Secure Compilation to Protected Module Architectures

Marco Patrignani ¹  Dave Clarke ¹,²  Frank Piessens ¹

¹iMinds-DistriNet, Dept. Computer Science, KU Leuven, Belgium  
first.last@cs.kuleuven.be

²Dept. Information Technology, Uppsala University, Sweden  
first.last@it.uu.se

27 April 2015
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 (What are Secure Compilation and PMA?)
   - Secure Compilation
   - PMA and Isolation
   - Fully Abstract Trace Semantics for PMA

2. Secure Compilation of J+E
   - Source Language J+E
   - Secure Compilation, Informally
   - Proof Strategy

3. Open Challenges
Outline

1. Background (What are Secure Compilation and PMA?)
   - Secure Compilation
   - PMA and Isolation
   - Fully Abstract Trace Semantics for PMA

2. Secure Compilation of J+E
   - Source Language J+E
   - Secure Compilation, Informally
   - Proof Strategy

3. Open Challenges
What is a Secure Program?

- a program is secure if it enjoys at least a security property
What is a Secure Program?

- a program is secure if it enjoys at least a security property
- a security property is one expressible via program equivalence (e.g., confidentiality, integrity, etc.)
What is a Secure Compiler?

- a compiler is a function from source to target programs.
What is a Secure Compiler?

- a compiler is a function from source to target programs
- a compiler is secure if it preserves source-level security properties in the programs it generates *no more, no less*
What is a Secure Compiler?

- a compiler is a function from source to target programs
- a compiler is secure if it preserves source-level security properties in the programs it generates *no more, no less*
- a fully abstract compiler is a secure compiler
Benefits of Fully abstract Compilation

- Fully abstract compilation preserves source-level abstractions in target-level languages.
- Protects against code injection attacks.
- Enables source-level reasoning.
Benefits of Fully abstract Compilation

Fully abstract compilation preserves source-level abstractions in target-level languages
Benefits of Fully abstract Compilation

Fully abstract compilation preserves source-level abstractions in target-level languages

- protect against code injection attacks
Benefits of Fully abstract Compilation

Fully abstract compilation preserves source-level abstractions in target-level languages

- protect against code injection attacks
- enables source-level reasoning
What is a Protected Modules Architecture?

- deep encapsulation at the lowest level of abstraction
What is a Protected Modules Architecture?

- deep encapsulation at the lowest level of abstraction
- the basis of several security-related works
What is a Protected Modules Architecture?

- deep encapsulation at the lowest level of abstraction
- the basis of several security-related works
- Intel wants to port it to future processors (SGX)
A+I: Untyped Assembly + PMA

```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
     ...
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0x0001
0x0b55    ...
     ...
0xab00    jmp 0xb53
0xab01    ...
```
A+I: Untyped Assembly + PMA

- memory space
- protected module = protected memory

```
0x0001   call 0xb53
0x0002   movs r0 0xb55

0xb52    movs r0 0xb55
0xb53    call 0x0002
0xb54    movs r0 0x0001
0xb55    ...

0xab00   jmp 0xb53
0xab01   ...
```
A+I: Untyped Assembly + PMA

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

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

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

0xab00  jmp 0xb53
0xab01  ...
```
A+I: Untyped Assembly + PMA

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

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

Marco Patrignani, Dave Clarke, Frank Piessens
**A+I: Untyped Assembly + PMA**

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

---

<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 0x0b55</td>
</tr>
<tr>
<td>0x0b52</td>
<td>movs r0 0x0b55</td>
</tr>
<tr>
<td>0x0b53</td>
<td>call 0x0002</td>
</tr>
<tr>
<td>0x0b54</td>
<td>movs r0 0x0001</td>
</tr>
<tr>
<td>0x0b55</td>
<td>...</td>
</tr>
<tr>
<td>0xab00</td>
<td>jmp 0xb53</td>
</tr>
<tr>
<td>0xab01</td>
<td>...</td>
</tr>
</tbody>
</table>
A+I: Untyped Assembly + PMA

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

```
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
A+I: Untyped Assembly + PMA

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

### Secure Compilation of J+E

- Open Challenges

### PMA and Isolation

- Fully Abstract Trace Semantics for PMA

### A+I: Untyped Assembly + PMA

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

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

0xab00 jmp 0xb53
0xab01 ...
```
A+I: Untyped Assembly + PMA

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

