The Tome of Secure Compilation
Fully-Abstract Compilation to Protected Modules Architectures

Marco Patrignani

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

Supervisors: Dave Clarke & Frank Piessens
The Tome of Secure Compilation

Fully-Abstract Compilation to Protected Modules Architectures
The Tome of Secure Compilation

Fully-Abstract Compilation to Protected Modules Architectures
The Tome of Secure Compilation

Fully-Abstract Compilation to Protected Modules Architectures
The Tome of Secure Compilation

Fully-Abstract Compilation to Protected Modules Architectures
The Tome of Secure Compilation

Fully-Abstract Compilation to Protected Modules Architectures
The Tome of Secure Compilation

Fully-Abstract Compilation to Protected Modules Architectures
What is PMA?
A security architecture

What is PMA?
What is PMA?

A security architecture

Isolation @ assembly
A+I: Untyped Assembly + PMA

<table>
<thead>
<tr>
<th>Address</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0001</td>
<td>call func. at 0xb53</td>
</tr>
<tr>
<td>0x0002</td>
<td>write r0 at 0x0b55</td>
</tr>
<tr>
<td>0x0b52</td>
<td>write r0 at 0x0b55</td>
</tr>
<tr>
<td>0x0b53</td>
<td>write r0 at 0x0001</td>
</tr>
<tr>
<td>0x0b54</td>
<td>call func. at 0x0002</td>
</tr>
<tr>
<td>0x0b55</td>
<td>...</td>
</tr>
<tr>
<td>0xab00</td>
<td>jump to 0x0001</td>
</tr>
<tr>
<td>0xab01</td>
<td>return to 0x0b53</td>
</tr>
<tr>
<td>0xab02</td>
<td>...</td>
</tr>
</tbody>
</table>

memory space

protected module (isolated)
split in code and data
protected code is unrestricted
unprotected code is restricted
entry points for communication (■)
A+I: Untyped Assembly + PMA

0x0001 call func. at 0xb53
0x0002 write r0 at 0x0b55

...  

0xb52 write r0 at 0xb55
0xb53 write r0 at 0x0001
0xb54 call func. at 0x0002
0xb55 ...

...  

0xab00 jump to 0x0001
0xab01 return to 0xb53
0xab02 ...

- memory space
- protected module (isolated)
A+I: Untyped Assembly + PMA

- memory space
- protected module (isolated)
- split in code and data
A+I: Untyped Assembly + PMA

- Memory space
- Protected module (isolated)
- Split in code and data
- Protected code is unrestricted

```plaintext
0x0001 call func. at 0xb53
0x0002 write r0 at 0xb55
...
0xb52 write r0 at 0xb55
0xb53 write r0 at 0x0001
0xb54 call func. at 0x0002
0xb55 ...

0xab00 jump to 0x0001
0xab01 return to 0xb53
0xab02 ...
```
A+I: Untyped Assembly + PMA

0x0001 call func. at 0xb53
0x0002 write r₀ at 0x0b55
...
0x0b52 write r₀ at 0x0b55
0x0b53 write r₀ at 0x0001
0x0b54 call func. at 0x0002
0x0b55 ...

- memory space
- protected module (isolated)
- split in code and data
- protected code is unrestricted

Marco Patrignani
A+I: Untyped Assembly + PMA

- call func. at 0xb53
- write r0 at 0xb55
- write r0 at 0xb55
- call func. at 0x0002
- jump to 0x0001
- return to 0xb53

- memory space
- protected module (isolated)
- split in code and data
- protected code is unrestricted

- r/w/x memory space
- protected module (isolated)
- split in code and data
- protected code is unrestricted
A+I: Untyped Assembly + PMA

- $0x0001$: call func. at $0xb53$
- $0x0002$: write $r_0$ at $0xb55$
- $0x0b52$: write $r_0$ at $0xb55$
- $0x0b53$: write $r_0$ at $0x0001$
- $0x0b54$: call func. at $0x0002$
- $0x0b55$: ...

- $0xab00$: jump to $0x0001$
- $0xab01$: return to $0xb53$
- $0xab02$: ...

