Secure Compilation to Protected Module Architectures

Marco Patrignani $^1$  
Dave Clarke $^{2,3}$  
Frank Piessens $^2$

$^1$Max Planck institute for software systems, Saarbrücken, Germany  
first.last@mpi-sws.org
$^2$iMinds-DistriNet, Dept. Computer Science, KU Leuven, Belgium  
first.last@cs.kuleuven.be
$^3$Dept. Information Technology, Uppsala University, Sweden  
first.last@it.uu.se

25 January 2016
Goal of the Talk

- present my research on secure compilation
Goal of the Talk

- present my research on secure compilation
- define secure (fully-abstract) compilation
Goal of the Talk

- present my research on secure compilation
- define secure (fully-abstract) compilation
- discuss present and future work
Background (What are Secure Compilation and PMA?)
Secure Compilation of J+E
Recent Work

Marco Patrignani, Dave Clarke, Frank Piessens
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. Recent Work
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. Recent Work
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
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 0xb55
...
0x0b52  movs r0 0xb55
0x0b53  call 0x0002
0x0b54  movs r0 0xb55
0x0b55  ...
...
0xab00  jmp 0xb53
0xab01  ...
```

- memory space

**Background (What are Secure Compilation and PMA?)**
- Secure Compilation of J+E
- Recent Work

**Secure Compilation**
- PMA and Isolation
- Fully Abstract Trace Semantics for PMA
A+I: Untyped Assembly + PMA

- 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  ...
```
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  ...

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

```
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

\[
\begin{array}{c}
0x0001 & \text{call } 0xb53 \\
0x0002 & \text{movs } r_0 \ 0x0b55 \\
\vdots \\
0x0b52 & \text{movs } r_0 \ 0x0b55 \\
0x0b53 & \text{call } 0x0002 \\
0x0b54 & \text{movs } r_0 \ 0x0001 \\
0x0b55 & \ldots \\
\vdots \\
0xab00 & \text{jmp } 0xb53 \\
0xab01 & \ldots \\
\end{array}
\]
### Background (What are Secure Compilation and PMA?)

Secure Compilation of J+E

Recent Work

#### Secure Compilation

**PMA and Isolation**

**Fully Abstract Trace Semantics for PMA**

---

### A+I: Untyped Assembly + PMA

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

### Assembly Code Example

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>0x0001 call 0xb53</th>
<th>0x0002 movs r0 0x0b55</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0b52</td>
<td>movs</td>
<td>r0 0x0b55</td>
<td></td>
</tr>
<tr>
<td>0x0b53</td>
<td>call</td>
<td>0x0002</td>
<td></td>
</tr>
<tr>
<td>0x0b54</td>
<td>movs</td>
<td>r0 0x0001</td>
<td></td>
</tr>
<tr>
<td>0x0b55</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xab00</td>
<td>jmp</td>
<td>0xb53</td>
<td></td>
</tr>
<tr>
<td>0xab01</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

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

<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>

- 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

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
- entry points for communication (■)

```
0x0001 call 0xb53
0x0002 movs r0 0x0b55
...
0xb52 movs r0 0xb55
0xb53 call 0x0002
0xb54 movs r0 0x0001
0xb55 ...
...
0xab00 jmp 0xb53
0xab01 ...
```
Trace Semantics for PMA
Trace Semantics for PMA

- behaviour in this case is:

```assembly
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

```
0x0001   call 0xb52
0x0002   ...
...
0x0b52   movi r0 1
0x0b53   movi r1 0xb56
0x0b54   jl r1
0x0b55   call 0xab01
0x0b56   ret
...
0xab01   ...
```

- behaviour in this case is: `call in`
Trace Semantics for PMA

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

```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, 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

```
0x0001  call 0xb52
0x0002  ...
...
0x0b52  movi r0 1
0x0b53  movi r1 0xb56
0x0b54  jl r1
0x0b55  call 0xab01
0x0b56  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. Recent Work
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

```
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

- 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

```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

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

- **proxy to createAccount**
- **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

- 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

Source level

O1
O2

Ext 1
Ext 2
Secure Compilation of J+E

Recent Work

Source Language J+E

Secure Compilation, Informally

Proof Strategy

PMA for Secure Compilation

Source level

O1 ← \texttt{O1.createAccount()}

O2

Target level

Ext 1

Ext 2

Protect against low-level attackers

Target code is vulnerable without PMA
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>
<tr>
<td>O3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Secure Compilation of J+E

