Efficient Formally Secure Compilers to a Tagged Architecture

E  std
   st  stt
     st tf dd f

5 year vision
ERC SECOMP: https://secure- compilation.github.io
Computers are insecure

• devastating low-level vulnerabilities

• programming languages, compilers, and hardware architectures
  – f t h f  f s t  t d  s f  s t  s f  s t  s f  t
  – f  s t  f  t f d  s t  s f  d f  d

• the world has changed
  – t f d  s t  f s t  s t  s f  s t  s f  t
  – f  s t  t  t  f  s t  f  t

---

-Dennis Ritchie and Ken Thompson, June 1972
Teasing out 2 important security problems

• 1. inherently insecure low-level languages
  – memory unsafe
    fśť fśť d f d tśť d
    hśť f ćdnfśť h d f f d śť

• 2. unsafe interoperability with lower-level code
  – ff d f śť f safer languages
    t fśť fśť f insecure low-level libraries
  – unsafe interoperability:
    h f f t f h śť fft t
Key enabler: Micro-Policies
Key enabler: Micro-Policies

```
t  f  f  t  t  f  h  t  f  t  h
```

```
<table>
<thead>
<tr>
<th>pc</th>
<th>mem[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>t</td>
</tr>
<tr>
<td>r1</td>
<td>mem[2]</td>
</tr>
<tr>
<td></td>
<td>mem[3]</td>
</tr>
</tbody>
</table>
```
Key enabler: Micro-Policies

Software-defined, hardware-accelerated, tag-based monitoring
Key enabler: Micro-Policies

```
<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>mem[2]</td>
<td>tm2</td>
</tr>
<tr>
<td>mem[3]</td>
<td>tm3</td>
</tr>
</tbody>
</table>
```

```
t  sft  fs  ss  sft  ddf  f  sft  f  h  tf  sft  h
```

```
tpc  tr0  tr1  tm3  tm1
```

```
mmonitor
```

Software-defined, hardware-accelerated, tag-based monitoring.
Key enabler: Micro-Policies

- Store r0 r1
- mem[0] tm0
- mem[2] tm2
- mem[3] tm3

Monitor software-defined, hardware-accelerated, tag-based monitoring
Key enabler: Micro-Policies

<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>mem[2]</td>
<td>tm2</td>
</tr>
<tr>
<td>mem[3]</td>
<td>tm3</td>
</tr>
</tbody>
</table>

monitor

allow

tpc’ tm3’
Key enabler: Micro-Policies

Software-defined, hardware-accelerated, tag-based monitoring =

![Diagram with tables and logic flow]
Key enabler: Micro-Policies

Software monitor’s decision is hardware cached
Key enabler: Micro-Policies

Monitor:
- Disallow policy violation stopped!
- Store r0 r1

Software-defined, hardware-accelerated, tag-based monitoring (e.g., out of bounds write)
Micro-policies are cool!

- low level + fine grained
  - unbounded per word metadata, checked & propagated on each instruction
- flexible
  - tags and monitor defined by software
- efficient
  - software decisions hardware cached
- expressive
  - complex policies for secure compilation
- secure and simple enough to verify security in Coq
- real
  - FPGA implementation on top of RISC-V
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)

Way beyond MPX, SGX, SSM, etc
Micro-Policies team

• Formal methods  architecture  systems
• Current team
  – Inria Paris  Cătălin Hrițcu, Guglielmo Fachini, Marco Stronati, Théo Laurent
  – 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

the first efficient formally secure compilers realistic programming languages

1. Provide secure semantics for low-level languages
   - E st f d f d f t f t f

2. Enforce secure interoperability with lower-level code
   - U E
     t f E t tf f f f st f st d
Secure Compilation

**Benefit**: sound security reasoning in the source language

- High-level attacker
- Low-level attacker
- Source component
- Target component
- Compiler correctness
- Program behavior
- Compiler
- Secure compilation
- Secure
- Protected
- No extra power
- E.g. arbitrary machine code

- Secure compilation
- Secure
- Secure
- Not enough
- Not enough
- Secure
- Secure
- Secure
- Secure
- Secure
- Secure
Our original secure compilation target:
fully abstract compilation

\[ \exists \text{high-level attacker} \]

\[ \exists \text{low-level attacker} \]

Problems
very hard to realistically achieve
very difficult to prove
Our new target: robust compilation