- memory space
- protected module (isolated)
- split in code and data
- protected code is unrestricted
- unprotected code is restricted
A+I: Untyped Assembly + PMA

- 0x0001 call func. at 0xb53
- 0x0002 write r_0 at 0x0b55
- ...
- 0x0b52 write r_0 at 0x0b55
- 0x0b53 write r_0 at 0x0001
- 0x0b54 call func. at 0x0002
- 0x0b55 ...
- ...
- 0xab00 jump to 0x0001
- 0xab01 return to 0x0b53
- 0xab02 ...

- memory space
- protected module (isolated)
- split in code and data
- protected code is unrestricted
- unprotected code is restricted
A+I: Untyped Assembly + PMA

0x0001 call func. at 0xb53
0x0002 write r0 at 0x0b55

... 

0xb52 write r0 at 0xb55
0xb53 write r0 at 0x0001
0xb54 call func. at 0x0002

0xb55 ... 

-memory space
-protected module (isolated)
-split in code and data
-protected code is unrestricted
-unprotected code is restricted
A+I: Untyped Assembly + PMA

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

```assembly
0x0001 call func. at 0xb53
0x0002 write r0 at 0x0b55
...
0xb52 write r0 at 0x0b55
0xb53 write r0 at 0x0001
0xb54 call func. at 0x0002
0xb55 ...
...
0xab00 jump to 0x0001
0xab01 return to 0x0b53
0xab02 ...
```
A+I: Untyped Assembly + PMA

- memory space
- protected module (isolated)
- split in code and data
- protected code is unrestricted
- unprotected code is restricted
- entry points for communication (■)
**Reasoning about A+I**

0x0001  call func. at 0xb52  
0x0002  write r0 at 0x0b55  

...  

0x0b52  write r0 at 0x0b55  
0x0b53  write r0 at 0x0001  
0x0b54  call func. at 0x0002  

0x0b55  ...  

...  

0xab00  jump to 0x0001  
0xab01  return to 0x0b53  
0xab02  ...

- interest in the behaviour of the module
Reasoning about A+I

- interest in the behaviour of the module: this one

<table>
<thead>
<tr>
<th>Address</th>
<th>Action</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0b52</td>
<td>write r&lt;sub&gt;0&lt;/sub&gt; at 0x0b55</td>
<td></td>
</tr>
<tr>
<td>0x0b53</td>
<td>write r&lt;sub&gt;0&lt;/sub&gt; at 0x0001</td>
<td></td>
</tr>
<tr>
<td>0x0b54</td>
<td>call func. at 0x0002</td>
<td></td>
</tr>
<tr>
<td>0x0b55</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Reasoning about A+I

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0001</td>
<td>call func. at 0xb52</td>
</tr>
<tr>
<td>0x0002</td>
<td>write r0 at 0x0b55</td>
</tr>
<tr>
<td>0xb52</td>
<td>write r0 at 0x0b55</td>
</tr>
<tr>
<td>0xb53</td>
<td>write r0 at 0x0001</td>
</tr>
<tr>
<td>0xb54</td>
<td>call func. at 0x0002</td>
</tr>
<tr>
<td>0xb55</td>
<td>...</td>
</tr>
<tr>
<td>0xb52</td>
<td>write r0 at 0x0b55</td>
</tr>
<tr>
<td>0xb53</td>
<td>write r0 at 0x0001</td>
</tr>
<tr>
<td>0xb54</td>
<td>call func. at 0x0002</td>
</tr>
<tr>
<td>0xb55</td>
<td>...</td>
</tr>
<tr>
<td>0xab00</td>
<td>jump to 0x0001</td>
</tr>
<tr>
<td>0xab01</td>
<td>return to 0x0b53</td>
</tr>
<tr>
<td>0xab02</td>
<td>...</td>
</tr>
</tbody>
</table>

- interest in the behaviour of the module: this one
- need to consider the rest
### Reasoning about A+I