```
0x0001  call 0xb53
0x0002  movs r0 0x0b55
......
0x0b52  movs r0 0x0b55
0x0b53  call 0x0002
0x0b54  movs r0 0x0001
0x0b55  ...
......
0xab00  jmp 0xb53
0xab01  ...
```
A+I: Untyped Assembly + PMA

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
A+I: Untyped Assembly + PMA

- 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 r0 0x0b55

0xb52  movs r0 0x0b55
0xb53  call 0x0002
0xb54  movs r0 0x0001
0xb55  ...

0xab00  jmp 0xb53
0xab01  ...
```
A+I: Untyped Assembly + PMA

- 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 r0 0x0b55
...  
0x0b52  movs r0 0x0b55
0x0b53  call 0x0002
0x0b54  movs r0 0x0001
0x0b55  ...
...  
0xab00  jmp 0xb53
0xab01  ...
```
Background (What are Secure Compilation and PMA?)

Secure Compilation

PMA and Isolation

Fully Abstract Trace Semantics for PMA

Marco Patrignani, Dave Clarke, Frank Piessens

Trace Semantics for PMA

0x0001 call 0xb52
0x0002 ...
0x0b52 movi r
0x0b53 movi r
0x0b56 jl r
0x0b55 call 0xab01
0x0b56 ret

...
Trace Semantics for PMA

behaviour in this case is:

```plaintext
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  ...
...
0x0b52  movi r0 1
0x0b53  movi r1 0x0b56
0x0b54  jl r1
0x0b55  call 0xab01
0x0b56  ret
...
0xab01  ...
```
Trace Semantics for PMA

- behaviour in this case is: *call in, ret 1*

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

```
0xb52   movi r0 1
0xb53   movi r1 0x0b56
0xb54   jl r1
0xb55   call 0xab01
0xb56   ret
```

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

- behaviour in this case is: call in, ret 1 or call in,

```
0x0001  call 0xb52
0x0002  ...
...
0x0b52  movi r0 1
0x0b53  movi r1 0x0b56
0x0b54  jl r1
0x0b55  call 0xab01
0x0b56  ret
...
0xab01  ...
```
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

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, 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

```plaintext
0x0001 call 0xb52
0x0002 ...
...
0xb52 movi r0 1
0xb53 movi r1 0xb56
0xb54 jl r1
0xb55 call 0xab01
0xb56 ret
...
0xab01 ...
```
Outline

1. Background (What are Secure Compilation and PMA?)
   - Secure Compilation
   - PMA and Isolation
   - Fully Abstract Trace Semantics for PMA

2. Secure Compilation of J+E
   - Source Language J+E
   - Secure Compilation, Informally
   - Proof Strategy

3. Open Challenges
J+E: Java-like Language

- source language: +/- Java jr
J+E: Java-like Language

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

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

```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;
```
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;
```
J+E: Java-like Language

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

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

    private counter : Int;
}
object extAccount : AccountClass;
J+E: Java-like Language

Dynamic dispatch
v-tables
Secure stack

```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;
```
J+E: Java-like Language

1. package PI;
2.   interface Account {
3.     public createAccount() : Foo;
4.   }
5.   extern extAccount : Account;

6. package PE;
7.   class AccountClass
8.     implements PI.Account {
9.     AccountClass() { counter = 0; }
10.    public createAccount() : Account {
11.        return new PE.AccountClass();
12.    }
13. }
14.  
15.    private counter : Int;
16. }
17. object extAccount : AccountClass;

- proxy to createAccount
- Dynamic dispatch
- v-tables
- Secure stack
J+E: Java-like Language

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;
PMA for Secure Compilation

<table>
<thead>
<tr>
<th>Source level</th>
<th>Ext 1</th>
<th>Ext 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Protect against low-level attackers

Target code is vulnerable without PMA
PMA for Secure Compilation

Source level

O1 ← 01.createElement()
O2

Target level

Ext 1
Ext 2
PMA for Secure Compilation

Source level

- O1
- O2
- O3

Ext 1
Ext 2

Protect against low-level attackers

Target code is vulnerable without PMA
PMA for Secure Compilation

\[ \text{Source level} \]

\begin{align*}
O1 & \rightarrow \text{return } O3 \\
O2 & \rightarrow \text{Ext 1} \\
O3 & \rightarrow \text{Ext 2}
\end{align*}

