

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

Me



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

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 0x0001
0x0b55    ...
:
:
0xab00    jmp 0xb53
0xab01    ...
```

- memory space

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

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

A+I: Untyped Assembly + PMA

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

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

r/w

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

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

A+I: Untyped Assembly + PMA

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

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

r/x

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

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

A+I: Untyped Assembly + PMA

```
0x0001    call 0xb53
0x0002    movs r0 0xb55
:
0x0b52    movs r0 0xb55
0x0b53    call 0x0002
0x0b54    movs r0 0x0001
0x0b55    ...
:
0xab00    jmp 0xb53
0xab01    ...
```

r/w/x

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

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    ...
```

r/w/x

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

A+I: Untyped Assembly + PMA

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

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

r/w/x

```
...  
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

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

r/w/x

```
0x0b52    movs r0 0xb55  
0x0b53    call 0x0002  
0x0b54    movs r0 0x0001  
0x0b55    ...
```

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

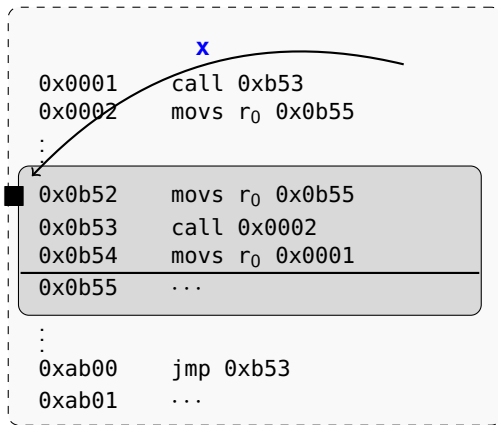
- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted

A+I: Untyped Assembly + PMA

```
0x0001    call 0xb53
0x0002    movs r0 0xb55
...
0xb52     movs r0 0xb55
0xb53     call 0x0002
0xb54     movs r0 0x0001
0xb55     ...
...
0xab00    jmp 0xb53
0xab01    ...
```

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

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 (■)

Trace Semantics for PMA

Trace Semantics for PMA

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

```
0x0b52    movi r0 1  
0x0b53    movi r1 0x0b56  
0x0b54    jl r1  
0x0b55    call 0xab01  
0x0b56    ret
```

```
...  
0xab01    ...
```

- behaviour in this case is:

Trace Semantics for PMA

0x0001 call 0xb52

0x0002 ...

...

0x0b52 movi r₀ 1

0x0b53 movi r₁ 0x0b56

0x0b54 jl r₁

0x0b55 call 0xab01

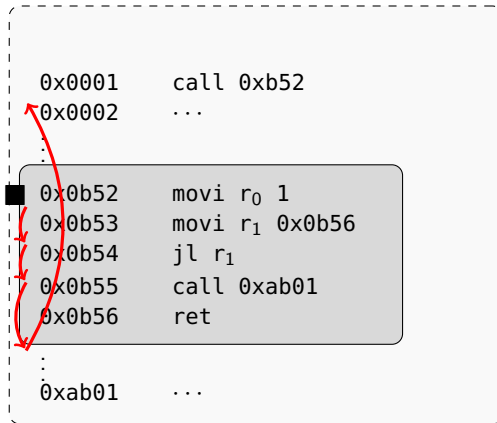
0x0b56 ret

...

0xab01 ...

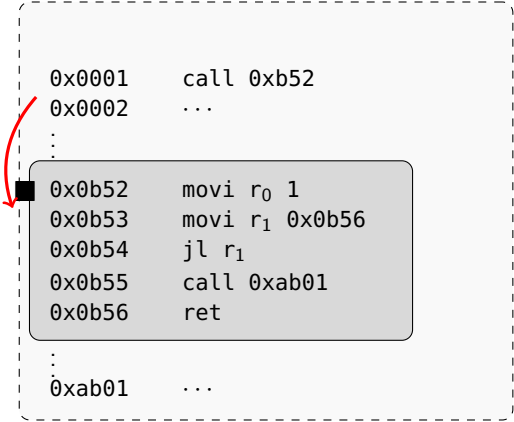
- behaviour in this case is:
call in

Trace Semantics for PMA



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

Trace Semantics for PMA



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

```
0x0b52    movi r0 1
0x0b53    movi r1 0x0b56
0x0b54    jl r1
0x0b55    call 0xab01
0x0b56    ret
```

```
:
```

```
0xab01    ...
```

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

Trace Semantics for PMA

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

```
0x0b52    movi r0 1  
0x0b53    movi r1 0x0b56  
0x0b54    jl r1  
0x0b55    call 0xab01  
0x0b56    ret
```

```
...  
0xab01    ...
```

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

Trace Semantics for PMA

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

```
0x0b52    movi r0 1  
0x0b53    movi r1 0x0b56  
0x0b54    jl r1  
0x0b55    call 0xab01  
0x0b56    ret
```

```
...  
0xab01    ...
```

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

Trace Semantics for PMA

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

```
0x0b52    movi r0 1  
0x0b53    movi r1 0x0b56  
0x0b54    jl r1  
0x0b55    call 0xab01  
0x0b56    ret
```

```
...  
0xab01    ...
```

- 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

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

```
1 package PI;
2   interface Account {
3       public createAccount() : Foo;
4   }
5   extern extAccount : Account;
6
7 package PE;
8   class AccountClass
9       implements PI.Account {
10       AccountClass() { counter = 0; }
11       public createAccount() : Account {
12           return new PE.AccountClass();
13       }
14
15       private counter : Int;
16   }
17   object extAccount : AccountClass;
```

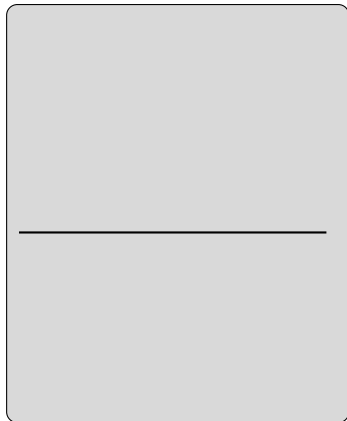
J+E: Java-like Language

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

Q: How to securely compile this code?