∀ safety properties π

∃ high-level attacker breaking π

∃ low-level attacker breaking π

Advantages easier to realistically achieve and prove
still useful ⌈tf ⌈ invariants ⌈ integrity properties

• preservation of robust safety
  t f f ⌈t ⌈t d f

• gives up ⌈t f ⌈t
  ⌈t f ⌈t f t d f
  − ⌈t t t f d f t

• conjectures:
  − stronger d f ⌈t
    d ⌈t d f t
  − weaker t ⌈t
    d f ⌈t ⌈t d f t

• less extensional
SECOMP: achieving secure compilation at scale

Low* language

C language

ASM language

Low* language

C language

ASM language

Low* language

C language

ASM language
Protecting component boundaries

- Add mutually distrustful components to C
  - $f \text{std } h$ strictly enforced interfaces

- CompSec compiler chain
  - tf E E f st

- Micro-policy simultaneously enforcing
  - d f tf st
  - f t f std df std d std st td f

- Interesting attacker model
  - t std t f ft std f h hf

Ongoing work, started with Yannis Juglaret et al
Protected components micro-policy

Protected components micro-policy

Protected components micro-policy

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

\[ \text{invariant:} \]
\[ \text{at most one return capability per call stack level} \]
Protected components micro-policy

\[ f \cdot st \]

M \cdot st

F \cdot st

U \cdot st \cdot st

st \cdot st

M \cdot st

\begin{align*}
\text{invariant:} & \\
& t \cdot f \\
& st \cdot st \cdot d \\
& f \cdot st \cdot t \cdot dn \cdot f
\end{align*}
Protected components micro-policy

invariant:

\( \text{t f} \)
\( \text{f s t d} \)
\( f \text{s d t d n} f f \)

cross-component return only allowed via return capability
Mutual-distrust attacker model

\[ \text{∃ high-level attack from some fully defined } A_2, A_4, A_5 \]

\[ \forall \text{ compromise scenarios } s. \]

\[ \forall \text{ scenario-indexed safety properties } \pi. \]

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

- **Low***: enforcing specifications in C
  - t f d f $\text{contracts}$, d f dnf
  - d micro-policies can speed this up

- **Limits of purely-dynamic enforcement**
  - d s$t$ f s$t$ $:\text{s$t}$ $:t $ h
  - push these limits further and combine with static analysis
SECOMP focused on dynamic enforcement but combining with static analysis can ...

• improve efficiency
  – removing spurious dynamic checks
  – f h s t f s i d f d n h s i t d f s t t f d f f f s i i f t s i i i d f f t f s i t

• improve transparency
  – allowing more safe behaviors
  – f h t d f f d d d f s i i s i t d f d f t f s i s i t
  – t d t f unsound static analysis is fine
Verification and testing

- So far most secure compilation work on paper
  - one can't verify an interesting compiler on paper
- SECOMP uses proof assistants: Coq and F∗
  - reduce effort
    - more automation (e.g. based on SMT, like in F∗)
    - integrate testing and proving (QuickChick and Luck)
- Problem not just with scale of mechanization
  - devising good proof techniques for secure compilation is a hot research topic of its own
Remaining challenges for micro-policies

• Micro-policies for C
  – $fff$ $std$ $d$ $fstd$ $t$
  – $dst$ $dft$ $f$ $t$ $fhst$ $fst$

• Secure micro-policy composition
  – $dst$ $dft$ $ft$ interferent $fst$ $df$ $st$
  – $f$ $dst$ $d$ $t$ $f$ $std$ $fnn$ $fst$ $h$ $ft$
  • $fhd$ $t$ $h$ $h$ $Ed$ $fn$
SECOMP in a nutshell

• We need more secure languages, compilers, hardware

• Key enabler: micro-policies

• Grand challenge: the first efficient formally secure compilers

• Answering challenging fundamental questions
  – properties/attacker models, proof techniques
  – secure composition, micro-policies for C

• Achieving strong security properties

• Measuring & lowering the cost of secure compilation

• Most of this is vaporware at this point but...
BACKUP SLIDES
Collaborators & Community

• Core team at Inria Paris
  – Looking for interns, students, researchers, and engineers
  – Traditional collaborators from Micro-Policies project
  – Other researchers working on secure compilation
  – Secure compilation meetings
  – Build larger research community, identify open problems, bring together communities
Broad view on secure compilation

• Different security goals / attacker models
  - robust compilation

• Different enforcement mechanisms
  - reference monitors

• Different proof techniques
  - simulation