<table>
<thead>
<tr>
<th>Address</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0001</td>
<td>call func. at 0xb52</td>
</tr>
<tr>
<td>0x0002</td>
<td>write r0 at 0x0b55</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0b52</td>
<td>write r0 at 0x0b55</td>
</tr>
<tr>
<td>0x0b53</td>
<td>write r0 at 0x0001</td>
</tr>
<tr>
<td>0x0b54</td>
<td>call func. at 0x0002</td>
</tr>
<tr>
<td>0x0b55</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>0xab00</td>
<td>jump to 0x0001</td>
</tr>
<tr>
<td>0xab01</td>
<td>return to 0x0b55</td>
</tr>
<tr>
<td>0xab02</td>
<td>...</td>
</tr>
</tbody>
</table>

- interest in the behaviour of the module: this one
- need to consider the *rest*

**Problem**

There is a lot of *the rest*
Reasoning about A+I

There is a lot of the rest

Solution

Use Trace semantics

Problem

There is a lot of the rest
What is trace semantics?

A trace is a sequence of actions:
What is trace semantics?

- A trace is a sequence of actions:

  call a function
  write something somewhere
  ···
A trace is a sequence of actions:
What is trace semantics?

A trace is a sequence of actions:
What is trace semantics?

- A trace is a sequence of actions: 
- Actions describe an observable behaviour abstractly
  
  \[
  \begin{cases}
  \text{call a function} \\
  \text{write something somewhere} \\
  \text{...}
  \end{cases}
\]
What is trace semantics?

- A trace is a sequence of actions: call a function, write something somewhere, …
- Actions describe an observable behaviour abstractly
- The behaviour of some code is a set of traces
Trace semantics for A+I

- 0x0001  call func. at 0xb52
- 0x0002  write r₀ at 0x0b55

disregard the rest

- 0x0b52  write r₀ at 0x0b55
- 0x0b53  write r₀ at 0x0001
- 0x0b54  call func. at 0x0002
- 0x0b55  ...

- 0xab00  jump to 0x0001
- 0xab01  return to 0x0b53
- 0xab02  ...
Trace semantics for A+I

- disregard the rest

<table>
<thead>
<tr>
<th>Address</th>
<th>Action</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0b52</td>
<td>write r₀</td>
<td>at 0x0b55</td>
</tr>
<tr>
<td>0x0b53</td>
<td>write r₀</td>
<td>at 0x0001</td>
</tr>
<tr>
<td>0x0b54</td>
<td>call func.</td>
<td>at 0x0002</td>
</tr>
<tr>
<td>0x0b55</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Trace semantics for A+I

- disregard the rest
- abstract its behaviour

from the module perspective:

<table>
<thead>
<tr>
<th>Address</th>
<th>Action</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0b52</td>
<td>write r0 at 0x0b55</td>
<td></td>
</tr>
<tr>
<td>0x0b53</td>
<td>write r0 at 0x0001</td>
<td></td>
</tr>
<tr>
<td>0x0b54</td>
<td>call func. at 0x0002</td>
<td></td>
</tr>
<tr>
<td>0x0b55</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Trace semantics for A+I

call args.

0x0b52 write r0 at 0x0b55
0x0b53 write r0 at 0x0001
0x0b54 call func. at 0x0002
0x0b55 ...

- disregard the rest
- abstract its behaviour from the module perspective:

1. jump to an entry (call/return)
Trace semantics for A+I

call args.

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0b52</td>
<td>write r0 at 0x0b55</td>
</tr>
<tr>
<td>0x0b53</td>
<td>write r0 at 0x0001</td>
</tr>
<tr>
<td>0x0b54</td>
<td>call func. at 0x0002</td>
</tr>
<tr>
<td>0x0b55</td>
<td>...</td>
</tr>
</tbody>
</table>

- disregard the rest
- abstract its behaviour 
  from the module perspective:
  1. jump to an entry (call/return)
- abstract the module behaviour 
  from the rest perspective:
Trace semantics for A+I

- disregard the rest
- abstract its behaviour
  from the module perspective:
  1. jump to an entry (call/return)
- abstract the module behaviour
  from the rest perspective:
  1. call/return outside
Trace semantics for A+I