```
1 package PI;
2   interface Account {
3       public createAccount() : Foo;
4   }
5   extern extAccount : Account;
6
7 package PE;
8   class AccountClass
9       implements PI.Account {
10       AccountClass() { counter = 0; }
11       public createAccount() : Account {
12           return new PE.AccountClass();
13       }
14
15       private counter : Int;
16   }
17   object extAccount : AccountClass;
```

J+E: Java-like Language



```
1 package PI;
2   interface Account {
3     public createAccount() : Foo;
4   }
5   extern extAccount : Account;
6
7 package PE;
8   class AccountClass
9     implements PI.Account {
10    AccountClass() { counter = 0; }
11    public createAccount() : Account {
12      return new PE.AccountClass();
13    }
14
15    private counter : Int;
16  }
17  object extAccount : AccountClass;
```

J+E: Java-like Language

Dynamic dispatch

v-tables

Secure stack

```
1 package PI;
2   interface Account {
3       public createAccount() : Foo;
4   }
5   extern extAccount : Account;
6
7 package PE;
8   class AccountClass
9       implements PI.Account {
10      AccountClass() { counter = 0; }
11      public createAccount() : Account {
12          return new PE.AccountClass();
13      }
14
15      private counter : Int;
16  }
17  object extAccount : AccountClass;
```

J+E: Java-like Language

■ proxy to createAccount

Dynamic dispatch

v-tables

Secure stack

```
1 package PI;
2   interface Account {
3     public createAccount() : Foo;
4   }
5   extern extAccount : Account;
6
7 package PE;
8   class AccountClass
9     implements PI.Account {
10    AccountClass() { counter = 0; }
11    public createAccount() : Account {
12      return new PE.AccountClass();
13    }
14
15    private counter : Int;
16  }
17  object extAccount : AccountClass;
```

J+E: Java-like Language

■ proxy to createAccount

createAccount body

constructor

Dynamic dispatch

v-tables

Secure stack

extAccount
counter

