Secure Compilation to Isolated Assembly

Marco Patrignani

\(^1\)iMinds-DistriNet, Dept. Computer Science, KU Leuven, Belgium

first.last@cs.kuleuven.be

7 April 2014
Goal of the Talk

- introduce my research on secure compilation
Goal of the Talk

- introduce my research on secure compilation
- define secure compilation and related notions
Goal of the Talk

- introduce my research on secure compilation
- define secure compilation and related notions
- point out open challenges
Outline

1. **Background**
   - PMA and Isolation
   - Secure Compilation: Motivations

2. **Secure Compilation of Java Jr**
   - Source Language
   - Secure Compilation, Informally
   - Proof Strategy
   - Fully Abstract Trace Semantics for PMA

3. **Open Challenges**
   - Multilanguage Model
   - Multi-principal Languages
   - Multithreaded Languages
   - Sky is the Limit
Outline

1. Background
   - PMA and Isolation
   - Secure Compilation: Motivations

2. Secure Compilation of Java Jr
   - Source Language
   - Secure Compilation, Informally
   - Proof Strategy
   - Fully Abstract Trace Semantics for PMA

3. Open Challenges
   - Multilanguage Model
   - Multi-principal Languages
   - Multithreaded Languages
   - Sky is the Limit
Why is Protected Modules Architecture (PMA) Interesting?

- it provides deep encapsulation at the lowest level of abstraction
Why is Protected Modules Architecture (PMA) Interesting?

- it provides deep encapsulation at the lowest level of abstraction
- it is the basis of several security-related works
Why is Protected Modules Architecture (PMA) Interesting?

- it provides deep encapsulation at the lowest level of abstraction
- it is the basis of several security-related works
- Intel wants to port it to future processors
What is a Protected Modules Architecture

- assembly-level isolation mechanism
What is a Protected Modules Architecture

- assembly-level isolation mechanism
- implemented via Hypervisor, Hardware, Software
What is a Protected Modules Architecture

- assembly-level isolation mechanism
- implemented via Hypervisor, Hardware, Software
- several research and industrial prototypes
What is a Protected Modules Architecture

- assembly-level isolation mechanism
- implemented via Hypervisor, Hardware, Software
- several research and industrial prototypes

**Q:** How does PMA work?
PMA in action

memory space

```
0x0001   call 0xb53
0x0002   movs r0 0x0b55

0x0b52   movs r0 0x0b55
0x0b53   call 0x0002
0x0b54   movs r0 0x0001
0x0b55   ...

0xab00   jmp 0xb53
0xab01   ...
```
PMA in action

Memory space

Protected module = protected memory

```
0x0001  call 0xb53
0x0002  movs r0 0x0b55

0x0b52  movs r0 0x0b55
0x0b53  call 0x0002
0x0b54  movs r0 0x0001
0x0b55  ...

0xab00  jmp 0xb53
0xab01  ...
```
PMA in action

0x0001 call 0xb53
0x0002 movs r0 0x0b55

0x0b52 movs r0 0x0b55
0x0b53 call 0x0002
0x0b54 movs r0 0x0001
0x0b55 ...

0xab00 jmp 0xb53
0xab01 ...

- memory space
- protected module = protected memory
- split in code and data
PMA in action

```
0x0001  call 0xb53
0x0002  movs r0 0x0b55

0x0b52  movs r0 0x0b55
0x0b53  call 0x0002
0x0b54  movs r0 0x0001
0x0b55  ...

0xab00  jmp 0xb53
0xab01  ...
```

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
PMA in action

0x0001  call 0xb53
0x0002  movs r0 0xb55

0x0b52  movs r0 0xb55
0x0b53  call 0x0002
0x0b54  movs r0 0x0001

0x0b55  ...

0xab00  jmp 0xb53
0xab01  ...

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
PMA in action

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0001</td>
<td>call 0xb53</td>
</tr>
<tr>
<td>0x0002</td>
<td>movs r0 0xb55</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>0xb52</td>
<td>movs r0 0xb55</td>
</tr>
<tr>
<td>0xb53</td>
<td>call 0x0002</td>
</tr>
<tr>
<td>0xb54</td>
<td>movs r0 0x0001</td>
</tr>
<tr>
<td>0xb55</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ab00</td>
<td>jmp 0xb53</td>
</tr>
<tr>
<td>ab01</td>
<td></td>
</tr>
</tbody>
</table>

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
PMA in action

```
0x0001 call 0xb53
0x0002 movs r0 0x0b55
...
0x0b52 movs r0 0x0b55
0x0b53 call 0x0002
0x0b54 movs r0 0x0001
0x0b55 ...
```

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted

Marco Patrignani
PMA in action

0x0001 call 0xb53
0x0002 movs r0 0x0b55
...
0xab00 jmp 0xb53
0xab01 ...

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted

Marco Patrignani
PMA in action

0x0001  call 0xb53
0x0002  movs r0 0x0b55
          ...