- disregard the rest
- abstract its behaviour
  from the module perspective:
    1. jump to an entry
       (call/return)
- abstract the module
  behaviour
  from the rest perspective:
    1. call/return outside
    2. read/write outside

Marco Patrignani
Trace semantics for this example:

```
0x0b52  write r0 at 0x0b55
0x0b53  write r0 at 0x0001
0x0b54  call func. at 0x0002
0x0b55  ...
```

Trace of this example:

```
call 0x0b52 (args) · write 0x0001 (arg) · call 0x0002 (args)
```
Trace semantics for this example

<table>
<thead>
<tr>
<th>Address</th>
<th>Operation</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0b52</td>
<td>write r0 at 0x0b55</td>
<td></td>
</tr>
<tr>
<td>0x0b53</td>
<td>write r0 at 0x0001</td>
<td></td>
</tr>
<tr>
<td>0x0b54</td>
<td>call func. at 0x0002</td>
<td></td>
</tr>
<tr>
<td>0x0b55</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Trace of this example:

- call 0x0b52 (args)
- write 0x0001 (arg)
- call 0x0002 (args)
Correctness of the Trace semantics

- formalise the trace semantics
Correctness of the Trace semantics

\[ TR = \left\{ \alpha = \begin{cases} \rightarrow \\
\text{call} \ a \ (\bar{v}) \\
\text{ret} \ v \\
\text{write}(a, v) \\
\text{read}(a, v) \\
\end{cases} \right\} \]

- formalise the trace semantics

define traces of a program
prove the semantics to be as precise as what the rest captured
this defines "what the rest captures" (i.e., contextual equivalence)
Correctness of the Trace semantics

\[ TR = \left\{ \alpha = \begin{array}{l}
\rightarrow \\
\text{call } a(\overline{v}) \\
\text{ret } v \\
\text{write}(a, v) \\
\text{read}(a, v)
\end{array} \right\} \]

- formalise the trace semantics
- define traces of a program
Correctness of the Trace semantics

\[ \text{TR} = \left\{ \alpha = \left\{ \begin{array}{l}
\rightarrow \\
call a(\overline{v}) \\
\text{ret } v \\
\text{write}(a,v) \\
\text{read}(a,v) \\
\alpha \rightarrow
\end{array} \right\} \right\} \]

- formalise the trace semantics
- define traces of a program

\[
\text{Traces}_{A+l}^S(P) = \{ \overline{\alpha} \mid P \xrightarrow{\alpha} P' \} 
\]
Correctness of the Trace semantics

\[ TR = \left\{ \alpha = \begin{cases} \rightarrow \\
\text{call } a(\overline{v}) \\
\text{ret } v \\
\text{write}(a, v) \\
\text{read}(a, v) \\
\alpha \rightarrow \\
\end{cases} \right\} \]

\[ \text{Traces}_{A+I}(P) = \{ \overline{\alpha} \mid P \xrightarrow{\overline{\alpha}} P' \} \]

- formalise the trace semantics
- define traces of a program
- prove the semantics to be as precise as what the rest captured
Correctness of the Trace semantics

$$\text{TR} = \begin{cases} \alpha = \{ \rightarrow \text{call} \ a \ (\overline{v}) \}, \\ \rightarrow \text{ret} \ v \\ \text{write}\ (a, v) \\ \text{read}\ (a, v) \} \end{cases}$$

\[ \text{Traces}^S_{A+I}(P) = \{ \overline{\alpha} \mid P \xRightarrow{\overline{\alpha}} P' \} \]

- formalise the trace semantics
- define traces of a program
- prove the semantics to be as precise as what the rest captured

\[ P_1 \simeq P_2 \iff \text{Traces}^S_{A+I}(P_1) = \text{Traces}^S_{A+I}(P_2) \]
Correctness of the Trace semantics

\[ TR = \begin{cases} \alpha = \begin{cases} \rightarrow \\
\text{call } a(\overline{v}) \\
\text{ret } v \\
\text{write}(a, v) \\
\text{read}(a, v) \\
\alpha \Rightarrow \alpha \end{cases} \end{cases} \]

\[ \text{Traces}_{A+1}^{S}(P) = \{ \overline{\alpha} \mid P \xrightarrow{\overline{\alpha}} P' \} \]

