Micro-Policies
A Framework for Verified, Tag-Based Security Monitors

Cătălin Hrițcu
INRIA Paris-Rocquencourt, Prosecco team
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 Tolmachh (Portland State)

- **Hardware architecture**
  - André DeHon, Udit Dhawan, ... (UPenn)
Computer systems are insecure
Computer systems are insecure

• Today’s CPUs 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
  – 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

• general **dynamic enforcement mechanism** for
  – critical invariants of **all** machine code
  – high-level abstractions and programming models

• main idea: add **word-sized tag** to each machine word
  – “this word is an instruction, and this one is a pointer”
  – “this word comes from the net, and this is private to A and B”

• **tags propagated on each instruction** ... efficiently
  – tags and rules **defined by software** (**miss handler**; **verified**)
  – **accelerated by hardware** (**rule cache**, near-zero overhead hits)
Micro-policies for ...

- information flow control (IFC)  [Oakland’13, POPL’14]
- monitor self-protection
- compartmentalization
- dynamic sealing
- memory safety
- code-data separation
- control-flow integrity (CFI)
- taint tracking
- ...

Verified (in Coq)  [Oakland’15]

Evaluated (<10% runtime overhead)  [ASPLOS’15]
Memory safety

• Prevent
  – spatial violations: reading/writing out of bounds
  – temporal violations: use after free, invalid free

• 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 \( c \)

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

\[ c = c \]

\[ q \leftarrow p + k \]

\[ q \leftarrow p + k \]

\[ c = c \]

\[ c' = c \]

\[ c' = c \]

\[ q \leftarrow 42 \]

out of bounds

\[ T_v ::= i \mid \text{ptr}(c) \]

\[ T_m ::= M(c,T_v) \mid F \]

tags on values

tags on memory
Memory safety micro-policy

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

\[ A8FK \text{@ptr}(c) = q \]

\[ q \leftarrow p + k \]

\[ c \neq c' \]

\[ !q \leftarrow 42 \]

**Out of bounds**

- Free p
- Use after free

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

\[ T_m ::= M(c, T_v) | F \quad \text{tags on memory} \]
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 \( IVec ::= \{ \text{op:opcode} ; t_{pc}:T_{pc} ; t_i:T_m ; ts: ... \} \)
   Record \( OVec \) (op:opcode) \( ::= \{ t_{rpc} : T_{pc} ; t_r : ... \} \)
   \( \text{transfer} : (iv:IVec) -> \text{option (OVec (op iv))} \)

Definition \( \text{transfer} \) iv :=
   match iv with
   \[
   | \{ \text{op=Load}; \ t_{pc} = \text{ptr}(c_{pc}); \ t_i = M(c_{pc},i); \ ts = [\text{ptr}(c); M(c,T_v)] \} \\
     \Rightarrow \{ t_{rpc} = \text{ptr}(c_{pc}); \ t_r = T_v \} \\
   | \{ \text{op=Store}; \ t_{pc} = \text{ptr}(c_{pc}); \ t_i = M(c_{pc},i); \ ts = [\text{ptr}(c); T_v; M(c,T_v')] \} \\
     \Rightarrow \{ t_{rpc} = \text{ptr}(c_{pc}); \ t_r = M(c,T_v) \}
   \]
   ...

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} := \{ \text{op:opcode} ; t_{pc} : T_{pc} ; t_i : T_m ; ts : \ldots \} \]
Record \textbf{OVec} (op:opcode) := \{ t_{rpc} : T_{pc} ; t_r : \ldots \} \]

\textbf{transfer} : (iv : IVec) -> option (OVec (op iv))

3. Monitor services
Record \textbf{service} := \{ addr : word; sem : state -> option state; \ldots \} \]

Definition \textbf{mem\_safety\_services} : list service :=
\texttt{[malloc; free; base; size; eq]}.  


Memory safe abstract machine

Symbolic machine

Concrete machine

Micro-policy

Rule cache

Monitor

Generic Framework

Verification

micro-policy

correctly implements*

memory safety monitor

memory safety micro-policy

*only proved for IFC [POPL 2014]

ASM
Abstract machine for \( P \)

Symbolic machine

Concrete machine

Micro-policy

Monitor

Rule cache

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

secure

refinement (data)

preserved by

secure

(e.g. noninterferent)
Future

• Interaction with loader and compiler (static + dynamic)
  – Fully abstract compilation to micro-policies (Yannis, intern 2015)
• ... and operating system (e.g. protect the OS itself)
• Micro-policy composition, formally
• Language for writing micro-policies (symbolic rules)
• Verification for real RISC instruction set (e.g. ARM)
• More realistic processor (our-of-order execution, multi-core)
• Concurrency (big can of worms, data race detection)
• More micro-policies (e.g. stack protection, ...)
• Formally study expressive power of micro-policies
• Switch to F* for the proofs