Micro-Policies

Formally Verified Low-Level Tagging Schemes for Safety and Security

Cătălin Hrițcu

INRIA Paris-Rocquencourt
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!
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!
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:
  tons of vulnerabilities in every large system
  – violations of known safety and security policies
Micro-Policies

• general efficient dynamic enforcement mechanism for
  – critical invariants of low-level code
  – high-level abstractions and programming models
Micro-Policies

• general efficient dynamic enforcement mechanism for
  – critical invariants of low-level code
  – high-level abstractions and programming models
• add large metadata **tags** to all machine words
  – “this word is an instruction, and this one is a pointer”
  “this word comes from the net, and this one is private”
tag structure defined entirely by software
Micro-Policies

- general efficient dynamic enforcement mechanism for
  - critical invariants of low-level code
  - high-level abstractions and programming models
- add large metadata **tags** to all machine words
  - “this word is an instruction, and this one is a pointer”
    - “this word comes from the net, and this one is private”
  - tag structure defined entirely by software
  - tags efficiently propagated on each instruction
    - **rules defined by software** (fault handler; verified)
    - rule lookup **accelerated by hardware** rule cache
Micro-Policies for ...

We already thoroughly explored:  dynamic information flow control (IFC)
Micro-Policies for ...

We already thoroughly explored: dynamic information flow control (IFC)

Currently exploring:

user-kernel distinction

• hardware types
  – int vs. pointer vs. instruction

memory safety
  – stop all spatial and temporal violations on heap and stack
  – pointers become capabilities

control-flow integrity

• call-stack protection

• opaque closures
  – first-class functions ($\lambda$)

• linear pointers
  – absence of aliasing
Micro-Policies for ...

We already thoroughly explored: dynamic information flow control (IFC)

Currently exploring:
user-kernel distinction
• hardware types
  – int vs. pointer vs. instruction
memory safety
  stop all spatial and temporal violations on heap and stack
  – pointers become capabilities
control-flow integrity
• call-stack protection
• opaque closures
  – first-class functions (λ)
• linear pointers
  – absence of aliasing

Longer term plans:
pointer permissions
  – "readable", "writeable", "jumpable", or "callable"
• process isolation
  – replacement for virtual memory
dynamic type tags
  – for C, Scheme, or even OCaml
dynamic sealing & trademarks
  cache result of dynamic contracts
• higher-order contracts
• data race detection
• user-defined metadata
Micro-Policies for ...

We already thoroughly explored: dynamic information flow control (IFC)

Currently exploring:

- user-kernel distinction
- memory safety
  - stop all spatial and temporal violations on heap and stack
  - pointers become capabilities
- control-flow integrity
- hardware types
  - int vs. pointer vs. instruction
- call-stack protection
- opaque closures
  - first-class functions ($\lambda$)
- linear pointers
  - absence of aliasing
- longer term plans:
  - pointer permissions
    - "readable", "writeable", "jumpable", or "callable"
  - process isolation
    - replacement for virtual memory
  - dynamic type tags
    - for C, Scheme, or even OCaml
  - dynamic sealing & trademarks
    - cache result of dynamic contracts
  - higher-order contracts
  - data race detection
  - user-defined metadata
IFC Micro-Policy