- formalise the trace semantics
- define traces of a program
- prove the semantics to be as precise as what the rest captured
- this defines “what the rest captures” (i.e., contextual equivalence)

\[ P_1 \not\sim P_2 \iff \text{Traces}_{A+1}^{S}(P_1) = \text{Traces}_{A+1}^{S}(P_2) \]
Trace Semantics with Equivalence Classes

Equivalence classes!!
What is secure compilation?
Compilation

What is compilation?
Compilation

What is compilation?
Compilation

What is compilation?

source
Compilation

What is compilation?
What is compilation?

source

target
Correct Compilation

What is **correct** compilation?
Correct Compilation

What is *correct* compilation?

[Diagram showing flags of the United Kingdom and France, indicating a relationship between them.]
Correct Compilation

What is **correct** compilation?
Correct Compilation

What is **correct** compilation?
Correct Compilation

What is **correct** compilation?
What is correct compilation?
Correct Compilation

What is *correct* compilation?
What is secure compilation?
What is **secure** compilation?
What is **secure** compilation?
What is secure compilation?
What is secure compilation?
Secure Compilation

What is secure compilation?
What is secure compilation?
Secure Compilation

What is **secure** compilation?

Source security properties
Secure Compilation

What is secure compilation?

Source security properties → Hold @ target
What is **secure** compilation?

- Source security properties are preserved.
- Fully abstract compilation!

---

Marco Patrignani

The Tome of Secure Compilation
J+E: Java-like Language

- 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;
```
component-based
private fields
programming to an interface
exceptions

Q: What is secure in 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

- component-based
- private fields
- programming to an interface
- exceptions

Q: What is secure in this code?

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

cpyackage 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

- component-based
- private fields
- programming to an interface
- exceptions

Q: What is secure in this code?

Q: How do we securely compile this code?

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

top-level: 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;
Dynamic dispatch
v-tables
Secure stack
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

```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;
```
PMA for Secure Compilation

Source level

O1
O2

Ext 1
Ext 2

Protect against low-level attackers

Target code is vulnerable without PMA
PMA for Secure Compilation

```
O1 ← O1.createAccount() ← Ext 1
O2    ← Ext 2
```

Source level

Target level

Protect against low-level attackers

Target code is vulnerable without PMA
PMA for Secure Compilation

Source level:
O1
O2
O3

Protected against low-level attackers:
Ext 1
Ext 2

Target level:
Ext 1
Ext 2
Trace Semantics for PMA

Secure Compilation to PMA

Marco Patrignani

The Tome of Secure Compilation
PMA for Secure Compilation

Source level

O1
O2
O3

03.counter

Ext 1
Ext 2

Target level

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

Source level

O1
O2
O3

03.counter

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

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

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

Marco Patrignani
PMA for Secure Compilation

Source level

- O1
- O2
- O3

Target level

- O1
- O2
- O3

Ext 1

Ext 2

01.createAccount()

Marco Patrignani
Trace Semantics for PMA
Secure Compilation to PMA

PMA for Secure Compilation

Source level
O1
O2
O3

Target level
O1
O2
O3

Ext 1
Ext 2

⇒

return O3

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

Marco Patrignani
The Tome of Secure Compilation 16/29
PMA for Secure Compilation

Source level

O1
O2
O3

Ext 1
Ext 2

Target level

O1↓
O2↓
O3↓

03↓.counter

Ext 1↓
Ext 2↓
PMA for Secure Compilation

- Protect against low-level attackers

Source level:
- O1
- O2
- O3

Target level:
- O1
- O2
- O3

Ext 1
Ext 2

Ext 1↓
Ext 2↓

03↓.counter
PMA for Secure Compilation

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

Source level

O1
O2
O3

Ext 1
Ext 2

Target level

O1↓
O2↓
O3↓

Ext 1↓
Ext 2↓

03↓ counter
Q: Is that all?
Q: : Is that all?

0x0001 Unprotected stack
0x0002
...
0xb52
0xb53
0xb54
0xb55
Q: : Is that all?

- protected stack

Leaks stack contents

0x0001
0x0002

0x0b52
0x0b53
0x0b54
0x0b55
Q: Is that all?

protected stack

Leaks stack contents

0x0001
0x0002
...

0x0b52
0x0b53
0x0b54
0x0b55
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
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
...
0xb52
0xb53 Protected stack
0xb54 je ...
0xb55
```