\[ \text{Target level} \]
PMA for Secure Compilation

Source level

O1
O2
O3

03.counter

Ext 1
Ext 2
PMA for Secure Compilation

Source level

O1
O2
O3

03.counter

Ext 1
Ext 2
PMA for Secure Compilation

Source level

O1
O2
O3

Ext 1
Ext 2

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

Source level

O1
O2
O3

Target level

O1↓
O2↓

Ext 1
Ext 2

Ext 1↓
Ext 2↓
PMA for Secure Compilation

Source level

O1
O2
O3

Target level

O1 ↓.createAccount()  
O2 ↓  
O3 ↓  

Ext 1↓
Ext 2↓
PMA for Secure Compilation

Source level

O1
O2
O3

Target level

O1\downarrow
O2\downarrow
O3\downarrow

\rightarrow

Ext 1\downarrow
Ext 2\downarrow

Protect against low-level attackers
Target code is vulnerable without PMA
Secure Compilation to PMA
PMA for Secure Compilation

Source level

- O1
- O2
- O3

Target level

- Ext 1
- Ext 2

- Protect against low-level attackers

Node O3 is counterbalanced by the Ext 1 and Ext 2 transformations.
PMA for Secure Compilation

- Protect against low-level attackers
- Target code is vulnerable without PMA

Source level:
- O1
- O2
- O3

Target level:
- O1\downarrow
- O2\downarrow
- O3\downarrow
- Ext 1\downarrow
- Ext 2\downarrow
- Ext 1\downarrow
- Ext 2\downarrow

\[O3\downarrow \text{counter} \]
Secure Compilation of Outcalls

Q: Is that all?
Secure Compilation of Outcalls

Q: Is that all?

0x0001 Unprotected stack
0x0002
\[...
0xb52
0xb53
0xb54
0xb55
Secure Compilation of Outcalls

Q: Is that all?

- protected stack

Leaks stack contents
0x0001
0x0002
0xb52
0xb53
0xb54
0xb55
Secure Compilation of Outcalls

Q: Is that all?

- protected stack

Leaked stack contents:
- 0x0001
- 0x0002
- ...
- 0xb52
- 0xb53 Protected stack
- 0xb54
- 0xb55
Secure Compilation of Outcalls

Q: Is that all?

- protected stack
- returnback entry point

0x0001 Unprotected stack
0x0002

..

0x0b52
0x0b53 Protected stack
0x0b54
0x0b55
Secure Compilation of Outcalls

Q: : Is that all?

- protected stack
- returnback entry point

[Diagram showing memory addresses: 0x0001 Unprotected stack, 0x0002, ..., 0xb52, 0xb53 Protected stack, 0xb54, 0xb55]
Secure Compilation of Outcalls

Q: Is that all?

- protected stack
- returnback entry point
Secure Compilation of Outcalls

Q: Is that all?

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

0x0001 Unprotected stack
0x0002
...
0xb52
0xb53 Protected stack
0xb54
0xb55
Secure Compilation of Outcalls

Q: Is that all?

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

0x0001 Unprotected stack
0x0002 :
...
0xb52
0xb53 Protected stack
0xb54 je ...
0xb55
Secure Compilation of Outcalls

Q: Is that all?

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

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

\( f_{ZS} = 0/1 \)
Secure Compilation of Outcalls

Q: Is that all?

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

\[ f_{\text{ZS}} = 0 \]

- 0x0001 Unprotected stack
- 0x0002
- 0x0b52
- 0x0b53 Protected stack
- 0x0b54 \textbf{je} \ldots
- 0x0b55
Secure Compilation of Outcalls

Q: Is that all?

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

0x0001 Unprotected stack
0x0002
...0x0b52
0x0b53 Protected stack
0x0b54
0x0b55
Secure Compilation of Outcalls

