SECOMP
Efficient Formally Secure Compilers
to a Tagged Architecture

ătălin Hrițcu

European Research Council
new grant
Computers are insecure

• devastating low-level vulnerabilities
• programming languages, compilers, and hardware architectures

• the world has changed

* “...the number of UNIX installations has grown to 10, with more expected...”
  -- Dennis Ritchie and Ken Thompson, June 1972
Teasing out 2 important problems

• 1. inherently insecure low-level languages
  – memory unsafe

• 2. unsafe interoperability with lower-level code
  – safer high-level languages
  – insecure low-level libraries
  – unsafe interoperability:
Key enabler: Micro-Policies

software-defined, hardware-accelerated, tag-based monitoring

store

software monitor’s decision is hardware cached
**Key enabler: Micro-Policies**
software-defined, hardware-accelerated, tag-based 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>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>mem[0]</th>
<th>tm0</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;store r0 r1&quot;</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>
```

- `store rϬ rϭ`: Disallow policy violation stopped! (e.g. out of bounds write)
- `monitor`
Micro-policies are cool!

• low level + fine grained
• flexible
• efficient
• expressive
• secure simple
• real

Micro-policies are cool!
Expressiveness

- information flow control (IFC)
- monitor self-protection
- protected compartments
- dynamic sealing
- heap memory safety
- code-data separation
- control-flow integrity (CFI)
- taint tracking

Verified (in Coq)
Evaluated (<10% runtime overhead)

[POPL’14]
[Oakland’15]
[ASPLOS’15]
Way beyond MPX, SGX, SSM, etc
• **Formal methods** architecture systems

• **Current team**
  – *Inria Paris* Cătălin Hrițcu, Guglielmo Fachini, Marco Stronati, Yannis Juglaret
  – *UPenn* André DeHon Benjamin Pierce, Arthur Azevedo de Amorim Nick Roessler
  – *Portland State* Andrew Tolmach
  – *MIT*: Howie Shrobe, Stelios Sidiropoglou-Douskos
  – *Industry*: Draper Labs

• Spinoff of past project: DARPA CRASH/SAFE (2011-2014)
SECOMP grand challenge

1. Provide secure semantics for low-level languages
2. Enforce secure interoperability with lower-level code

the first efficient formally secure compilers realistic programming languages
Formally verify: **full abstraction**

**Benefit**: sound security reasoning in the source language
Fully abstract compilation, definition

1st high-level component \\rightarrow \text{high-level attacker} \\leftarrow \text{1st compiled component}

∃ high-level attacker

∃ low-level attacker

compiler

\not\exists \text{compiler}

1st compiled component \\rightarrow \text{low-level attacker} \\leftarrow \text{1st high-level component}

2nd high-level component \\rightarrow \text{high-level attacker} \\leftarrow \text{2nd compiled component}

\not\exists \text{compiler}

2nd compiled component \\rightarrow \text{low-level attacker} \\leftarrow \text{2nd high-level component}
SECOMP: achieving full abstraction at scale

Low* language

C language

ASM language
Protecting component boundaries

- Add mutually distrustful components to C
  - strictly enforced interfaces

- CompSec compiler chain
  -

- Micro-policy simultaneously enforcing
  -
  -

- Interesting attacker model
  -

Recent work, joint with Yannis Juglaret et al
Protected components micro-policy

Protected components micro-policy

Protected components micro-policy

Protected components micro-policy

Protected components micro-policy

Protected components micro-policy

memory

Jal r
...
...
...@EntryPoint
Store \( r_a \to *r_m \)
...
Load \( *r_m \to r_a \)
Jump \( r_a \)

registers

linear return capability

\( @(n+1) \)

\( @\text{Ret n} \)

\( \text{pc} \)

\( r_a \)

\( r_m \)

loads and stores to the same component always allowed
Protected components micro-policy

memory

Jal r
...
...
...@EntryPoint
Store \( r_a \rightarrow hr_m \)
...
Load \( hr_m \rightarrow r_a \)
Jump \( r_a \)

registers

\( \text{pc} \) \( r_a \) \( r_m \)