\( f_{ZS} = 0/1 \)
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
- 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
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
... 0xb52
0xb53 Protected stack
0xb54
0xb55

check r0 = 1/0
Dynamic Memory Allocation

Source level

O1
O2

Ext 1
Ext 2

Target level

O1↓
O2↓

Ext 1↓
Ext 2↓
Dynamic Memory Allocation

```
O1 createAccount() ⇒
O1 ↓ O2 ↓

01.createAccount()
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
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Target level

O1↓
O2↓

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

Marco Patrignani
Dynamic Memory Allocation

Source level

\[ \text{O1} \rightarrow \text{return O3} \rightarrow \text{Ext 1} \]
\[ \text{O2} \rightarrow \text{Ext 2} \]
\[ \text{O4} \]
\[ \text{O3} \]

Target level

\[ \text{O1} \downarrow \rightarrow \text{Ext 1} \downarrow \]
\[ \text{O2} \downarrow \rightarrow \text{Ext 2} \downarrow \]
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Target level

O1\downarrow .createAccount()
O2\downarrow

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

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

\[
\begin{array}{c}
O1 \\
O2 \\
O4 \\
O3 \\
\end{array}
\]

Target level

\[
\begin{array}{c}
O1\downarrow \\
O2\downarrow \\
O4\downarrow \\
O3\downarrow \\
\end{array}
\]

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
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↓
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Target level

0x001
0x005
0x009
0x00C

Ext 1
Ext 2

return 0x00C

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
Ext 1↓
Ext 2↓
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Target level

0x001
0x005
0x009
0x00C

Ext 1
Ext 2

0x009.createAccount()
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Target level

0x001
0x005
0x009
0x00C

0x009.createAccount()

Ext 1
Ext 2

04.createAccount()

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

Marco Patrignani
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

Object id guessing
- map Oid to natural numbers

Source level

Ext 1
01
02
04
03

Target level

Ext 1\downarrow
0x001 ↦ 1
0x005 ↦ 2

Ext 2↓
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Target level

0x001 1
0x005 2

Ext 1
Ext 2

- Object id guessing
- map Oid to natural numbers

1.createAccount()
Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Target level

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

Ext 1
Ext 2

Ext 1↓
Ext 2↓

- 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 $\mapsto$ 3

Ext 1
Ext 2

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

Marco Patrignani
The Tome of Secure Compilation 18/29
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

Marco Patrignani
Dynamic Memory Allocation

Source level

O1
O2
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
The Tome of Secure Compilation 18/29
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

Marco Patrignani
The Tome of Secure Compilation 18/29
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 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

Marco Patrignani
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

2.createAccount()
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

Marco Patrignani
The Tome of Secure Compilation
Dynamic Memory Allocation

Source level

O1 : Account
O2 : Pair
O4
O3

Ext 1
Ext 2

Target level

0x001 ↦ 1 2.createAccount()
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 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 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
Why is this secure?

Source level

\[
\text{class ...}
\text{secret} = 1
\]

Target level
Why is this secure?

Source level

```
class ...
secret = 1
```  

Target level

```
class ...
secret = 1
```
Why is this secure?

Source level

```
class ... secret = 1
```

Target level

```
class ... secret = 1
```

- we only add checks

---

This is not trivial, but we use trace semantics and prove the contrapositive.
Why is this secure?

- we only add checks

**Source level**