0x0b52  movs r0 0x0b55
0x0b53  call 0x0002
0x0b54  movs r0 0x0001
0x0b55  ...

0xab00  jmp 0xb53
0xab01  ...

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted
PMA in action

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted
- entry points for communication (■)

```
0x0001 call 0xb53
0x0002 movs r 0x0002
...  
0x0b52 movs r 0x0b53
0x0b53 call 0x0002
0x0b54 movs r 0x0b55
0x0b55 ...  
...  
0xab00 jmp 0xb53
0xab01 ...  
```
PMA in action

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted
- entry points for communication (■)
Benefits of Secure Compilation

- Secure compilation preserves source-level abstractions in target-level languages.
- Protects against code injection attacks.
- Enables source-level reasoning.

Marco Patrignani
Benefits of Secure Compilation

Secure compilation preserves source-level abstractions in target-level languages.
Secure compilation preserves source-level abstractions in target-level languages

- protect against code injection attacks
Benefits of Secure Compilation

Secure compilation preserves source-level abstractions in target-level languages

- protect against code injection attacks
- enables source-level reasoning
Outline

1 Background
   - PMA and Isolation
   - Secure Compilation: Motivations

2 Secure Compilation of Java Jr
   - Source Language
   - Secure Compilation, Informally
   - Proof Strategy
   - Fully Abstract Trace Semantics for PMA

3 Open Challenges
   - Multilanguage Model
   - Multi-principal Languages
   - Multithreaded Languages
   - Sky is the Limit
Languages of the Compiler

- source language: ±/− Java jr
Languages of the Compiler

- source language: +/- Java jr
  - component-based
  - private fields
  - programming to an interface
  - exceptions
Languages of the Compiler

- source language: +/- Java jr
  - component-based
  - private fields
  - programming to an interface
  - exceptions

```
package PI;

interface Account {
    public createAccount() : Foo;
}

extern extAccount : Account;

package PE;

class AccountClass implements PI.Account {
    AccountClass() { counter = 0; }
    public createAccount() : Account {
        return new PE.AccountClass();
    }

    private counter : Int;
}

object extAccount : AccountClass;
```
Languages of the Compiler

- source language: +/- Java jr
  - component-based
  - private fields
  - programming to an interface
  - exceptions

Q: How to securely compile this code?

```java
package PI;
interface Account {
    public createAccount() : Foo;
}
extern extAccount : Account;

package PE;
class AccountClass implements PI.Account {
    AccountClass() { counter = 0; }
    public createAccount() : Account {
        return new PE.AccountClass();
    }
    private counter : Int;
}
object extAccount : AccountClass;
```
package PI;

interface Account {
  public createAccount() : Foo;
}

extern extAccount : Account;

package PE;

class AccountClass implements PI.Account {
  AccountClass() { counter = 0; }
  public createAccount() : Account {
    return new PE.AccountClass();
  }

  private counter : Int;
}

object extAccount : AccountClass;
Languages of the Compiler

Dynamic dispatch

v-tables

Secure stack

package PI;
interface Account {
    public createAccount() : Foo;
}
extern extAccount : Account;

package PE;
class AccountClass implements PI.Account {
    AccountClass() { counter = 0; }
    public createAccount() : Account {
        return new PE.AccountClass();
    }
    private counter : Int;
}
object extAccount : AccountClass;
Languages of the Compiler

### Secure Compilation of Java Jr

- **Open Challenges**
  - Secure Compilation, Informally
  - Proof Strategy
  - Fully Abstract Trace Semantics for PMA

#### Source Language

#### Dynamic dispatch
- v-tables
- Secure stack

#### Proxy to createAccount

```java
package PI;

interface Account {
    public createAccount() : Foo;
}

extern extAccount : Account;

package PE;

class AccountClass implements PI.Account {
    AccountClass() { counter = 0; }
    public createAccount() : Account {
        return new PE.AccountClass();
    }

    private counter : Int;
}

object extAccount : AccountClass;
```
Languages of the Compiler

- proxy to createAccount
- createAccount body
- constructor
- Dynamic dispatch
- v-tables
- Secure stack
- extAccount
- counter

```
package PI;

interface Account {
    public createAccount() : Foo;
}

extern extAccount : Account;

package PE;

class AccountClass implements PI.Account {
    AccountClass() { counter = 0; }
    public createAccount() : Account {
        return new PE.AccountClass();
    }

    private counter : Int;
}

object extAccount : AccountClass;
```
Secure Compilation, Informally

Source level

O1
O2

Target level

Ext 1
Ext 2

Protect against low-level attackers
Target code is vulnerable without PMA
Secure Compilation, Informally

O1.createAccount()  ➔  Ext 1

Source level

O1  ➔  Ext 1
O2  ➔  Ext 2

Target level
Secure Compilation, Informally

Source level

O1
O2
O3

Ext 1
Ext 2

Source Language
Secure Compilation of Java Jr
Open Challenges

Proof Strategy
Fully Abstract Trace Semantics for PMA

Protect against low-level attackers
Target code is vulnerable without PMA
Secure Compilation, Informally