Recent Work

Source Language J+E

Secure Compilation, Informally

Proof Strategy

Marco Patrignani, Dave Clarke, Frank Piessens

Secure Compilation to PMA
Background (What are Secure Compilation and PMA?)
Secure Compilation of J+E
Recent Work

Secure Compilation, Informally
Proof Strategy

PMA for Secure Compilation

Source level

O1
O2
O3

03.counter

Ext 1
Ext 2

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

Marco Patrignani, Dave Clarke, Frank Piessens
Secure Compilation to PMA
PMA for Secure Compilation

Source level
O1
O2
O3

Target level
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

Ext 1
Ext 2
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

Ext 1
Ext 2

Target level

O1\downarrow .createAccount()
O2\downarrow
O3\downarrow

Ext 1\downarrow
Ext 2\downarrow
**PMA for Secure Compilation**

Source level

O1
O2
O3

Target level

O1
O2
O3

<table>
<thead>
<tr>
<th>Ext 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ext 2</td>
</tr>
</tbody>
</table>

Protect against low-level attackers

Target code is vulnerable without PMA

Marco Patrignani, Dave Clarke, Frank Piessens
Background (What are Secure Compilation and PMA?)
Secure Compilation of J+E
Recent Work

Secure Compilation, Informally
Proof Strategy

PMA for Secure Compilation

Source level
O1
O2
O3

Target level
O1↓
O2↓
O3↓

Ext 1
Ext 2

O1
↓
Ext 1↓

O2
↓
Ext 2↓

03↓ .counter

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

Marco Patrignani, Dave Clarke, Frank Piessens
PMA for Secure Compilation

Protect against low-level attackers

Source level

O1
O2
O3

Target level

O1↓
O2↓
O3↓

Ext 1
Ext 2

O1
Ext 1↓

O2
Ext 2↓

O3↓ Counter
PMA for Secure Compilation

- Source level
  - O1
  - O2
  - O3

- Target level
  - O1\downarrow
  - O2\downarrow
  - O3\downarrow

- Ext 1\downarrow
- Ext 2\downarrow

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

Source: Marco Patrignani, Dave Clarke, Frank Piessens

Secure Compilation to PMA
Secure Compilation of Outcalls

Q: : Is that all?
Secure Compilation of Outcalls

Q: Is that all?

0x0001 Unprotected stack
0x0002

0x0b52
0x0b53
0x0b54
0x0b55
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?

protection stack

Leaks stack contents

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

Q: Is that all?

- protected stack
- returnback entry point

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

Marco Patrignani, Dave Clarke, Frank Piessens
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

![Diagram showing call and return at protected and unprotected stack locations]
Q: Is that all?

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

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

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

```
0x0001 Unprotected stack
0x0002
:

0x0b52
0x0b53 Protected stack
0x0b54 je ...
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
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

r_0 : Bool
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
```

check $r_0 = 1/0$
Dynamic Memory Allocation

Source level

O1
O2

Ext 1
Ext 2

Target level

O1\downarrow
O2\downarrow

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

Ext 1
Ext 2

Target level

O1↓
O2↓

Ext 1↓
Ext 2↓
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Return O3

Ext 1
Ext 2

Target level

O1↓
O2↓

Ext 1↓
Ext 2↓
Dynamic Memory Allocation

Source level:
- O1
- O2
- O4
- O3

Target level:
- O1
- O2

Ext 1
- Ext 1

Ext 2
- Ext 2

O1→.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

Ext 1
Ext 2

Target level

O1↓
O2↓
O4↓
O3↓

return O3↓

return O3↓

Ext 1↓
Ext 2↓
Dynamic Memory Allocation

Source level:
- O1
- O2
- O4
- O3

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

Return 0x00C

Ext 1
- Ext 1
- Ext 2

Ext 2
- Ext 2

Marco Patrignani, Dave Clarke, Frank Piessens
**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

0x009.createAccount()

Ext 1
Ext 2

04.createAccount()

O4.createAccount()
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Target level

0x001
0x005
0x009
0x00C

Ext 1
Ext 2

- - - - -

Object id guessing

04.createAccount()

0x009.createAccount()
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Ext 1
Ext 2

Target level

0x001
0x005

Ext 1
Ext 2

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

Source level

O1
O2
O4
O3

Ext 1
Ext 2

Target level

0x001 ↦ 1
0x005 ↦ 2

Ext 1↓
Ext 2↓

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

Source level

O1
O2
O4
O3

Ext 1
Ext 2

Object id guessing
map Oid to natural numbers

Target level

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

1.createAccount()
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\downarrow

Ext 2\downarrow

- 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 ↦ 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 ↦ 1
0x005 ↦ 2
0x009
0x00C ↦ 3

return 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 ⇔ 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
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 → 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