```plaintext
class ...
secret = 1
```

**Target level**

```plaintext
class ...
\[
\text{secret} = 1
\]
```
Why is this secure?

Source level

\[
\begin{align*}
\text{class ... secret} &= 1 \\
\text{class ... secret} &= 2
\end{align*}
\]

Target level

\[
\begin{align*}
\text{class ... secret} &= 1 \\
\text{class ... secret} &= 2
\end{align*}
\]

- we only add checks
- we need to prove full abstraction

\( \simeq \)
Why is this secure?

- we only add checks
- we need to prove full abstraction
- this is trivial
Why is this secure?

- we only add checks
- we need to prove full abstraction
- this is trivial

Source level:

class ...
secret = 1

≈

class ...
secret = 2

Target level:

class ...
secret = 1

≈

class ...
secret = 2

class ...
secret = 2

class ...
secret = 1
Why is this secure?

- we only add checks
- we need to prove full abstraction
- this is trivial
- this is NOT
Why is this secure?

- we only add checks
- we need to prove full abstraction
- this is trivial
- this is NOT
- but we use trace semantics
Why is this secure?

- we only add checks
- we need to prove full abstraction
- this is trivial
- this is **NOT**
- but we use trace semantics
Why is this secure?

- we only add checks
- we need to prove full abstraction
- this is trivial
- this is **NOT**
- but we use trace semantics
Why is this secure?

- we only add checks
- we need to prove full abstraction
- this is trivial
- this is NOT
- but we use trace semantics

Source level

Target level

class ... secret = 1  \sim \  class ... secret = 2

\[\begin{align*}
\text{class} & \ldots \\
\text{secret} & = 1
\end{align*}\]

\[\begin{align*}
\text{class} & \ldots \\
\text{secret} & = 2
\end{align*}\]
Why is this secure?

Source level

Class ...
secret = 1

Target level

Class ...
secret = 1

Class ...
secret = 2

- we only add checks
- we need to prove full abstraction
- this is trivial
- this is **NOT**
- but we use trace semantics
- and prove the contrapositive
Why is this secure?

- we only add checks
- we need to prove full abstraction
- this is trivial
- this is **NOT**
- but we use trace semantics
- and prove the contrapositive
Why is this secure?

Source level

\[
\text{class ... secret} = 1 \quad \not\equiv \quad \text{class ... secret} = 2
\]

Target level

\[
\begin{align*}
\text{class ... secret} &= 1 \\
\gamma? \quad \gamma! \quad \gamma!
\end{align*}
\]

\[
\begin{align*}
\text{class ... secret} &= 2 \\
\gamma? \quad \gamma! \quad \gamma!
\end{align*}
\]

- we only add checks
- we need to prove full abstraction
- this is trivial
- this is **NOT**
- but we use trace semantics
- and prove the contrapositive
Why can we do this?

Source

Target level

class ...
secret = 1
Why can we do this?

- analyse incoming traces

Source

Target level

class ...
secret = 1
Why can we do this?

Program $P \approx \gamma$?

- analyse incoming traces
- let the *good* ones is
Why can we do this?

- analyse incoming traces
- let the *good* ones is
Why can we do this?

Program $P \not\approx \gamma$?

- analyse incoming traces
- let the *good* ones in
- block the *bad* ones
Why can we do this?

Program $P \not\equiv \gamma$?

Attack detected!

- analyse incoming traces
- let the *good* ones is
- block the *bad* ones
Why can we do this?

- analyse incoming traces
- let the \textit{good} ones is
- block the \textit{bad} ones

\begin{itemize}
\item class ...
\item secret = 1
\end{itemize}
Why can we do this?

Program $P \approx \gamma$?

- analyse incoming traces
- let the good ones in
- block the bad ones
Why can we do this?

Program $P \not\approx \gamma$?

- analyse incoming traces
- let the good ones is