Source level

O1
O2
O3

return O3

Ext 1
Ext 2

Target level

Protect against low-level attackers

Target code is vulnerable without PMA
Secure Compilation, Informally

Source level

O1
O2
O3

O3.counter

⇓

Ext 1
Ext 2

Target level

Ext 1
Ext 2

Protect against low-level attackers

Target code is vulnerable without PMA
Secure Compilation, Informally

Source level

O1
O2
O3

03.counter

Ext 1
Ext 2

Target level

Ext 1
Ext 2

Protect against low-level attackers

Target code is vulnerable without PMA
Secure Compilation, Informally

Source level

O1
O2
O3

→

Target level

Ext 1
Ext 2

Protect against low-level attackers
Target code is vulnerable without PMA
Secure Compilation, Informally

Source level
O1
O2
O3

Target level
O1↓
O2↓

Ext 1
Ext 2

Ext 1↓
Ext 2↓
Secure Compilation, Informally

Source level

O1
O2
O3

Target level

O1↓.createAccount()
O2↓
O3↓

Ext 1↓
Ext 2↓
Ext 1
Ext 2

Protect against low-level attackers
Target code is vulnerable without PMA
Secure Compilation, Informally

Source level

O1
O2
O3

Target level

O1
O2
O3

\[ \text{return} \; O3 \]

Ext 1
Ext 2

Protect against low-level attackers
Target code is vulnerable without PMA
Secure Compilation, Informally

Source level

O1
O2
O3

Target level

O1\downarrow
O2\downarrow
O3\downarrow

Ext 1
Ext 2
Ext 1\downarrow
Ext 2\downarrow

Protect against low-level attackers

Target code is vulnerable without PMA

Marco Patrignani
Secure Compilation to Isolated Assembly
Secure Compilation, Informally

- Protect against low-level attackers

Source level:
- O1
- O2
- O3

Target level:
- O1
- O2
- O3

Ext 1
Ext 2
Ext 1
Ext 2
Secure Compilation, Informally

- Protect against low-level attackers
- Target code is vulnerable without PMA
Q: : Is that all?
Q: : Is that all?

- protected stack

0x0001 Unprotected stack
0x0002
:
0x0b52
0x0b53
0x0b54
0x0b55
Q: Is that all?

- protected stack

Leaks stack contents:
- 0x0b52
- 0x0b53
- 0x0b54
- 0x0b55
Q: Is that all?

- protected stack

```
0x0001 Unprotected stack
0x0002
... 0xb52
0xb53 Protected stack
0xb54
0xb55
```

Leaks stack contents
Q: : Is that all?

- protected stack
- returnback entry point

```
0x0001 Unprotected stack
0x0002
...
0x0b52
0x0b53 Protected stack
0x0b54
0x0b55
```

Marco Patrignani
Q: Is that all?

- protected stack
- returnback entry point
Q: Is that all?

- protected stack
- returnback entry point
Agten's Work

Q: Is that all?

- protected stack
- returnback entry point
- reset flags and registers

0x0001 Unprotected stack
0x0002
:

0x0b52
0x0b53 Protected stack
0x0b54
0x0b55
Agten’s Work

Q: : Is that all?

- protected stack
- returnback entry point
- reset flags and registers

```
0x0001 Unprotected stack
0x0002
:
0x0b52
0x0b53 Protected stack
0x0b54 je ...
0x0b55
```
Q: Is that all?

- protected stack
- returnback entry point
- reset flags and registers

\[ f_{ZS} = 0/1 \]
Q: Is that all?

- protected stack
- returnback entry point
- reset flags and registers
Q: Is that all?

- protected stack
- returnback entry point
- reset flags and registers
- ground-typed values check

0x0001 Unprotected stack
0x0002
...
0xb52
0xb53 Protected stack
0xb54
0xb55
Agten’s Work

Q: Is that all?

- protected stack
- returnback entry point
- reset flags and registers
- ground-typed values check

- 0x0001 Unprotected stack
- 0x0002
- ...  
- 0xb50 Unprotected stack
- 0xb52
- 0xb53 Protected stack  
- 0xb54
- 0xb55
Agten’s Work

Q: Is that all?

- protected stack
- returnback entry point
- reset flags and registers
- ground-typed values check

```
0x0001 Unprotected stack
0x0002
...
0xb52
0xb53 Protected stack
0xb54
0xb55
```

check \( r_0 = 1/0 \)
Agten’s Work

Q: Is that all?

- protected stack
- returnback entry point
- reset flags and registers
- ground-typed values check

Q: Is there more?