```
1 package PI;
2   interface Account {
3     public createAccount() : Foo;
4   }
5   extern extAccount : Account;
6
7 package PE;
8   class AccountClass
9     implements PI.Account {
10    AccountClass() { counter = 0; }
11    public createAccount() : Account {
12      return new PE.AccountClass();
13    }
14
15    private counter : Int;
16  }
17  object extAccount : AccountClass;
```

PMA for Secure Compilation

Source level

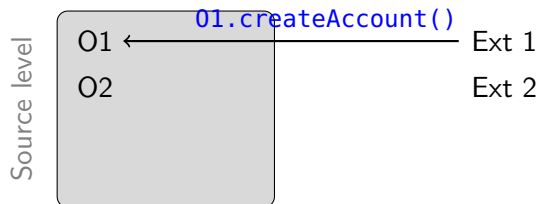
O1

O2

Ext 1

Ext 2

PMA for Secure Compilation



PMA for Secure Compilation

Source level

O1

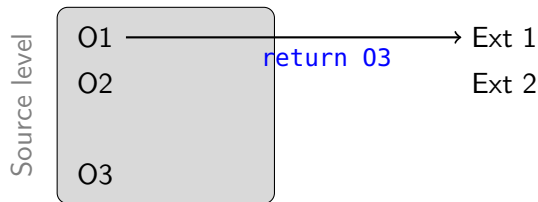
O2

O3

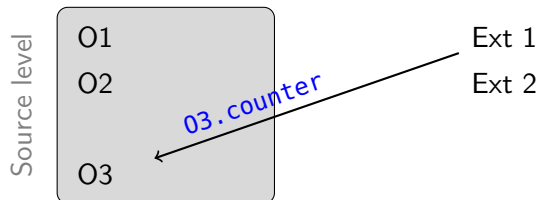
Ext 1

Ext 2

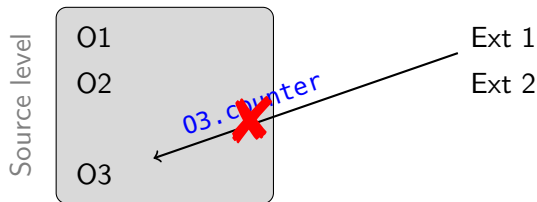
PMA for Secure Compilation



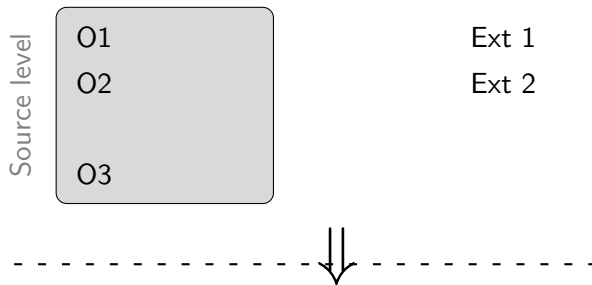
PMA for Secure Compilation



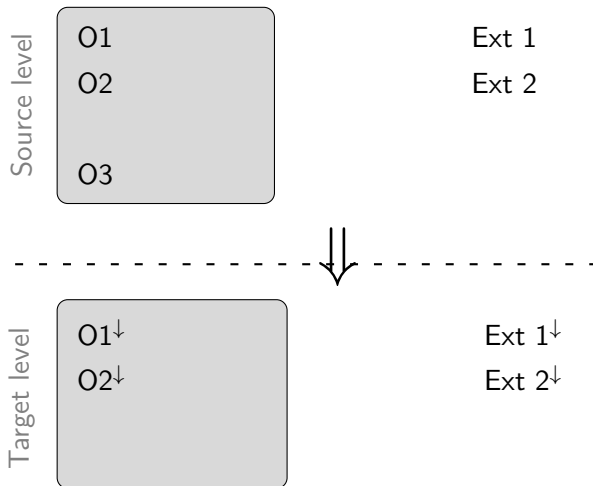
PMA for Secure Compilation



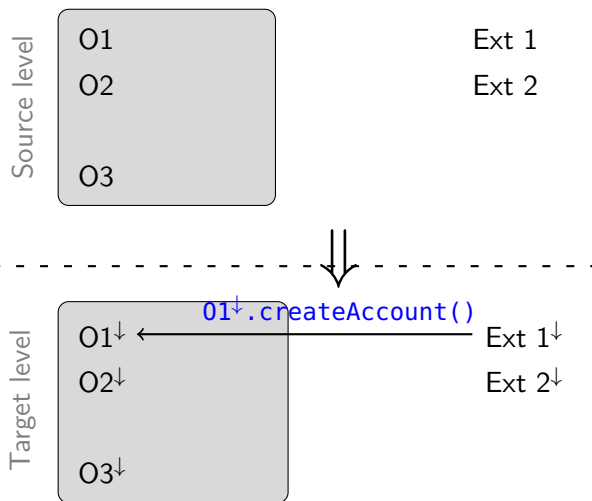
PMA for Secure Compilation



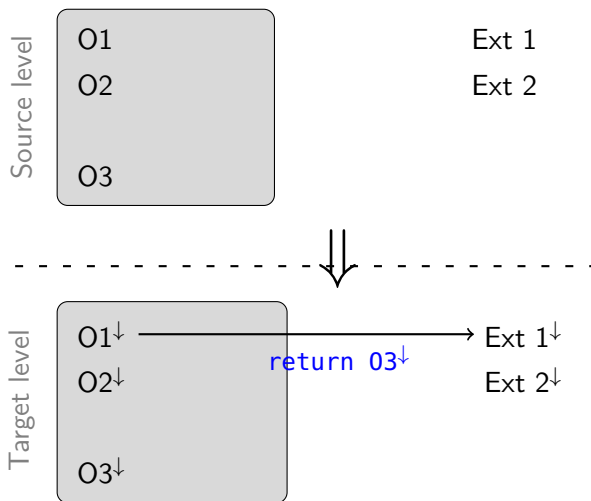
PMA for Secure Compilation



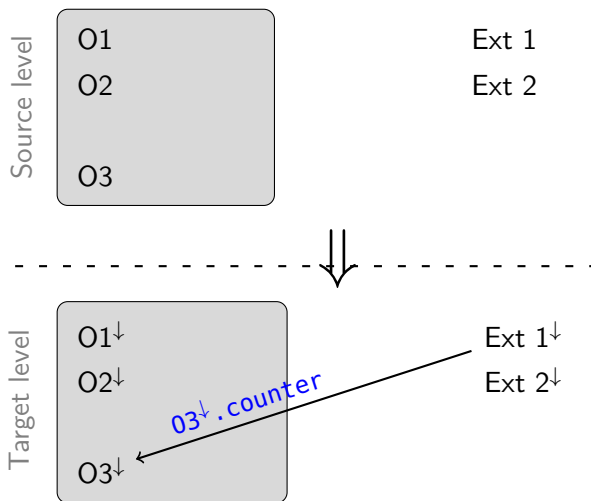