**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
```

\[ r_0 : \text{Bool} \]
Secure Compilation of Outcalls

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

Q: : Is that all?

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

check r0 = 1/0
Dynamic Memory Allocation

Source level

O1
O2

Target level

O1\downarrow
O2\downarrow

Ext 1
Ext 2

Ext 1\downarrow
Ext 2\downarrow
Dynamic Memory Allocation

Source level

O1
O2

Ext 1
Ext 2

Target level

O1
O2

Ext 1
Ext 2

01.createAccount()
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 -> return O3 -> Ext 1
O2 -> Ext 2
O4
O3

Target level

O1↓
O2↓
Ext 1↓
Ext 2↓
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Ext 1
Ext 2

Target level

O1↓.createAccount()

01↓.createAccount()

Ext 1↓
Ext 2↓
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Ext 1
Ext 2

Target level

O1↓
O2↓
O4↓
O3↓

Ext 1↓
Ext 2↓
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Target level

O1
O2
O4
O3

return O3

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

0x001
0x005
0x009
0x00C

return 0x00C

Ext 1
Ext 2

Map object id to natural numbers:
- Add object id to map.
- Lookup when number is received.

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

Source level

O1
O2
O4
O3

Ext 1
Ext 2

Target level

0x001
0x005
0x009
0x00C

0x009.createAccount()

Ext 1
Ext 2
Dynamic Memory Allocation

Source level
O1
O2
O4
O3

Target level
0x001
0x005
0x009
0x00C

Ext 1
Ext 2

04.createAccount()

0x009.createAccount()
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

04.createAccount()

Ext 1
Ext 2

Target level

0x001
0x005
0x009
0x00C

0x009.createObject()

0x009.createObject()

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

Ext 1
Ext 2

Object id guessing
map Oid to natural numbers

Target level

0x001
0x005

Ext 1↓
Ext 2↓
Dynamic Memory Allocation

Source level

- O1
- O2
- O4
- O3

Ext 1
Ext 2

Target level

- 0x001 $\mapsto$ 1
- 0x005 $\mapsto$ 2

- Object id guessing
- map Oid to natural numbers

Marco Patrignani, Dave Clarke, Frank Piessens
Dynamic Memory Allocation

Source level

- O1
- O2
- O4
- O3

Ext 1
Ext 2

Target level

- 0x001 \(\mapsto\) 1
- 0x005 \(\mapsto\) 2

1. createAccount()

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

Source level

- O1
- O2
- O4
- O3

Target level

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

Extended

- Ext 1
- Ext 2

Object id guessing
- map Oid to natural numbers

Proof Strategy

Secure Compilation of J+E

Open Challenges

Secure Compilation, Informally

Marco Patrignani, Dave Clarke, Frank Piessens
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Ext 1
Ext 2

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
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Ext 1
Ext 2

Target level

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

return 3

Ext 1
Ext 2

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

Marco Patrignani, Dave Clarke, Frank Piessens
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Ext 1
Ext 2

Target level

0x001 ↦ 1
0x005 ↦ 2
0x009
0x00C ↦ 3

- 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

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
\texttt{O2.createAccount()}

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

\textbf{Proof Strategy}

\textbf{Object id guessing}

\textbf{Object id guessing}

\textbf{Object id guessing}

\textbf{Object id guessing}

\textbf{Object id guessing}
Dynamic Memory Allocation

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

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

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

Marco Patrignani, Dave Clarke, Frank Piessens
Dynamic Memory Allocation

Source level

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

Target level

- 0x001 \rightarrow 1
- 0x005 \rightarrow 2
- 0x009
- 0x00C \rightarrow 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
- 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
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
2.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
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 1 \downarrow

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, Dave Clarke, Frank Piessens
Exceptions

Stack

Secure stack

Insecure stack

to
Exceptions

Stack

- \( f_s \)
- \( g \)
- \( h_s \)
- \( i \)
- \( l_s \)

1

throw e

Secure stack

- \( f_s \)
- \( h_s \)
- \( l_s \)

\textbf{throw e}

Insecure stack

- \( g \)
- \( i \)
Exceptions

Stack

\[ f_s \]
\[ g \]
\[ h_s \]
\[ i \]
\[ l_s \]

\[ \text{throw } e \]

Secure stack

\[ f_s \]
\[ h_s \]
\[ l_s \]

\[ \text{throw } e \]

Insecure stack

\[ g \]
\[ i \]
 Exceptions

Stack