0x0001 Unprotected stack
0x0002

0xb52
0xb53 Protected stack
0xb54
0xb55
Dynamic Memory Allocation

Source level

O1
O2

Target level

O1↓
O2↓

Ext 1
Ext 2

Ext 1↓
Ext 2↓
Dynamic Memory Allocation

Source level:
- O1
- O2

Target level:
- O1
- O2

O1.createAccount() → Ext 1

Ext 1
Ext 2

Object id guessing
map Oid to natural numbers
add Oid to map
lookup \(O(1)\) when number is received

dynamic typecheck for: current object
arguments
no need of extra information
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Target level

O1↓
O2↓

Ext 1
Ext 2

Ext 1↓
Ext 2↓

Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup (O(1)) when number is received

Dynamic typecheck for: current object arguments
No need of extra information
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Target level

O1↓
O2↓

return O3

⇒

Ext 1
Ext 2

⇒

Ext 1↓
Ext 2↓
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Target level

O1
O2

01\downarrow .createAccount()
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Target level

O1↓
O2↓
O4↓
O3↓

Ext 1
Ext 2

Ext 1↓
Ext 2↓
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Target level

O1
O2
O4
O3

Ext 1
Ext 2

return O3

Ext 1
Ext 2

Object id guessing
map Oid to natural numbers
add Oid to map
lookup (O1) when number is received
dynamic typecheck for: current object
arguments
no need of extra information
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Ext 1
Ext 2

Target level

0x001
0x005
0x009
0x00C

return 0x00C

Ext 1
Ext 2

Object id guessing
map Oid to natural numbers
add Oid to map
lookup (O(1)) when number is received
dynamic typecheck for: current object arguments
no need of extra information
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Target level

0x001
0x005
0x009
0x00C

0x009.createAccount()

Ext 1
Ext 2

Ext 1
Ext 2

Object id guessing
map Oid to natural numbers
add Oid to map
lookup (O(1)) when number is received
dynamic typecheck for: current object
arguments
no need of extra information

Marco Patrignani
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Ext 1
Ext 2

Target level

0x001
0x005
0x009
0x00C

Ext 1↓
Ext 2↓
Dynamic Memory Allocation

Source level:
- O1
- O2
- O4
- O3

Target level:
- 0x001
- 0x005
- 0x009
- 0x00C

Object id guessing

O4.createAccount()
Dynamic Memory Allocation

Source level:
- O1
- O2
- O4
- O3

Target level:
- 0x001
- 0x005

Ext 1
- Object id guessing
- map Oid to natural numbers

Ext 2

- Ext 1↓
- Ext 2↓
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Target level

0x001 ↦ 1
0x005 ↦ 2

Ext 1
Ext 2

οbject id guessing
οbject oid to natural numbers

Ext 1↓
Ext 2↓
Dynamic Memory Allocation

Source level:
- O1
- O2
- O4
- O3

Ext 1
Ext 2

Target level:
- 0x001 → 1
- 0x005 → 2

1.createAccount()

• Object id guessing
• map Oid to natural numbers
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Ext 1
Ext 2

Target level

0x001 $\mapsto$ 1
0x005 $\mapsto$ 2
0x009
0x00C

Ext 1↓
Ext 2↓

- Object id guessing
- map Oid to natural numbers
Dynamic Memory Allocation

Source level

- O1
- O2
- O3
- O4

Target level

- Ext 1
- Ext 2

- 0x001 ↦ 1
- 0x005 ↦ 2
- 0x009
- 0x00C ↦ 3

- Object id guessing
- map Oid to natural numbers
- add Oid to map

Marco Patrignani
Secure Compilation to Isolated Assembly 12/30
Dynamic Memory Allocation

Source level

<table>
<thead>
<tr>
<th>O1</th>
<th>Ext 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>O2</td>
<td></td>
</tr>
<tr>
<td>O4</td>
<td></td>
</tr>
<tr>
<td>O3</td>
<td></td>
</tr>
</tbody>
</table>

Target level

\[0 \times 001 \mapsto 1\]
\[0 \times 005 \mapsto 2\]
\[0 \times 009 \mapsto 3\]

● Object id guessing
● map Oid to natural numbers
● add Oid to map
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Ext 1
Ext 2

Target level

0x001 $\mapsto$ 1
0x005 $\mapsto$ 2
0x009
0x00C $\mapsto$ 3

- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup (O(1)) when number is received

Marco Patrignani
Dynamic Memory Allocation

Source level

| O1 : Account |
| O2 : Pair   |
| O4          |
| O3          |

Target level

| 0x001 ⟷ 1 |
| 0x005 ⟷ 2 |
| 0x009     |
| 0x00C ⟷ 3 |

Ext 1

Ext 2
- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup \(O(1)\) when number is received
Dynamic Memory Allocation

Source level:

O1 : Account
O2 : Pair
O4
O3

Target level:

0x001 ⇔ 1
0x005 ⇔ 2
0x009
0x00C ⇔ 3

Ext 1:

02.createAccount()

Ext 2:

- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup (O(1)) when number is received
Dynamic Memory Allocation