PMA for Secure Compilation



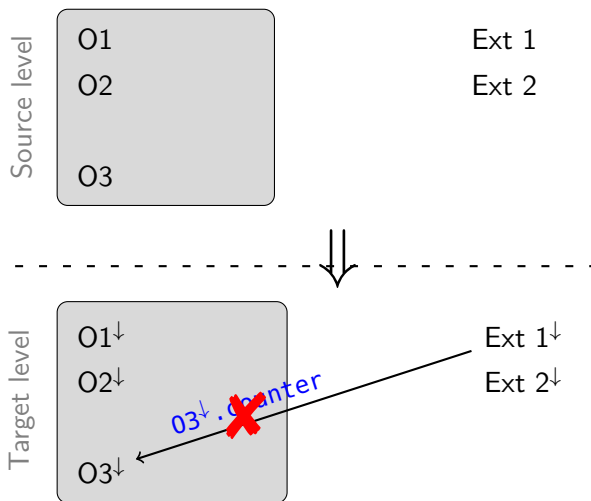
PMA for Secure Compilation



PMA for Secure Compilation

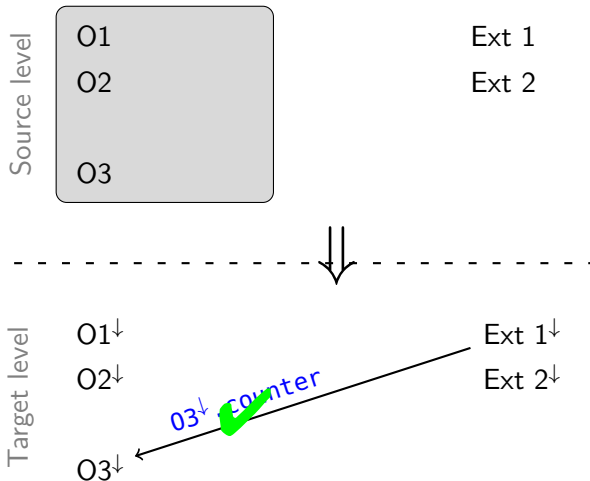


PMA for Secure Compilation



- Protect against low-level attackers

PMA for Secure Compilation



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

Secure Compilation of Outcalls

Q: : Is that all?

Secure Compilation of Outcalls

Q: : Is that all?

0x0001 Unprotected stack

0x0002

⋮

0x0b52

0x0b53

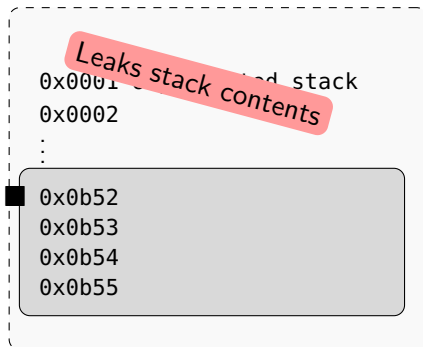
0x0b54

0x0b55

Secure Compilation of Outcalls

Q: : Is that all?

- protected stack



Secure Compilation of Outcalls

Q: : Is that all?

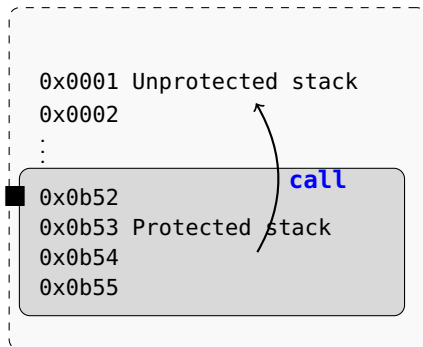
- protected stack



Secure Compilation of Outcalls

Q: : Is that all?

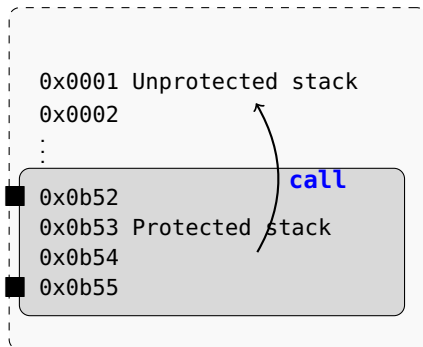
- protected stack
- returnback entry point



Secure Compilation of Outcalls

Q: : Is that all?

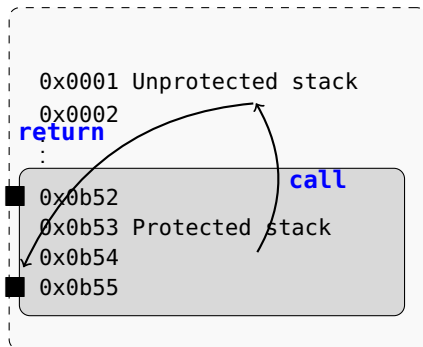
- protected stack
- returnback entry point



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