<table>
<thead>
<tr>
<th>opcode</th>
<th>allow</th>
<th>$e_{rpc}$</th>
<th>$e_{r}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub</td>
<td>TRUE</td>
<td>LAB$_pc$</td>
<td>LAB$_1 \sqcup$ LAB$_2$</td>
</tr>
<tr>
<td>output</td>
<td>TRUE</td>
<td>LAB$_pc$</td>
<td>LAB$_1 \sqcup$ LAB$_pc$</td>
</tr>
<tr>
<td>push</td>
<td>TRUE</td>
<td>LAB$_pc$</td>
<td>BOT</td>
</tr>
<tr>
<td>load</td>
<td>TRUE</td>
<td>LAB$_pc$</td>
<td>LAB$_1 \sqcup$ LAB$_2$</td>
</tr>
<tr>
<td>store</td>
<td>LAB$_1 \sqcup$LAB$_pc \subset$ LAB$_3$</td>
<td>LAB$_pc$</td>
<td>LAB$_1 \sqcup$ LAB$_2 \sqcup$ LAB$_pc$</td>
</tr>
<tr>
<td>jump</td>
<td>TRUE</td>
<td>LAB$_1 \sqcup$ LAB$_pc$</td>
<td>--</td>
</tr>
<tr>
<td>bnz</td>
<td>TRUE</td>
<td>LAB$_1 \sqcup$ LAB$_pc$</td>
<td>--</td>
</tr>
<tr>
<td>call</td>
<td>TRUE</td>
<td>LAB$_1 \sqcup$ LAB$_pc$</td>
<td>LAB$_pc$</td>
</tr>
<tr>
<td>ret</td>
<td>TRUE</td>
<td>LAB$_1$</td>
<td>--</td>
</tr>
</tbody>
</table>

- A Verified Information Flow Architecture [POPL 2014]
- Testing Noninterference, Quickly [ICFP 2013]
- All Your IFCException Are Belong To Us [S&P 2013]
- A Theory of Information-Flow Labels [CSF 2013]
Noninterference proof in Coq [POPL 2014]
Noninterference proof in Coq [POPL 2014]

Abstract IFC Machine

Quasi-Abstract IFC Machine

Concrete Machine

satisfies noninterference
Noninterference proof in Coq [POPL 2014]

Abstract IFC Machine

Quasi-Abstract IFC Machine
- Rule table
- Interpret

Concrete Machine
- Rule cache
- Fault handler
- Compile

satisfies noninterference

bisimulation
Noninterference proof in Coq [POPL 2014]
Noninterference proof in Coq [POPL 2014]

Abstract IFC Machine

Quasi-Abstract IFC Machine

Concrete Machine

satisfies noninterference

bisimulation

correct compilation

parse compile interpret
Noninterference proof in Coq [POPL 2014]
Noninterference proof in Coq [POPL 2014]

Abstract IFC Machine

Quasi-Abstract IFC Machine

Concrete Machine

interpret

compile

satisfies noninterference

preserved by

bisimulation

correct compilation

satisfies noninterference
Memory safety

• Goal: prevent all memory safety violations
  – spatial violations: accessing arrays out of bounds
  – temporal violations: dereferencing pointer after its region was freed
    – for simplicity here only for heap-allocated data and excluding unpacked C structs

• Pointers become unforgeable capabilities
  – can only obtain a valid pointer to a memory region
    • by allocating that region or
    • by copying or offsetting an existing pointer to that region
Micro-Policy for memory safety

Tag := V(Type) | M(n,Type) | F
Type := i | ptr(n)
Micro-Policy for memory safety

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

\[
\text{Tag} := \text{V}(\text{Type}) \mid \text{M}(n,\text{Type}) \mid \text{F} \\
\text{Type} := \text{i} \mid \text{ptr}(n)
\]
Micro-Policy for memory safety

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

Tag := V(Type) | M(n,Type) | F
Type := i | ptr(n)
Micro-Policy for memory safety

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

\[ p = \text{A8F0} \]

Tag := \( V(\text{Type}) \mid M(n,\text{Type}) \mid F \)

Type := \( i \mid \text{ptr}(n) \)
Micro-Policy for memory safety

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

fresh \( n \)

\[ p = A8F0 \]

\[ \text{Tag := } V(\text{Type}) \mid M(n,\text{Type}) \mid F \]

\[ \text{Type := } i \mid \text{ptr}(n) \]
Micro-Policy for memory safety

\[ \text{p} \leftarrow \text{alloc k} \]
\[ \text{fresh n} \]

\[ \text{0@M(n,i) } \]
\[ \text{0@M(n,i) } \]
\[ \text{...} \]
\[ \text{0@M(n,i) } \]

\[ \text{Tag := V(Type) } | \text{ M(n,Type) } | \text{ F} \]
\[ \text{Type := i } | \text{ ptr(n)} \]
Micro-Policy for memory safety

\[ p \leftarrow \text{alloc } k \]
\[ \text{fresh } n \]