Source level:
- O1: Account
- O2: Pair
- O4
- O3

Target level:
- 0x001 → 1
- 0x005 → 2
- 0x009
- 0x00C → 3

Ext 1
Ext 2

- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup (O(1)) when number is received
Dynamic Memory Allocation

Source level

O1 : Account
O2 : Pair
O4
O3

Target level

0x001 $\mapsto$ 1
0x005 $\mapsto$ 2
0x009
0x00C $\mapsto$ 3

Ext 1

Ext 2
- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup (O(1)) when number is received

 Marco Patrignani Secure Compilation to Isolated Assembly
Dynamic Memory Allocation

Source level

O1 : Account
O2 : Pair
O4
O3

Target level

0x001 \leftrightarrow 1
0x005 \leftrightarrow 2
0x009
0x00C \leftrightarrow 3

Ext 1

Ext 2

- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup (O(1)) when number is received

Ext 1\downarrow

Ext 2\downarrow

- dynamic typecheck for: current object
Dynamic Memory Allocation

Source level

O1 : Account
O2 : Pair
O4
O3

Target level

0x001 \mapsto 1
0x005 \mapsto 2
0x009
0x00C \mapsto 3

Ext 1
Ext 2

- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup \((O(1))\) when number is received
- dynamic typecheck for: current object arguments

Ext 1↓
Ext 2↓
Dynamic Memory Allocation

Source level:
- O1: Account
- O2: Pair
- O4
- O3

Target level:
- 0x001 $\mapsto$ 1
- 0x005 $\mapsto$ 2
- 0x009
- 0x00C $\mapsto$ 3

 Ext 1
- createAccount()

 Ext 2
- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup (O(1)) when number is received
- dynamic typecheck for: current object arguments
- no need of extra information

Object id guessing:
- map Oid to natural numbers
- add Oid to map
- lookup (O(1)) when number is received
- dynamic typecheck for: current object arguments
- no need of extra information
Exceptions

Stack

- $f_s$
- $g$
- $h_s$
- $i$
- $l_s$

throw e

Secure stack

- $f_s$
- $h_s$
- $l_s$

throw e

Insecure stack

- $g$
- $i$

Secure Compilation to Isolated Assembly
Exceptions

Stack

\[
\begin{align*}
&f_s \\
g \\
h_s \\
i \\
l_s
\end{align*}
\]

Secure stack

\[
\begin{align*}
&f_s \\
h_s \\
l_s
\end{align*}
\]

Insecure stack

\[
\begin{align*}
g \\
i
\end{align*}
\]

throw e
Exceptions

Stack

secure stack

Insecure stack

throw e

 Marco Patrignani Secure Compilation to Isolated Assembly 14/30
Exceptions

Stack

Secure stack

Insecure stack

f_s

g

hs

i

l_s

f_s

hs

i

Throw e

Throw e
Exceptions

Stack

Secure stack

Insecure stack

Marco Patrignani
Exceptions

Stack:
1. \( l_s \)
2. \( i \)
3. \( h_s \)
4. \( g \)
5. \( f_s \)

Secure stack:
- \( f_s \)
- \( h_s \)
- \( l_s \)

Insecure stack:
- \( g \)
- \( i \)

throw e
Exceptions

Stack:

- $f_s$
- $g$
- $h_s$
- $i$
- $l_s$

throw e

Secure stack:

- $f_s$
- $h_s$
- $l_s$

Insecure stack:

- $g$
- $i$

throw e
Exceptions

Secure stack:
- $f_s$
- $h_s$
- $l_s$

Insecure stack:
- $g$
- $i$

Stack:
1. $l_s$
2. $i$
3. $h_s$
4. $g$
5. $f_s$

throw $e$
Exceptions

Stack

1. \( l_s \)
2. \( i \)
3. \( h_s \)
4. \( g \)
5. \( f_s \)

Secure stack

1. \( l_s \)
2. \( h_s \)
3. \( f_s \)

Insecure stack

1. \( i \)
2. \( g \)

Record passed exceptions
Exceptions

Stack

Secure stack

Insecure stack

Record passed exceptions
Check that exception could be thrown
Exceptions

Stack

Secure stack

Insecure stack

Record passed exceptions
Check that exception could be thrown
Exceptions

Record passed exceptions
Check that exception could be thrown
So now...

- We have a strategy to securely compile Java jr code
So now...

- We have a strategy to securely compile Java jr code
- We have the tools to implement it
So now...

- We have a strategy to securely compile Java jr code
- We have the tools to implement it
- We have an idea of the security properties of our secure compilation scheme
So now...

- We have a strategy to securely compile Java jr code
- We have the tools to implement it
- We have an idea of the security properties of our secure compilation scheme

Q: What is missing?
We have a strategy to securely compile Java jr code
We have the tools to implement it
We have an idea of the security properties of our secure compilation scheme

What is missing?