Secure Compilation of Outcalls

Q: : Is that all?

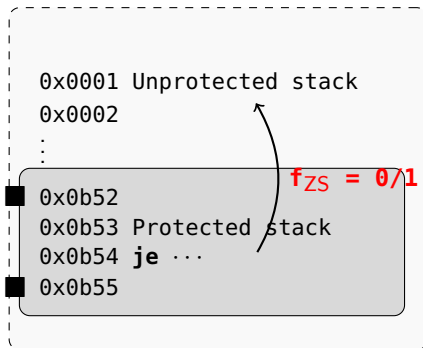
- protected stack
- returnback entry point
- reset flags and registers



Secure Compilation of Outcalls

Q: : Is that all?

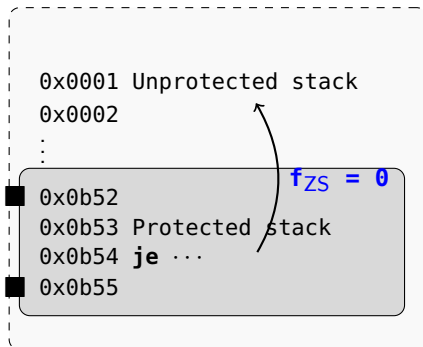
- protected stack
- returnback entry point
- reset flags and registers



Secure Compilation of Outcalls

Q: : Is that all?

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



Secure Compilation of Outcalls

Q: : Is that all?

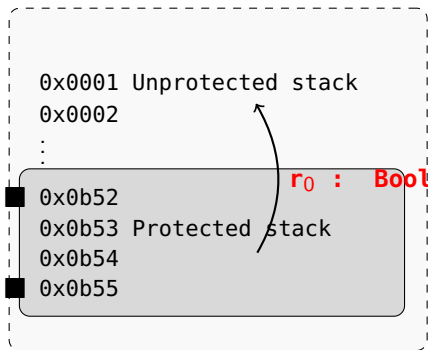
- protected stack
- returnback entry point
- reset flags and registers
- ground-typed values check



Secure Compilation of Outcalls

Q: : Is that all?

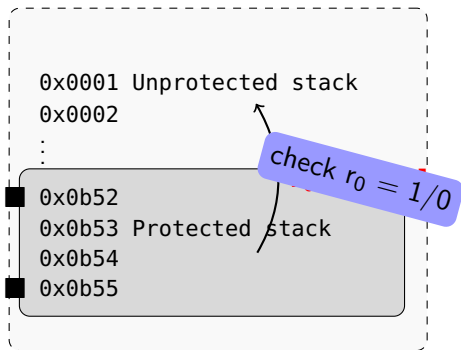
- protected stack
- returnback entry point
- reset flags and registers
- ground-typed values check



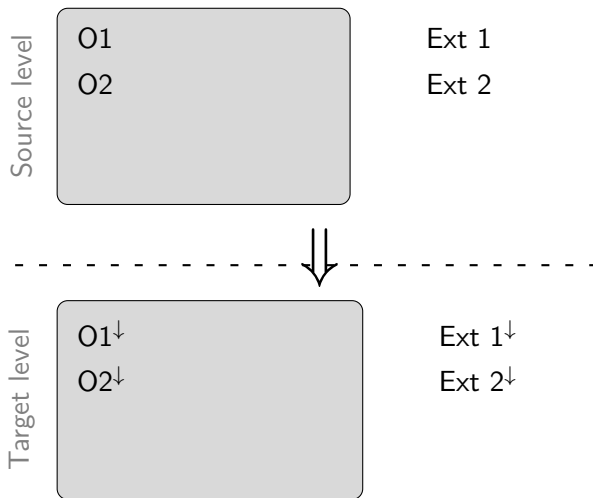
Secure Compilation of Outcalls

Q: : Is that all?

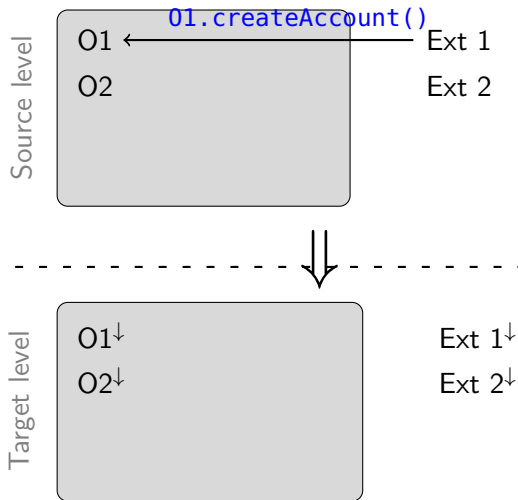
- protected stack
- returnback entry point
- reset flags and registers
- ground-typed values check