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

Tag := V(Type) | M(n,Type) | F
Type := i | ptr(n)
Micro-Policy for memory safety

\[ p = \text{A8F0} \oplus \text{V} (\text{ptr}(n)) \]

\[ q \leftarrow p + 1 \]

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)
Micro-Policy for memory safety

\[ p = A8F0 \text{V}(\text{ptr}(n)) \]
\[ A8F1 \text{V}(\text{ptr}(n)) = q \]

Tag := V(Type) | M(n,Type) | F
Type := i | ptr(n)
Micro-Policy for memory safety

\[ q \leftarrow p + k \]

\[
p = A8F0@V(ptr(n)) \quad A8F1@V(ptr(n)) = q
\]

\[
Tag := V(Type) \mid M(n,Type) \mid F
\]

\[
Type := i \mid ptr(n)
\]
### Micro-Policy for memory safety

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>...</th>
<th>k-1</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>0@M(n,i)</td>
<td>0@M(n,i)</td>
<td>...</td>
<td>0@M(n,i)</td>
<td>7@M(n',i)</td>
</tr>
</tbody>
</table>

\[ p = A8F0@V(ptr(n)) \]

\[ A8FK@V(ptr(n)) = q \]

\[ q \leftarrow p + k \]

**Tag**: \( V(Type) | M(n-Type) | F \)

**Type**: \( i | ptr(n) \)
# Micro-Policy for memory safety

$$p = A8F0\overline{V(ptr(n))}$$  
$$A8FK\overline{V(ptr(n))} = q$$

$$x \leftarrow !p$$

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>...</th>
<th>k-1</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0@M(n,i)$</td>
<td>$0@M(n,i)$</td>
<td>...</td>
<td>$0@M(n,i)$</td>
<td>$7@M(n',i)$</td>
</tr>
</tbody>
</table>

Tag := $V(\text{Type})$ | $M(n,\text{Type})$ | $F$

Type := $i$ | $\text{ptr}(n)$
Micro-Policy for memory safety

\[ p = A8F0 @ V(ptr(n)) \]

\[ A8FK @ V(ptr(n)) = q \]

\[ q \leftarrow p + k \]

\[ x \leftarrow !p \]

\[ \text{same } n \]

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)
Micro-Policy for memory safety

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

\[ q \leftarrow p + k \]

\[ x \leftarrow \neg p \]

\[ x = 0 @ V(i) \]

\[ \text{Tag} := V(\text{Type}) \mid M(n, \text{Type}) \mid F \]

\[ \text{Type} := i \mid \text{ptr}(n) \]
Micro-Policy for memory safety

\[ p = A8F0 \oplus V(\text{ptr}(n)) \]

\[ q = p + k \]

\[ q \leftarrow 42 \]

Tag := V(Type) | M(n,Type) | F
Type := i | ptr(n)
Micro-Policy for memory safety

\[ p = A8F0@V(ptr(n)) \]

\[ A8FK@V(ptr(n)) = q \]

\[ !q \leftarrow 42 \]

\[ n \neq n' \]

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)
Micro-Policy for memory safety

\[ p = A8F0@V(ptr(n)) \]

\[ A8FK@V(ptr(n)) = q \]

\[ q \leftarrow p + k \]

\[ n \neq n' \]

Out of bounds

\[ Tag := V(Type) \mid M(n,Type) \mid F \]

\[ Type := i \mid ptr(n) \]
Micro-Policy for memory safety

$p = \text{A8F0}\text{V}(\text{ptr}(n))$

$q \leftarrow p + k$

$q \not= 42$

Free $p$

Tag := $V$(Type) | $M$(n,Type) | F

Type := $i$ | $\text{ptr}(n)$
Micro-Policy for memory safety

\[ p = A8F0 \oplus V(ptr(n)) \]

\[ A8FK @ V(ptr(n)) = q \]

\[ q \leftarrow p + k \]

\[ !q \leftarrow 42 \]

out of bounds

\[ \text{free } p \]

