executing micro-policies

lookup → zero overhead hits!

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

lookup → misses trap to software produced “rule” cached

hardware cache
Experiments for naive implementation

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

simple RISC processor: 5-stage in-order Alpha

![Graph showing runtime overhead for different benchmarks, with mean 50%]

![Graph showing energy overhead for different benchmarks, with mean 220%]
Targeted architectural optimizations

• grouping opcodes and ignoring unused tags
• transferring only unique tags to/from DRAM
• using much shorter tags on-chip
• caching composite policies separately
Experiments for optimized impl.

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

simple RISC processor: 5-stage in-order Alpha

no free lunch

50% -> 7%

220% -> 60%
Formally verified security
(using Coq proof assistant)

Memory safe abstract machine

correctly implements

Symbolic machine

Micro-policy

correctly implements

Concrete machine

Rule cache

Monitor

Correctly implements memory safety

Micro-policy

Correctly implements memory safety

Monitor

ASM

Generic Framework
Abstract machine for P

Symbolic machine

Correctly implements

Concrete machine

Rule cache

Monitor

Micro-policy

P in \{\text{IFC}, \text{CFI}\}

Secure (e.g. noninterference)

Secure
Upcoming

• Interaction with loader, compiler, and OS
• Secure micro-policy composition
• Better energy efficiency + adaptive usage
• Modern RISC instruction set (e.g. ARM)
• More realistic processor
  (our-of-order execution, multi-core)
Take away

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

• Current collaborators (**INRIA & UPenn**):
  – Arthur Azevedo de Amorim, André DeHon, Maxime Dénès, Udit Dhawan, Nick Giannarakis, Cătălin Hrițcu, Yannis Juglaret, Benjamin Pierce, Antal Spector-Zabusky, Andrew Tolmach, Nikos Vasilakis
Other highlights in Prosecco team
Other highlights in Prosecco team

• programming securely with cryptography
• **Proverif** and **Cryptooverif** protocol analyzers
• **miTLS**: verified reference implementation
• **F***: program verification system for OCaml/F#
• **QuickChick**: property-based testing for Coq
• Prosecco permanent researchers:
  – Karthikeyan Bhargavan (leader), Bruno Blanchet, Cătălin Hrițcu, Graham Steel (Cryptosense startup)
BACKUP SLIDES
Current collaborators on this project

- **Formal verification**
  - Arthur Azevedo de Amorim (UPenn; INRIA intern 2014)
  - Maxime Dénès (INRIA Gallium; previously UPenn)
  - Nick Giannarakis (ENS Cachan; INRIA intern 2014)
  - Cătălin Hrițcu (INRIA Prosecco; previously UPenn)
  - Yannis Juglaret (Paris 7; INRIA intern 2015)
  - Benjamin Pierce (UPenn)
  - Antal Spector-Zabusky (UPenn)
  - Andrew Tolmach (Portland State)

- **Hardware architecture**
  - André DeHon, Udit Dhawan, ... (UPenn)
The end

• Today’s computer’s were designed long time ago

• **Computer designers from the 50s-90s have a good excuse for getting security wrong** (e.g. horrors like buffer overflows):
  – security wasn't a big issue before the Internet age
  – performance was much more important

• Today the situation is reversed
  – and HP has an opportunity to fix security
  – **but HP will have no excuse** if it reinvents the insecure computer