```
fs
g
hs
i
ls
```

Secure stack

```
fs
hs
ls
```

Insecure stack

```
g
i
```

Throw e

3
2
1

Marco Patrignani, Dave Clarke, Frank Piessens
Exceptions

Stack

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

Insecure stack

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

Secure stack

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

throw e

Marco Patrignani, Dave Clarke, Frank Piessens
Exceptions

Stack

5  
4  
3  
2  
1  
throw e

Secure stack

Secure stack

Insecure stack

throw e

Marco Patrignani, Dave Clarke, Frank Piessens
Exceptions

Stack

Secure stack

Insecure stack

Throw e

 Marco Patrignani, Dave Clarke, Frank Piessens
Exceptions

Stack

5
\[f_s\]
4
\[g\]
3
\[h_s\]
2
\[i\]
1
\[l_s\]

\textbf{throw e}

Secure stack

\[
\begin{aligned}
&f_s \\
&h_s \\
&l_s \\
&\text{throw e}
\end{aligned}
\]

Insecure stack

\[
\begin{aligned}
&g \\
&i
\end{aligned}
\]
Exceptions

Stack

Insecure stack

Secure stack

Record passed exceptions

Stack:

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

Throw exception $e$.

Secure stack:

1. $l_s$
2. $h_s$
3. $f_s$

Insecure stack:

1. $i$
2. $g$
Exceptions

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

Stack

Secure Compilation of J+E

Open Challenges

Source Language J+E

Secure Compilation, Informally

Proof Strategy

Record passed exceptions
Check that exception could be thrown
Additional Features

- cross-package inheritance
- inner classes
- ML-like modules & functors (joint work with A. Larmuseau from UU)
So now...

- We have a strategy to securely compile J+E code
So now...

- We have a strategy to securely compile J+E code
- We have the tools to implement it
So now...

- We have a strategy to securely compile J+E 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 J+E 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?
So now...

- We have a strategy to securely compile J+E 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?

A PROOF!
Secure Compilation, Formally

\[ C_1 \simeq^{J+E} C_2 \iff C_1 \downarrow \simeq^{A+I} C_2 \downarrow \]
Secure Compilation, Formally

\[ C_1 \sim_{J+E} C_2 \iff C_1 \sim_{A+I} C_2 \]
Secure Compilation, Formally

$C_1 \sim_{J+E} C_2 \iff C_1 \downarrow \sim_{A+I} C_2$

Contextual Equivalence
Contextual Equivalence

\[ C_1 \sim^S C_2 \triangleq \forall C. \; C[C_1] \uparrow \iff C[C_2] \uparrow \]
Contextual Equivalence

\[ C_1 \sim^S C_2 \triangleq \forall C. \ C[C_1] \uparrow \iff C[C_2] \uparrow \]
Contextual Equivalence

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

All contexts
Secure Compilation

\[
C_1 \simeq_{\text{J+E}} C_2 \iff C_1 \downarrow \simeq_{\text{A+I}} C_2 \downarrow
\]

Very complex!

Marco Patrignani, Dave Clarke, Frank Piessens
Secure Compilation

\[ C_1 \sim^{J+E} C_2 \iff C_1 \downarrow \sim^{A+I} C_2 \downarrow \]

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

\[ C_1 \sim^{J+E} C_2 \iff C_1 \downarrow \sim^{A+I} C_2 \downarrow \]

(\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!
Secure Compilation

\[ C_1 \sim^{J+E} C_2 \iff C_1 \downarrow \sim^{A+I} C_2 \downarrow \]
Secure Compilation

\[ C_1 \sim^{J+E} C_2 \iff C_1 \downarrow \sim^{A+I} C_2 \downarrow \]
Secure Compilation

\[ C_1 \sim_{J+E} C_2 \checkmark \quad C_1 \downarrow \sim_{A+I} C_2 \downarrow \]
Secure Compilation

\[ C_1 \sim^{J+E} C_2 \Rightarrow C_1^\downarrow \sim^{A+I} C_2^\downarrow \]
Secure Compilation

\[ C_1 \sim_{J^E} C_2 \Rightarrow C_1^{\downarrow} \sim_{A^I} C_2^{\downarrow} \]

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

Fully Abstract Trace Semantics
Secure Compilation

\[ C_1 \not\simeq_{\text{J+E}} C_2 \iff \text{Traces}(C_1) \neq \text{Traces}(C_2) \]
Secure Compilation

\[ C_1 \not\cong_{J^E} C_2 \quad \text{✓} \quad \text{Traces}(C_1) \neq \text{Traces}(C_2) \]
Secure Compilation

\[ C_1 \not\sim^{J+E} C_2 \]

\[
C_1 \downarrow \sim^{A+I} C_2 \downarrow
\]

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

Fully Abstract Trace Semantics
Outline

1 Background (What are Secure Compilation and PMA?)
   - Secure Compilation
   - PMA and Isolation
   - Fully Abstract Trace Semantics for PMA

2 Secure Compilation of J+E
   - Source Language J+E
   - Secure Compilation, Informally
   - Proof Strategy

3 Open Challenges
Multi-principal Languages

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