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)
Micro-Policy for memory safety

\[ p = A8F0\text{V(ptr(n))} \]

\[ A8FK\text{V(ptr(n))} = q \]

\[ q \leftarrow p + k \]

\[ !q \times 42 \]

Out of bounds

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)
Micro-Policy for memory safety

\[ p = A8F0 @ V(ptr(n)) \]

\[ A8FK @ V(ptr(n)) = q \]

\[ q \leftarrow p + k \]

\[ \text{free } p \]

\[ x \leftarrow !p \]

\[ \text{Tag := } V(\text{Type}) \mid M(n, \text{Type}) \mid F \]

\[ \text{Type := } i \mid \text{ptr}(n) \]

out of bounds
Micro-Policy for memory safety

\[ p = A8F0 \oplus V(ptr(n)) \]

\[ q \leftarrow p + k \]

\[ \neg q \leftarrow 42 \]

out of bounds

Tag := V(Type) | M(n,Type) | F

Type := i | ptr(n)

free p

use after free
Direction of this project

• **Beyond IFC:**
  – show generality: study diverse set of micro-policies
  – formally verify enforced properties
  – implement and evaluate practical viability

• **Beyond clean-slate (CRASH/SAFE):**
  – targeting a stock RISC architecture
  – extended with tags and a rule cache
  – legacy software with little or no changes
Future challenges

• **Micro-policy composition**
  – hardware supports compound tags
  – but policies are often not orthogonal (e.g. tags can leak information)
  – this is not just reference monitoring / safety properties
    • “micro-calls” into privileged code can inspect tags
    • policy violations are often recoverable
  – sequential (vertical) vs. parallel (cross product)
  – further improve efficiency

• **Meta-language for micro-policies**
  – beyond disparate DSLs
Collaborators on this project*

UPenn
Arthur Azevedo de Amorim**
Maxime Denes
Leonidas Lampropoulos
Benoit Montagu
Benjamin Pierce
Antal Spector-Zabusky

INRIA Paris
Nick Giannarakis**
Cătălin Hrițcu

Portland State
Nathan Collins
Andrew Tolmach

IRISA Rennes
Delphine Demange
David Pichardie

Harvard
Greg Morrisett
Randy Pollack

* Started part of DARPA CRASH/SAFE
** Soon interns at INRIA Paris
Collaborators on this project*

Formal side

UPenn
- Arthur Azevedo de Amorim**
- Maxime Denes
- Leonidas Lampropoulos
- Benoit Montagu
- Benjamin Pierce
- Antal Spector-Zabusky

INRIA Paris
- Nick Giannarakis**
- Cătălin Hrițcu

Portland State
- Nathan Collins
- Andrew Tolmach

IRISA Rennes
- Delphine Demange
- David Pichardie

Harvard
- Greg Morrisett
- Randy Pollack

Architecture side

UPenn
- Andre DeHon
- Udit Dhawan
- Ben Karel
- Nikos Vasilakis
- Jonathan M. Smith

MIT
- Tom Knight
- Howard Shrobe

BAE Systems
- Greg Sullivan

...

* Started part of DARPA CRASH/SAFE
** Soon interns at INRIA Paris
My other two current projects

• **QuickChick**: Speeding up Formal Proofs with Property-Based Testing
  – General Framework for Polarized Mutation Testing
  – Language for Custom Test-Data Generators
  – Deep Integration with Coq/SSReflect

• **νF**: Next Generation Security Type Checker
  – Better refinement type inference (Dijkstra monad)
  – Beyond value-dependency
  – Better control of effects (including termination)
  – Smarter (semantic) termination checking
THANK YOU
BACKUP SLIDES
Computer systems are insecure
Computer systems are insecure