A PROOF!
Secure Compilation, Formally

\[
C_1 \simeq_H C_2 \iff C_1^\uparrow \simeq_L C_2^\uparrow
\]
Secure Compilation, Formally

\[ C_1 \sim_H C_2 \iff C_1 \downarrow \sim_L C_2 \]
Secure Compilation, Formally

\[ C_1 \cong_H C_2 \iff C_1 \downarrow \cong_L C_2 \]

Contextual Equivalence
Contextual Equivalence

\[ C_1 \simeq C_2 \triangleq \forall C. \ C[C_1] \uparrow \iff C[C_2] \uparrow \]
Contextual Equivalence

\[ C_1 \simeq C_2 \iff \forall C. C[C_1] \uparrow \iff C[C_2] \uparrow \]
Contextual Equivalence

\[ C_1 \sim C_2 \triangleq \forall C. \uparrow C[C_1] \iff \uparrow C[C_2] \]

All contexts
Secure Compilation

\[ C_1 \simeq_H C_2 \iff C_1 \downarrow \simeq_L C_2 \downarrow \]
Secure Compilation

\[ C_1 \simeq_H C_2 \iff C_1 \uparrow \simeq_L C_2 \uparrow \]

\[(\forall C. \ C[C_1] \uparrow \iff C[C_2] \uparrow) \iff (\forall M. \ M[C_1] \uparrow \iff M[C_2] \uparrow)\]
Secure Compilation

\[ C_1 \simeq_H C_2 \iff C_1^\downarrow \simeq_L C_2^\downarrow \]

\[
(\forall C. C[C_1]^\uparrow \iff C[C_2]^\uparrow ) \iff (\forall M. M[C_1]^\uparrow \iff M[C_2]^\uparrow )
\]

VERY COMPLEX!
$C_1 \simeq_H C_2 \iff C_1 \downarrow \simeq_L C_2 \downarrow$
$C_1 \simeq_H C_2 \iff C_1^\uparrow \simeq_L C_2^\uparrow$
Secure Compilation

\[ C_1 \simeq_H C_2 \quad \checkmark \quad C_1 \downarrow \simeq_L C_2 \downarrow \]
$C_1 \simeq_H C_2 \Rightarrow C_1 \downarrow \simeq_L C_2 \downarrow$
Secure Compilation

\[ C_1 \simeq_H C_2 \quad \Rightarrow \quad C_1 \downarrow \simeq_L C_2 \downarrow \]

\[ \text{Traces}(C_1 \downarrow) = \text{Traces}(C_2 \downarrow) \]

Fully Abstract Trace Semantics
Secure Compilation

\[ C_1 \not\sim_H C_2 \iff \text{Traces}(C_1) \neq \text{Traces}(C_2) \]
Secure Compilation

\[ C_1 \not\equiv_H C_2 \quad \checkmark \quad \text{Traces}(C_1) \neq \text{Traces}(C_2) \]
Secure Compilation

\[
C_1 \not\equiv_H C_2 \quad \checkmark
\]

\[
C_1 \downarrow \cong_L C_2 \downarrow
\]

\[
\Leftrightarrow
\]

\[
\text{Traces}(C_1) = \text{Traces}(C_2)
\]

Fully Abstract Trace Semantics
Trace Semantics for PMA
Trace Semantics for PMA

 behaviour in this case is:

```
0x0001 call 0xb52
0x0002 ...

: 0xb52 movi r0, 1
0xb53 movi r1, 0xb56
0xb54 jl r1
0xb55 call 0xab01
0xb56 ret

: 0xab01 ...
```
Trace Semantics for PMA

behaviour in this case is: call in

```
0x0001  call 0xb52
0x0002  ...
  ...
```

```
0xb52  movi r0 1
0xb53  movi r1 0xb56
0xb54  jl  r1
0xb55  call 0xab01
0xb56  ret
  ...
```

```
0xab01  ...
```
Trace Semantics for PMA

behaviour in this case is: call in, \texttt{ret 1}

\begin{verbatim}
0x0001 call 0xb52
0x0002 ...

0xb52 movi r0 1
0xb53 movi r1 0xb56
0xb54 jl r1
0xb55 call 0xab01
0xb56 ret

0xab01 ...
\end{verbatim}
Trace Semantics for PMA

- behaviour in this case is: call in, ret 1
  or call in,
Trace Semantics for PMA

- behaviour in this case is: call in, ret 1
  or call in, call out
Trace Semantics for PMA

- behaviour in this case is: call in, ret 1 or call in, call out
- traces rely only on the PMA code

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0001</td>
<td>call 0xb52</td>
</tr>
<tr>
<td>0x0002</td>
<td>...</td>
</tr>
<tr>
<td>0xb52</td>
<td>movi r0 1</td>
</tr>
<tr>
<td>0xb53</td>
<td>movi r1 0xb56</td>
</tr>
<tr>
<td>0xb54</td>
<td>jl r1</td>
</tr>
<tr>
<td>0xb55</td>
<td>call 0xab01</td>
</tr>
<tr>
<td>0xb56</td>
<td>ret</td>
</tr>
</tbody>
</table>