- block the $\textsf{bad}$ ones
Secure Compilation with Equivalence classes

Equivalence classes!!
PMA is an interesting security architecture
PMA is an interesting security architecture

- you can reason about it with simple traces
PMA is an interesting security architecture
you can reason about it with simple traces
you can securely compile Java-like code to it
Conclusion

- PMA is an interesting security architecture
- you can reason about it with simple traces
- you can securely compile Java-like code to it
- you can link that code to other secure code and preserve security
Conclusion

Thank you!

- It's an interesting security architecture
- you can reason about it with simple traces
- you can securely compile Java-like code to it
- you can link that code to other secure code and preserve security
Conclusion

Trace Semantics for PMA
Secure Compilation to PMA

Thank you!

- PMA is an interesting security architecture
- you can reason about it with simple traces
- you can securely compile Java-like code to it
- you can link that code to other secure code and preserve security

Qs ?
Compiler compositionality
Labels \( L \)
\[
l ::= \tau \mid a
\]

Observable actions \( A \)
\[
a ::= g? \mid d! \mid \sqrt{}
\]

Actions \( G \)
\[
g ::= \text{call} \ p(r; f) \mid \text{ret} \ p(r; f)
\]

Prefixable actions \( D \)
\[
d ::= g \mid o(a, v).d
\]

Prefixes \( O \)
\[
o ::= \text{read} \mid \text{write}
\]
Labels of Traces $S_{A+1}$

Labels

\[ \lambda ::= \tau \mid \alpha \]

Observable actions

\[ \alpha ::= \gamma? \mid \gamma! \mid \sqrt{\text{-}} \]

Actions

\[ \gamma ::= \text{call } \overline{\nu} \mid \text{ret } p \; \nu \]
\[
\text{StripNI}(\cdot, \cdot)
\]

\[
\begin{align*}
\text{StripNI}(\Theta, g) &= g \\
\text{StripNI}(\Theta, \text{write}(a, v) \cdot) &= \text{write}(a, v) \cdot' \\
&\quad \text{if StripNI}(\Theta, \cdot) = \cdot' \\
\text{StripNI}(\Theta, \text{read}(a, v) \cdot) &= \cdot' \\
&\quad \text{if StripNI}(\Theta, \cdot) = \cdot' \text{ and NI}(\Theta, a) \\
\text{StripNI}(\Theta, \text{read}(a, v) \cdot) &= \text{read}(a, v) \cdot' \\
&\quad \text{if StripNI}(\Theta, \cdot) = \cdot' \text{ and } \neg\text{NI}(\Theta, a)
\end{align*}
\]
\( \text{NI}(\Theta, a) \triangleq \forall v, w. \quad \Theta \xrightarrow{a_1} \Theta' \text{ and } \Theta \xrightarrow{a_2} \Theta'' \)

and \( a_1 = \overline{o(a', v') \text{read}(a, v) \varepsilon} \)

and \( a_2 = \overline{o(a', v') \text{read}(a, w) \varepsilon} \)

and \( \text{Tr-state}(\Theta') = \text{Tr-state}(\Theta'') \)
Definitions

Definition (Contextual equivalence for $A+I$)

Definition (Fully abstract trace semantics for $A+I$)

Given that $\text{Traces}_{A+I}^L(P) = \text{Tr-state}(\Theta_0(P))$,

\[ P_1 \simeq_{A+I} P_2 \iff P_1 \equiv_{A+I}^T P_2 \]

Definition (Fully abstract compilation)

For any two source-level components $C_1$ and $C_2$, we have:

\[ C_1 \simeq_{J+E} C_2 \iff \llbracket C_1 \rrbracket_{A+I} \simeq_{A+I} \llbracket C_2 \rrbracket_{A+I} \]
### Definition (Contextual preorder for A\(\text{IM}\))

\[ P_1 \preceq^\text{AIM}_\sigma P_2 \triangleq \Pr(\forall \mathcal{P}, O_1. \exists O_2. \mathbb{P}[P_1, O_1] \uparrow \iff \mathbb{P}[P_2, O_2] \uparrow) > \sigma. \]

### Definition (Contextual equivalence for A\(\text{IM}\))

\[ P_1 \simeq^\text{AIM}_\sigma P_2 \triangleq P_1 \preceq^\text{AIM}_\sigma P_2 \text{ and } P_2 \preceq^\text{AIM}_\sigma P_1. \]

### Definition (Trace equivalence for A\(\text{IM}\))

\[ P_1 \equiv^\text{AIM}_\sigma P_2 \triangleq \Pr(\text{Traces}_{\text{AIM}}(P_1, O_1) = \text{Traces}_{\text{AIM}}(P_2, O_2)) > \sigma. \]