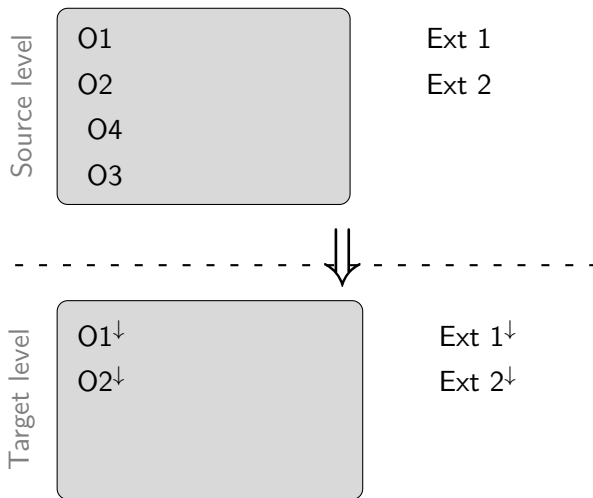
Dynamic Memory Allocation



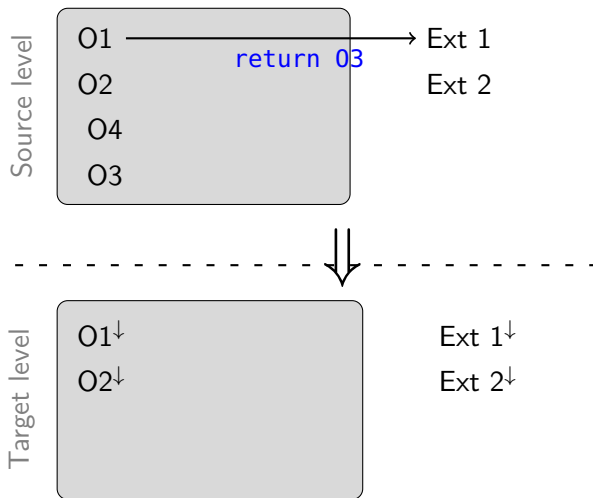
Dynamic Memory Allocation



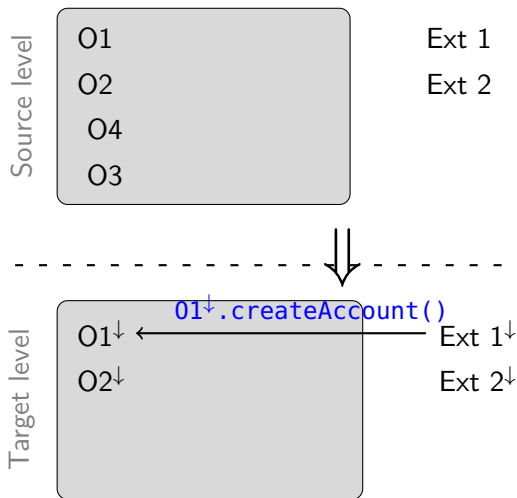
Dynamic Memory Allocation



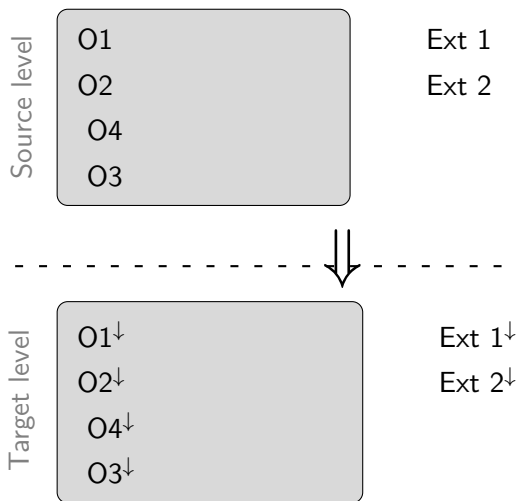
Dynamic Memory Allocation



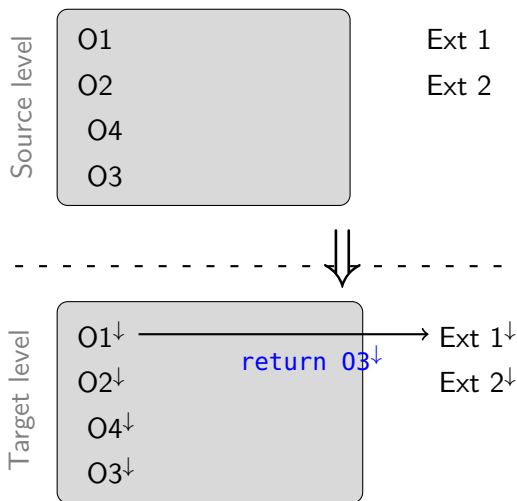
Dynamic Memory Allocation



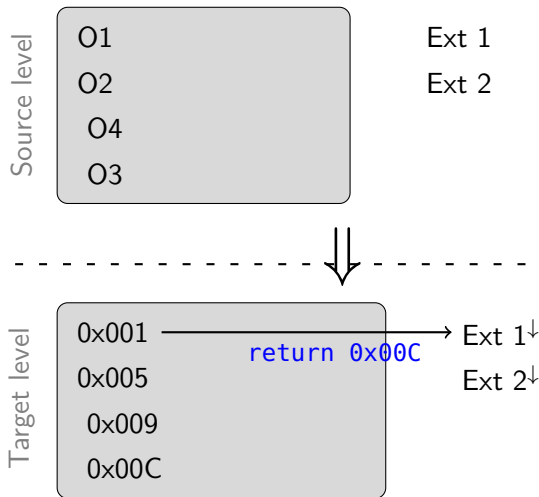
Dynamic Memory Allocation



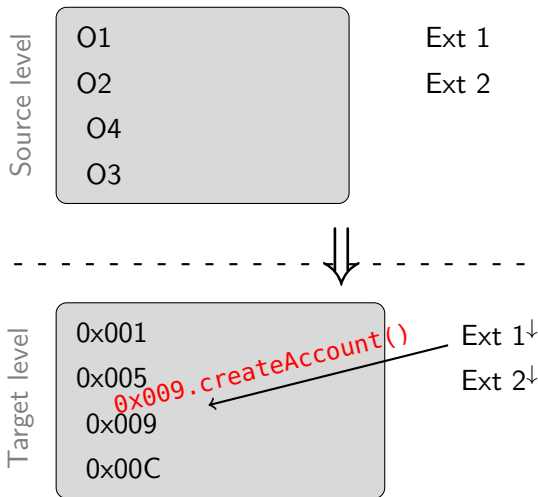
Dynamic Memory Allocation



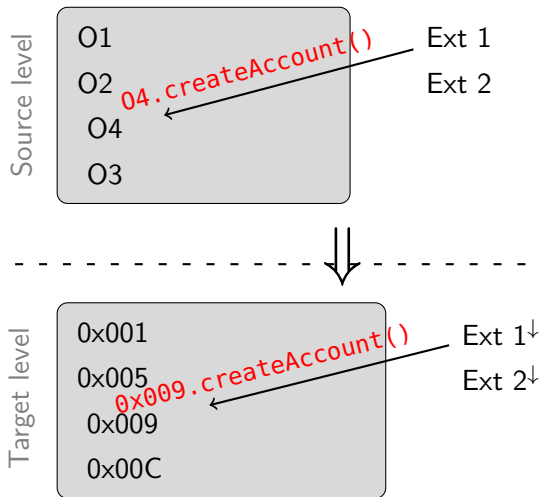
Dynamic Memory Allocation



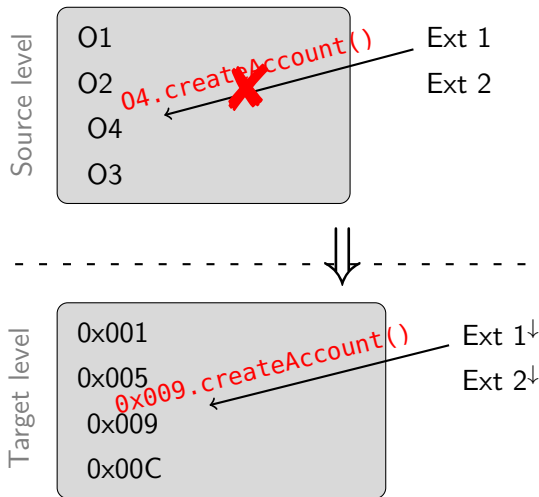
Dynamic Memory Allocation



Dynamic Memory Allocation

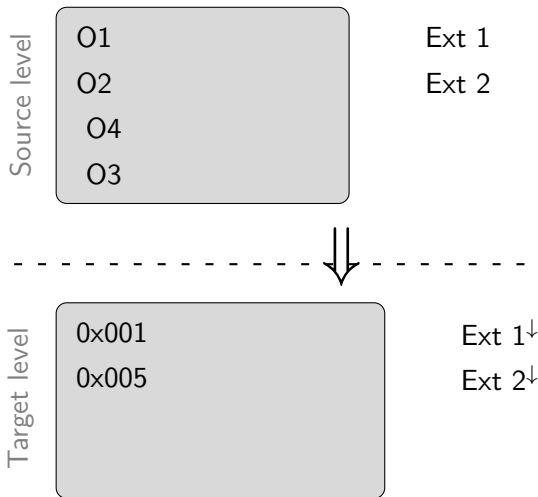


Dynamic Memory Allocation



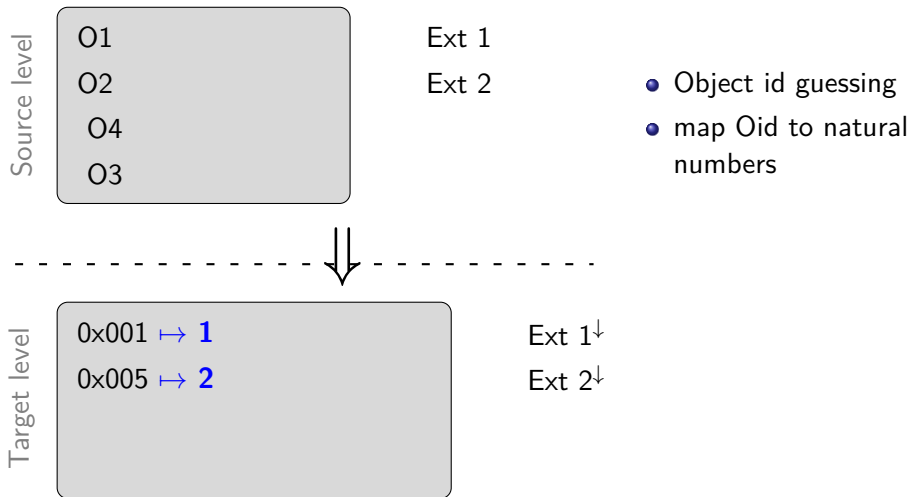
- Object id guessing

Dynamic Memory Allocation

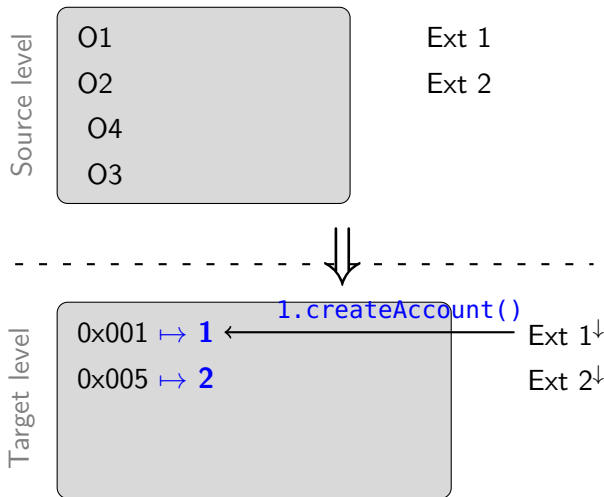


- Object id guessing
- map Oid to natural numbers

Dynamic Memory Allocation

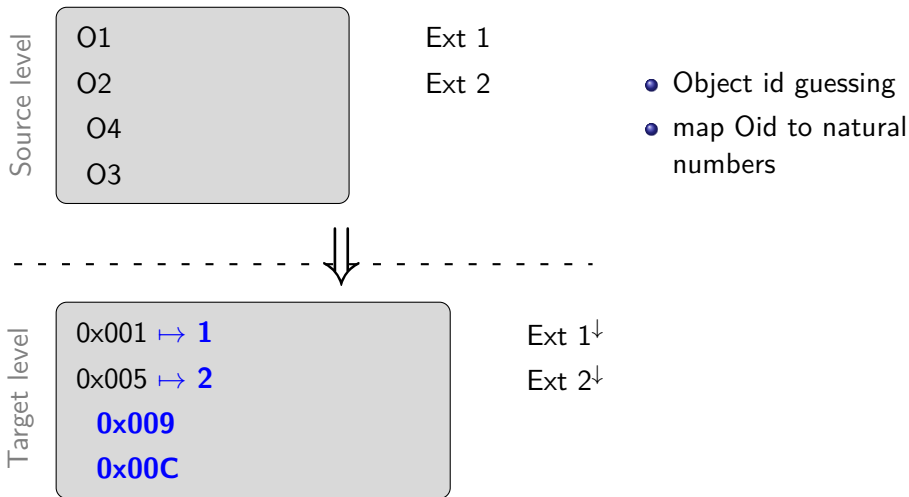


Dynamic Memory Allocation

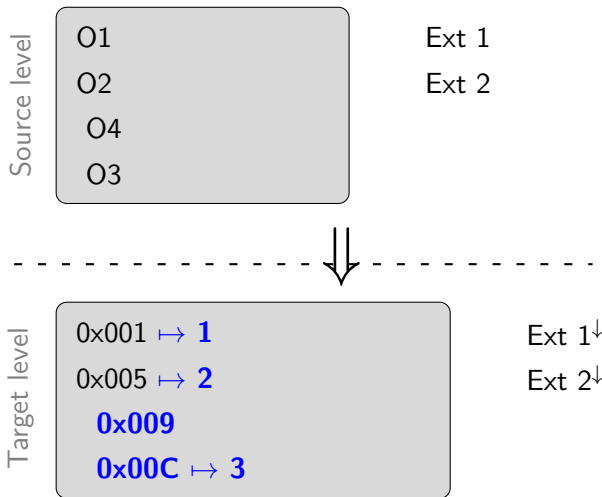