Marco Patrignani
Trace Semantics for PMA

- behaviour in this case is: call in, ret 1 or call in, call out
- traces rely only on the PMA code
- they describe what can be observed from the outside of protected PMA code

```
0x0001  call 0xb52
0x0002  ...

0xb52  movi r0 1
0xb53  movi r1 0xb56
0xb54  jl r1
0xb55  call 0xab01
0xb56  ret

0xab01  ...
```
define states $S$ for programs
Trace Semantics for PMA, Semi-formally

- define states $S$ for programs
- define a semantics for PMA only: $\overset{i}{\rightarrow}: S \times S$
Trace Semantics for PMA, Semi-formally

- define states $S$ for programs
- define a semantics for PMA only: $\rightarrow_i : S \times S$
- define labels (observables) $\alpha$
Trace Semantics for PMA, Semi-formally

- define states $S$ for programs
- define a semantics for PMA only: $\rightarrow: S \times S$
- define labels (observables) $\alpha: \text{call } p \bar{r}$
Trace Semantics for PMA, Semi-formally

- define states $S$ for programs
- define a semantics for PMA only: $i: S \times S$
- define labels (observables) $\alpha : \text{call p r} \mid \text{ret r}_0$
define states \( S \) for programs

- define a semantics for PMA only: \( \overset{i}{\rightarrow}: S \times S \)

- define labels (observables) \( \alpha: \text{call } p \bar{r} \mid \text{ret } r_0 \)

- define a semantics with labels \( \overset{\alpha}{\rightarrow}: S \times \alpha \times S \)
Trace Semantics for PMA, Semi-formally

- define states $S$ for programs
- define a semantics for PMA only: $\rightarrow_i : S \times S$
- define labels (observables) $\alpha : \text{call } p \, \bar{r} | \text{ret } r_0$
- define a semantics with labels $\xrightarrow{\alpha} : S \times \alpha \times S$
- $\text{TR}(C) = \{\bar{\alpha} \mid \exists S'. S(C) \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} S'\}$
define states $S$ for programs
• define a semantics for PMA only: $\xrightarrow{i}: S \times S$
• define labels (observables) $\alpha : \text{call } p \bar{r} \mid \text{ret } r_0$
• define a semantics with labels $\xrightarrow{\alpha} : S \times \alpha \times S$
• $TR(C) = \{ \bar{\alpha} \mid \exists S'. S(C) \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} S' \}$

$TR = \left\{ \alpha = \begin{cases} \frac{i}{;} \\text{call } p \bar{r} \\text{ret } r_0 \\xrightarrow{\alpha} \end{cases} \right\}$
Challenge: Precise Reasoning

- formalism to reason about PMA code simply: ✓
Challenge: Precise Reasoning

- formalism to reason about PMA code simply: ✓
- precise formalism? ✗
Challenge: Precise Reasoning

- Formalism to reason about PMA code simply: ✔
- Precise formalism? ❌
  - PMA code can write in unprotected memory
Challenge: Precise Reasoning

- formalism to reason about PMA code simply: ✔
- precise formalism? ✗
  1. PMA code can write in unprotected memory
  2. flags convey information across function calls
Challenge: Precise Reasoning

- formalism to reason about PMA code simply: ✔
- precise formalism? ✗
  1. PMA code can write in unprotected memory
  2. flags convey information across function calls
  3. registers besides \( r_0 \) in \( \text{ret} \) as well
To ensure maximal precision, prove the trace semantics to be fully abstract
To ensure maximal precision, prove the trace semantics to be fully abstract

i.e. there are no other things that we missed
The Spectrum of Full Abstraction (by Curien)

\[ TR = \left\{ \alpha = \left\{ \begin{array}{l}
\text{call } p \ x \\
\text{ret } r_0 \\
\end{array} \right\} \right\} \]
The Spectrum of Full Abstraction (by Curien)

\[
\begin{align*}
\text{TR} &= \left\{ \alpha = \left\{ \begin{array}{l}
\text{call p r} \\
\text{ret r_0}
\end{array} \right\}; \right\} \\
\text{TR}_L &= \left\{ \alpha = \left\{ \begin{array}{l}
\text{call r f} \\
\text{ret r f} \\
\text{movs r v}
\end{array} \right\}; \right\}
\end{align*}
\]
The Spectrum of Full Abstraction (by Curien)

\[ TR = \{ \alpha = \{ \frac{i_{\gamma}}{}, \text{call } p \bar{r} \}; \} \]

\[ TR_L = \{ \alpha = \{ \frac{i_{\gamma}}{}, \text{call } \bar{r} \bar{f} \}; \} \]

\[ TR_S = \{ \alpha = \{ \frac{i_{\gamma}}{}, \text{ret } r_\theta \}; \} \]
The Spectrum of Full Abstraction (by Curien)

\[ TR = \begin{cases} \alpha = \{ \begin{array}{l} i_x; \\
\text{call } p \quad r_1 \\
\text{ret } r_0 \\
\end{array} \} ; \end{cases} \]