- current model has a single secure entity

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

- current model has a single secure entity

  Q: How can we improve on this?

- secure compilation of languages with multiple security principals
Better Proof Technique

- current proof technique is unpleasant
Better Proof Technique

- current proof technique is unpleasant

**Q:** How can we improve on this?
Better Proof Technique

- current proof technique is unpleasant

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

- develop a more scalable proof technique (most likely based on logical relations)
Garbage Collection

- current secure compilation scheme does not account for garbage collection
Garbage Collection

- current secure compilation scheme does not account for garbage collection

**Q:** How can we improve on this?
current secure compilation scheme does not account for garbage collection

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

- find a suitable technique (distributed GC-based)
Garbage Collection

- current secure compilation scheme does not account for garbage collection

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

- find a suitable technique (distributed GC-based)
- prove that it does not introduce security leaks
PMA Expressiveness

Q: Are there language features that cannot be securely compiled through PMA?
I think the answer is NO.
Q: Are there language features that cannot be securely compiled through PMA?
PMA Expressiveness

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 ?
define states $S$ for programs
Trace Semantics for PMA, Semi-formally

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

- define states $S$ for programs
- define a semantics for PMA only: $\rightarrow: S \times S$
- define labels (observables) $\alpha$
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: $\rightarrow: S \times S$
- define labels (observables) $\alpha: \text{call } p \bar{r} | \text{ret } r_0$
Trace Semantics for PMA, Semi-formally

- define states $S$ for programs
- define a semantics for PMA only: $\rightarrow: S \times S$
- define labels (observables) $\alpha : \text{call } p \bar{r} | \text{ret } r_0$
- define a semantics with labels $\xrightarrow[\alpha]{} : S \times \alpha \times S$
Trace Semantics for PMA, Semi-formally

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

$$\text{TR} = \left\{ \alpha = \left\{ \begin{array}{c} i; \\ \text{call p } \bar{r} \\ \text{ret } r_0 \\ \alpha \end{array} \right\} \right\}$$
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 `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)

\[ TR = \\begin{cases} \alpha = \{ \text{call } p \overline{p}, \text{ret } r_0 \}; \end{cases} \]
The Spectrum of Full Abstraction (by Curien)

\[ TR = \begin{cases} 
\alpha = \{ \text{call } p \bar{r} \} ; \\
\text{ret } r_0 \\
\alpha \Rightarrow \end{cases} \]

\[ TR_L = \begin{cases} 
\alpha = \{ \text{call } \bar{r} \bar{f} \} ; \\
\text{ret } \bar{r} \bar{f} \\
\text{movs } r v \\
\alpha \Rightarrow \end{cases} \]
The Spectrum of Full Abstraction (by Curien)

\[ \text{TR} = \left\{ \alpha = \left\{ i, \right\} ; \right\} \]

\[ \text{TR}_L = \left\{ \alpha = \left\{ \text{call } r\bar{r} \right\} ; \right\} \]

\[ \text{TR}_S = \left\{ \alpha = \left\{ \text{call } p\bar{r} \right\} ; \right\} \]
The Spectrum of Full Abstraction (by Curien)

\[ \text{TR} = \begin{cases} \alpha = \{ \frac{i}{\alpha}, \text{call p \overline{r}} \} ; \alpha \Rightarrow \{ \text{ret } r_0 \} ; \end{cases} \]

\[ \text{TR}_L = \begin{cases} \alpha = \{ \frac{i}{\alpha}, \text{call } r \overline{f} \} ; \text{ret } r \overline{f} \} ; \text{movs } r v \} ; \alpha \Rightarrow \end{cases} \]

\[ \text{TR}_S = \begin{cases} \alpha = \{ \frac{i}{\alpha}, \text{call p } \overline{r} \} ; \text{ret } r_0 \} ; \end{cases} \]
The Spectrum of Full Abstraction (by Curien)

\[ TR = \left\{ \alpha = \left\{ \begin{array}{l} i; \\text{call } p \bar{r} \\ \text{ret } r_0 \end{array} \right\}; \right\} \]

\[ TR_L = \left\{ \alpha = \left\{ \begin{array}{l} i; \\text{call } \bar{r} \bar{f} \\ \text{ret } \bar{r} \bar{f} \\ \text{movs } r v \end{array} \right\}; \right\} \]

\[ TR_S = \left\{ \alpha = \left\{ \begin{array}{l} i; \\text{call } p \bar{r} \\ \text{ret } r_0 \end{array} \right\}; \right\} \]

\[ TR_X(C_1) = TR_X(C_2) \iff C_1 \simeq C_2 \]