- Object id guessing
- map Oid to natural numbers

Dynamic Memory Allocation

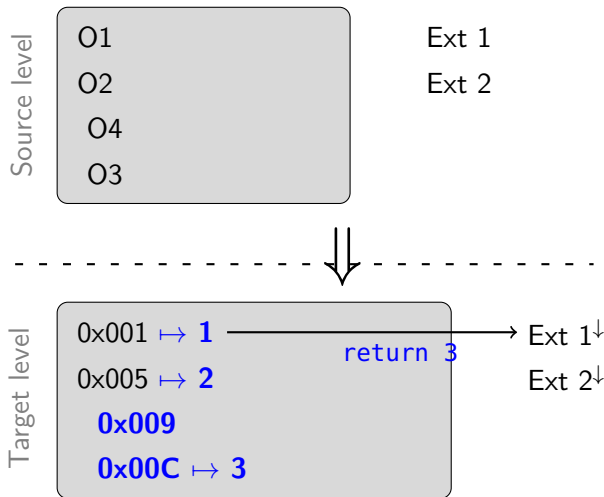


Dynamic Memory Allocation



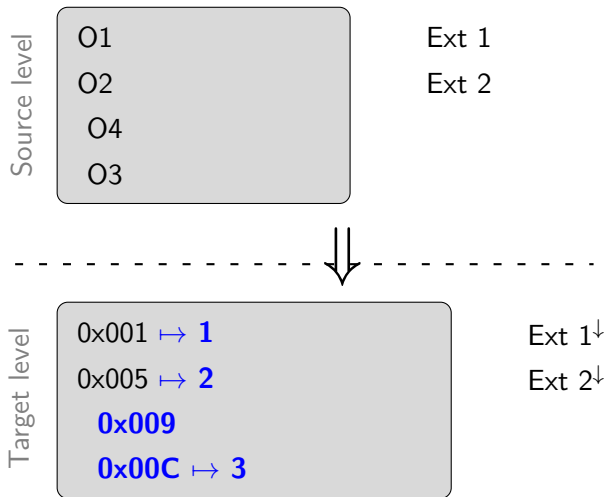
- Object id guessing
- map Oid to natural numbers
- add Oid to map

Dynamic Memory Allocation



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

Dynamic Memory Allocation



- 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

Ext 1

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



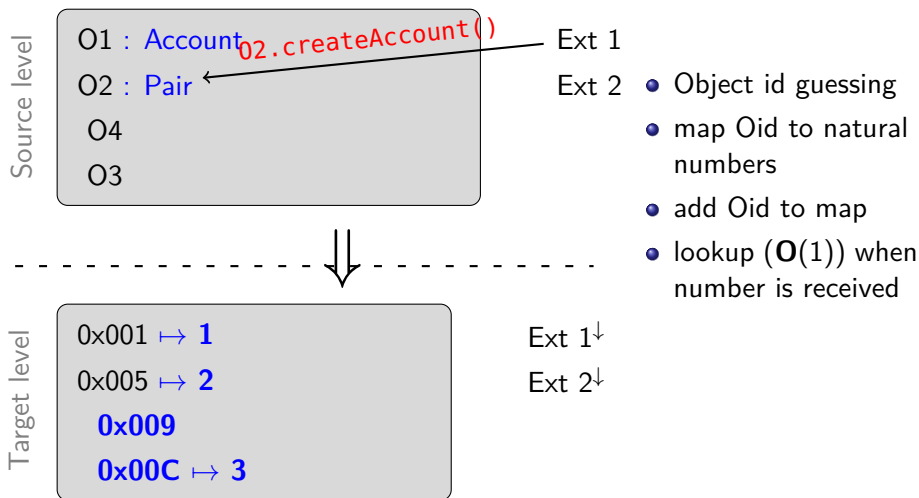
Target level

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

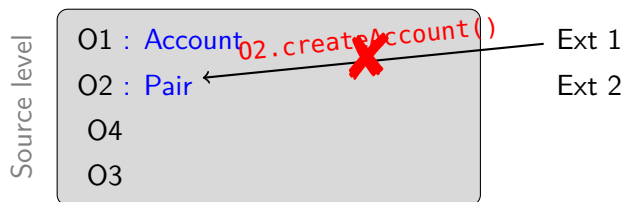
Ext 1 \downarrow

Ext 2 \downarrow

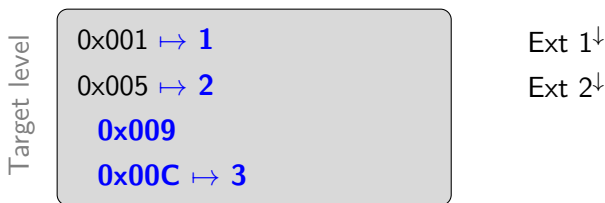
Dynamic Memory Allocation



Dynamic Memory Allocation



- 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

Ext 1

Ext 2

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



Target level

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

2.createAccount()

Ext 1 \downarrow

Ext 2 \downarrow

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