\[ TR_L = \begin{cases} \alpha = \{ \begin{array}{l} i_x; \\
\text{call } \bar{r} \quad f \\
\text{ret } \bar{r} \quad f \\
\text{movs } r \quad v \\
\end{array} \} ; \end{cases} \]

\[ TR_S = \begin{cases} \alpha = \{ \begin{array}{l} i_y; \\
\text{call } p \quad \bar{r} \\
\text{ret } r_0 \\
\end{array} \} ; \end{cases} \]
The Spectrum of Full Abstraction (by Curien)

\[
\begin{align*}
\text{TR} &= \left\{ \alpha = \left\{ \begin{array}{c}
i \\
\text{call } p \bar{r} \\
\text{ret } r_0 \\
\hline
\end{array} \right\} ; \right\} \\
\text{TR}_L &= \left\{ \alpha = \left\{ \begin{array}{c}
i \\
\text{call } \bar{r} \bar{f} \\
\text{ret } \bar{r} \bar{f} \\
\text{movs } r \bar{v} \\
\hline
\end{array} \right\} ; \right\} \\
\text{TR}_S &= \left\{ \alpha = \left\{ \begin{array}{c}
i \\
\text{call } p \bar{r} \\
\text{ret } r_0 \\
\hline
\end{array} \right\} ; \right\}
\end{align*}
\]

\[\text{TR}_X(C_1) = \text{TR}_X(C_2) \iff C_1 \simeq C_2\]
Outline

1. Background
   - PMA and Isolation
   - Secure Compilation: Motivations

2. Secure Compilation of Java Jr
   - Source Language
   - Secure Compilation, Informally
   - Proof Strategy
   - Fully Abstract Trace Semantics for PMA

3. Open Challenges
   - Multilanguage Model
   - Multi-principal Languages
   - Multithreaded Languages
   - Sky is the Limit
Multilanguage Model (by Adriaan)

- reasoning at assembly level is complex
Multilanguage Model (by Adriaan)

- reasoning at assembly level is complex
- a $\lambda$-calculus model simplifies proofs for other language features
Multilanguage Model (by Adriaan)

- reasoning at assembly level is complex
- a λ-calculus model simplifies proofs for other language features

Q: How to proceed?
reasoning at assembly level is complex

a $\lambda$-calculus model simplifies proofs for other language features

Q: How to proceed?

devise a multilanguage model
reasoning at assembly level is complex

a $\lambda$-calculus model simplifies proofs for other language features

Q: How to proceed?

devise a multilanguage model

show that it models precisely PMA
reasoning at assembly level is complex
a $\lambda$-calculus model simplifies proofs for other language features

Q: How to proceed?

- devise a multilanguage model
- show that it models precisely PMA
- adopt it in other proofs!
Multi-principal Languages

- current model has a single secure entity
Multi-principal Languages

- current model has a single secure entity
- current prototypes allow a flat security order
Multi-principal Languages

- current model has a single secure entity
- current prototypes allow a flat security order

Q: How can we improve on this?
Multi-principal Languages

- current model has a single secure entity
- current prototypes allow a flat security order

Q: How can we improve on this?

- model a security lattice with the current prototypes
Multi-principal Languages

- current model has a single secure entity
- current prototypes allow a flat security order

**Q:** How can we improve on this?

- model a security lattice with the current prototypes
- secure compilation of languages with multiple security principals
Multithreaded Languages

- current PMA prototypes are single-threaded
current PMA prototypes are single-threaded
secure compilation for PMA does not consider concurrency nor distribution
Multithreaded Languages

- current PMA prototypes are single-threaded
- secure compilation for PMA does not consider concurrency nor distribution

Q: What’s next on this line?
Multithreaded Languages

- current PMA prototypes are single-threaded
- secure compilation for PMA does not consider concurrency nor distribution

Q: What’s next on this line?

- existing works cover concurrency and distribution for secure compilers
Multithreaded Languages

- current PMA prototypes are single-threaded
- secure compilation for PMA does not consider concurrency nor distribution

**Q:** What’s next on this line?

- existing works cover concurrency and distribution for secure compilers
- investigate the implementation: which interrupts to handle?
Multithreaded Languages

- current PMA prototypes are single-threaded
- secure compilation for PMA does not consider concurrency nor distribution

**Q:** What’s next on this line?

- existing works cover concurrency and distribution for secure compilers
- investigate the implementation: which interrupts to handle?
- the single-machine, multithreaded setting is poorly explored
What is the limit?

Q: Are there language features that cannot be securely compiled through PMA? How to formalise this statement? I think the answer is NO.
Q: Are there language features that cannot be securely compiled through PMA?
What is the limit?

**Q:** Are there language features that cannot be securely compiled through PMA?

- how to formalise this statement?
Q: Are there language features that cannot be securely compiled through PMA?

- how to formalise this statement?
- I think the answer is NO
Questions

Thank you!

Qs ?