Why?
Computer systems are insecure

<table>
<thead>
<tr>
<th>Rank</th>
<th>Score</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>93.8</td>
<td>CWE-89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL injection')</td>
</tr>
<tr>
<td>[2]</td>
<td>83.3</td>
<td>CWE-78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
</tr>
<tr>
<td>[3]</td>
<td>79.0</td>
<td>CWE-120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
</tr>
<tr>
<td>[4]</td>
<td>77.7</td>
<td>CWE-79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
</tr>
<tr>
<td>[6]</td>
<td>76.8</td>
<td>CWE-862</td>
<td>Missing Authorization</td>
</tr>
<tr>
<td>[7]</td>
<td>75.0</td>
<td>CWE-798</td>
<td>Use of Hard-coded Credentials</td>
</tr>
<tr>
<td>[8]</td>
<td>75.0</td>
<td>CWE-311</td>
<td>Missing Encryption of Sensitive Data</td>
</tr>
<tr>
<td>[9]</td>
<td>74.0</td>
<td>CWE-434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>[10]</td>
<td>73.8</td>
<td>CWE-307</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
</tr>
<tr>
<td>[11]</td>
<td>73.1</td>
<td>CWE-250</td>
<td>Execution with Unnecessary Privileges</td>
</tr>
<tr>
<td>[12]</td>
<td>70.1</td>
<td>CWE-352</td>
<td>Cross-Site Request Forgery ('CSRF')</td>
</tr>
<tr>
<td>[13]</td>
<td>69.3</td>
<td>CWE-22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
</tr>
<tr>
<td>[14]</td>
<td>68.5</td>
<td>CWE-494</td>
<td>Download of Code Without Integrity Check</td>
</tr>
<tr>
<td>[16]</td>
<td>66.0</td>
<td>CWE-829</td>
<td>Inclusion of Functionality from Untrusted Control Sphere</td>
</tr>
<tr>
<td>[17]</td>
<td>65.5</td>
<td>CWE-732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
</tr>
<tr>
<td>[18]</td>
<td>64.6</td>
<td>CWE-676</td>
<td>Use of Potentially Dangerous Function</td>
</tr>
<tr>
<td>[19]</td>
<td>64.1</td>
<td>CWE-327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
</tr>
<tr>
<td>[20]</td>
<td>62.4</td>
<td>CWE-131</td>
<td>Incorrect Calculation of Buffer Size</td>
</tr>
<tr>
<td>[21]</td>
<td>61.5</td>
<td>CWE-307</td>
<td>Improper Restriction of Excessive Authentication Attempts</td>
</tr>
<tr>
<td>[22]</td>
<td>61.1</td>
<td>CWE-601</td>
<td>URL Redirection to Untrusted Site ('Open Redirect')</td>
</tr>
<tr>
<td>[23]</td>
<td>61.0</td>
<td>CWE-134</td>
<td>Uncontrolled Format String</td>
</tr>
<tr>
<td>[24]</td>
<td>60.3</td>
<td>CWE-190</td>
<td>Integer Overflow or Wraparound</td>
</tr>
<tr>
<td>[25]</td>
<td>59.9</td>
<td>CWE-759</td>
<td>Use of a One-Way-Hash without a Salt</td>
</tr>
</tbody>
</table>

Source: 2011 CWE/SANS Top 25 Most Dangerous Software Errors
Micro-Policy for memory safety

• Tag := Val(Type) | Mem(n,Type) | Free  Type := Int | Ptr(n)

• allocation:
  – generate fresh n
  – initialize region with 0@Mem(n,Int)
  – return <pointer-to-region>@Val(Ptr(n))

• memory access (read/write):
  – check that pointer tagged @Val(Ptr(n))
  – check that referenced location tagged @Mem(n,Type)
  – on memory read tag result with @Val(Type)
  – when writing w@Val(NType) retag location with @Mem(n,Type)

• reclaiming memory (free):
  – check that pointer and referenced location have the same n
  – overwrite region with 0@Free
Formal verification side

• Verification of low-level code
  – bisimulation/refinement
  – verified structured code generators