linear return capability

\( @\text{Ret n} \)

\( @(n+1) \)
Protected components micro-policy

Memory:
- Jal r
- ... (multiple levels)
- @EntryPoint
  - Store \( r_a \rightarrow *r_m \)
  - ... (multiple levels)
- Jump \( r_a \)

Registers:
- @Ret n
- \( pc \)
- \( r_a \)
- \( r_m \)

Invariant:
At most one return capability per call stack level.
Protected components micro-policy

memory

Jal \textit{r}

\ldots

\ldots

\ldots@@EntryPoint

\textit{Store }r_{a} \rightarrow h_{r_{m}}

\ldots

\textit{Load }h_{r_{m}} \rightarrow r_{a}

\textit{Jump }r_{a}

\textbf{invariant:}

at most one return capability per call stack level

registers

\textit{pc } \rightarrow \textit{ra } \rightarrow \textit{rm}

\textit{linear return capability}

\textit{\@Ret n}
Protected components micro-policy

memory

<table>
<thead>
<tr>
<th>Jal r</th>
<th>...</th>
<th>...</th>
<th>...@EntryPoint</th>
<th>Store $r_a \rightarrow *r_m$</th>
<th>...</th>
</tr>
</thead>
</table>

| Jump $r_a$     | @Ret $n$       | @Ret $n$       |                | linear return capability    |                |

registers

<table>
<thead>
<tr>
<th>pc</th>
<th>$r_a$</th>
<th>$r_m$</th>
</tr>
</thead>
</table>

invariant: at most one return capability per call stack level

cross-component return only allowed via return capability

@$(n+1)$
∀ low-level attack from compromised C
↓, C↓, C↓
∃ high-level attack from some fully defined A
↓, A↓, A↓

∀ compromise scenarios.

follows from "structured full abstraction for unsafe languages" + "separate compilation"

[Beyond Good and Evil, Juglaret, Hritcu, et al, CSF’16]
Protecting higher-level abstractions

• Low*: enforcing specifications using micro-policies
  contracts,
  trivial for C interfaces

• Limits of purely-dynamic enforcement
  push these limits further and combine with static analysis
but combining with static analysis can ...

• improve efficiency
  – removing spurious checks

• improve transparency
  – allowing more safe behaviors
  – unsound static analysis is fine
Beyond full abstraction

- Variants / similar properties
  -

- Strictly weaker properties
  -

- Orthogonal properties
  -
What secure compilation adds over compositional compiler correctness

- mapping back arbitrary low-level contexts
- preserving integrity properties
- preserving confidentiality properties
- stronger notion of components and interfaces
Verification and testing

• on paper
  – but one can’t verify an interesting compiler on paper
• proof assistants
• Reduce effort
  –
  –
• Problems not just with effort/scale
  – proof techniques
    is a hot research topic of its own
Micro-policies:
remaining fundamental challenges

•Micro-policies for C

•Secure micro-policy composition
  interferent
  policy’s behavior can break another’s guarantees
SECOMP in a nutshell

• We need more **secure languages, compilers, hardware**
• Key enabler: **micro-policies**
• Grand challenge: **the first efficient formally secure compilers**
  realistic programming languages
• Answering challenging fundamental questions
  
  --
  
  --

• Achieving strong security properties like full abstraction
  +

• Measuring & lowering the cost of secure compilation

• Measuring & lowering the cost of secure compilation
  vaporware

• in order to try to make some of this real
• Looking for excellent interns, PhD students, PostDocs, starting researchers, and engineers.

• We can also support outstanding candidates in the Inria permanent researcher competition.
Collaborators & Community

- Traditional collaborators from Micro-Policies project

- Several other researchers working on **secure compilation**

- Secure compilation meetings (informal)

  - build larger research community, identify open problems, bring together communities
BACKUP SLIDES
To compose compilers need
1. higher-level micro-policies
2. composing micro-policies
• user-specified micro-policies for C

• micro-policy composition is easy
  • But how do we ensure programmers won’t break security?
  • secure micro-policy composition is hard!