02.createAccount() Ext 1
Ext 2

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
Secure Compilation of J+E

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

2. createAccount()
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

Ext 1↓
Ext 2↓

- 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.\texttt{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**
- 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
- no need of extra information
Exceptions

Stack

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

\text{throw e}

Secure stack

\[ f_s \]
\[ h_s \]
\[ i_s \]

\text{throw e}

Insecure stack

\[ g \]
\[ i \]
Exceptions

Stack

\[ f_s \]

\[ g \]

\[ h_s \]

\[ i \]

\[ l_s \]

1

throw e

Secure stack

\[ f_s \]

\[ h_s \]

\[ l_s \]

throw e

Insecure stack

\[ g \]

\[ i \]
Exceptions

Stack

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

1. Throw e

Secure stack

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

Insecure stack

- $g$
- $i$

- Throw e
Exceptions

Stack

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

Secure stack

\[ f_s, h_s, l_s \]

Insecure stack

\[ g, i \]

throw e

throw e
Exceptions

Stack

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

Secure stack

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

Insecure stack

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

Stack

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

throw e

Secure stack

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

throw e

Insecure stack

- $g$
- $i$

Marco Patrignani, Dave Clarke, Frank Piessens
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

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

Secure stack

Insecure stack

Stack

Secure Compilation of J+E

Proof Strategy
Exceptions

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

Insecure stack
- $g$
- $i$

Record passed exceptions

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

$\text{throw } e$
Exceptions

Record passed exceptions
Check that exception could be thrown
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
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

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

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

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

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

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

VERY COMPLEX!

Marco Patrignani, Dave Clarke, Frank Piessens
Secure Compilation

\[ C_1 \sim_{J+E} C_2 \iff C'_1 \sim_{A+I} C'_2 \]

\[
(\forall C.\, C[C_1] \uparrow \iff C[C_2] \uparrow) \iff (\forall M.\, M[C'_1] \uparrow \iff M[C'_2] \uparrow)
\]
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

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

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

\[ C_1 \overset{\approx}{\sim}^J+E C_2 \]

\[ C_1 \downarrow \overset{\approx}{\sim}^A+I C_2 \downarrow \]
Secure Compilation

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

$C_1 \sim_{J+E} C_2 \implies$

$C_1 \Downarrow \sim_{A+I} C_2$

$\updownarrow$

$\text{Traces}(C_1) = \text{Traces}(C_2)$

Fully Abstract Trace Semantics
Secure Compilation

\[ C_1 \not\approx^J E C_2 \iff \text{Traces}(C_1^\downarrow) \neq \text{Traces}(C_2^\downarrow) \]
Secure Compilation

\[ C_1 \not\equiv_{\mathcal{J+E}} C_2 \quad \checkmark \quad \text{Traces}(C_1) \neq \text{Traces}(C_2) \]
Secure Compilation

$$C_1 \not\equiv_{\text{J+E}} C_2$$

$$C_1 \sim_{A+I} C_2$$

$$\Updownarrow$$

$$\text{Traces}(C_1) = \text{Traces}(C_2)$$

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. Recent Work
Modular, Secure Compilation

- current model has a single secure module
Modular, Secure Compilation

- current model has a single secure module
- assembly-level linking of securely-compiled modules is not investigated
Modular, Secure Compilation

- current model has a single secure module
- assembly-level linking of securely-compiled modules is not investigated
- can we link securely-compiled modules securely?
Modular, Secure Compilation

- current model has a single secure module
- assembly-level linking of securely-compiled modules is not investigated
- can we link securely-compiled modules securely?
- what attacks arise in the presence of linking?
Recent Work

Logical-relations Based Proof Technique

- Devirese et al. @ POPL’16
Logical-relations Based Proof Technique

- Devirese et al. @ POPL’16
- proof technique for fully-abstract compilation based on logical relations
Logical-relations Based Proof Technique

- Devirese et al. @ POPL’16
- proof technique for fully-abstract compilation based on logical relations
- check out the video on the popl website!
Secure Compilation Statement

- does full abstraction mean security?
Secure Compilation Statement

- does full abstraction mean security?
- can it express all security properties?
Secure Compilation Statement

- does full abstraction mean security?
- can it express all security properties?
- is there a better/more precise notion of secure compilation?
Secure Compilation Statement

- does full abstraction mean security?
- can it express all security properties?
- is there a better/more precise notion of secure compilation?
- can we relate it to hyperproperties (i.e., properties over sets of traces)?
Questions

Thank you!

Qs ?