Ext 1 \downarrow

Ext 2 \downarrow

Dynamic Memory Allocation

Source level

O1 : Account
O2 : Pair
O4
O3



Target level

0x001 → 1
0x005 → 2
0x009
0x00C → 3

2.createAccount()

Ext 1

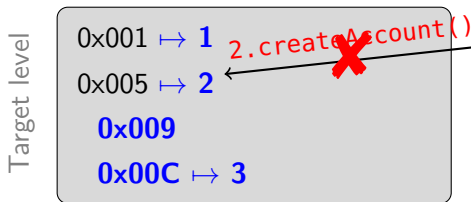
Ext 2

- 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

Ext 1↓

Ext 2↓

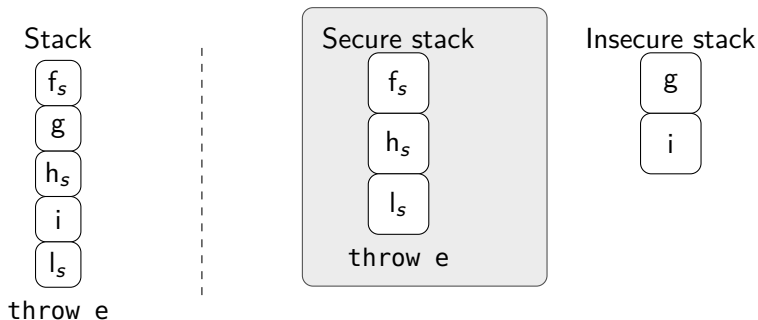
Dynamic Memory Allocation



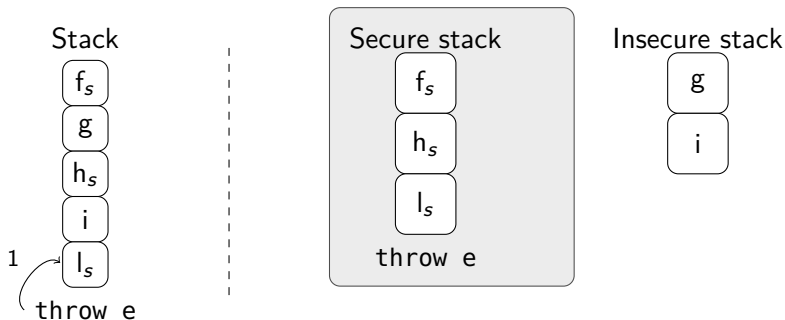
Ext 1

- 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

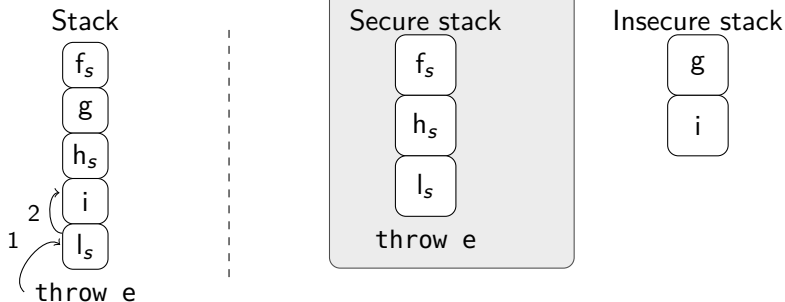
Exceptions



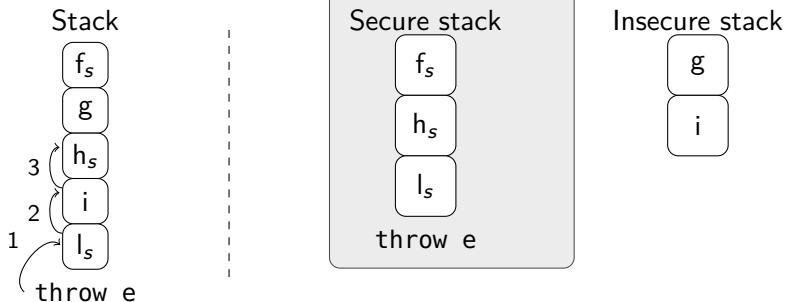
Exceptions



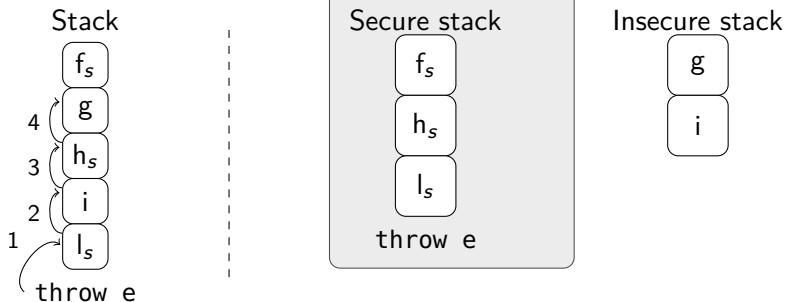
Exceptions



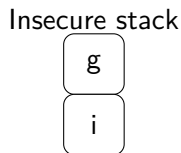
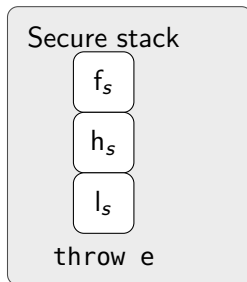
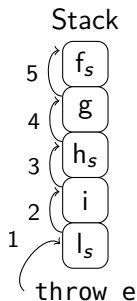
Exceptions



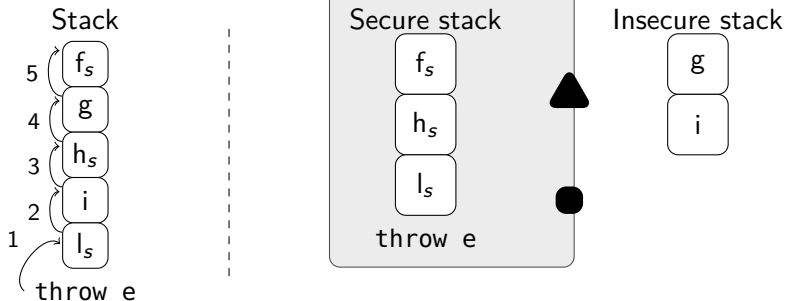
Exceptions



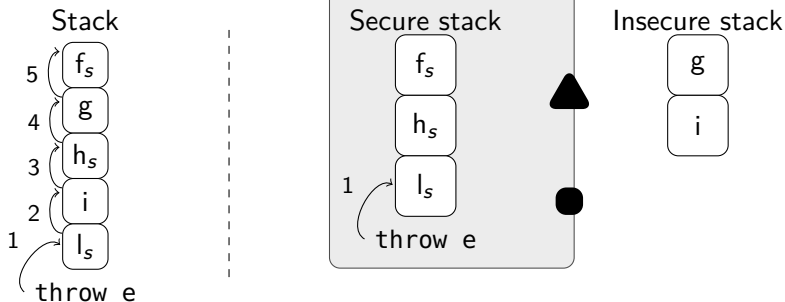
Exceptions



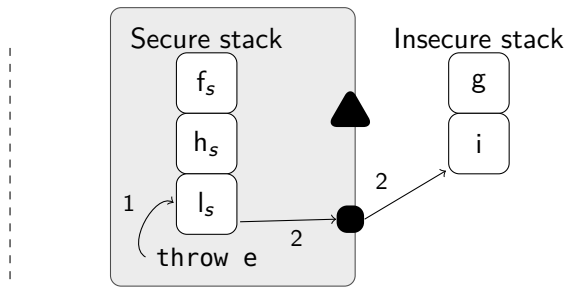
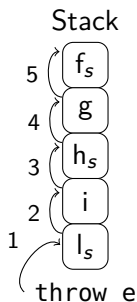
Exceptions



Exceptions

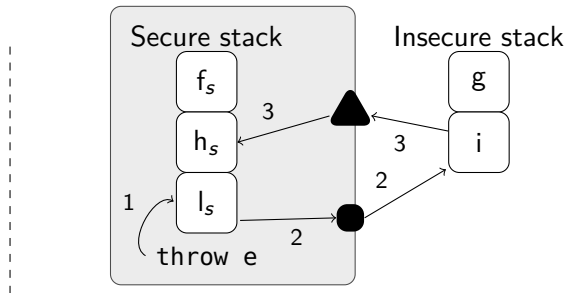
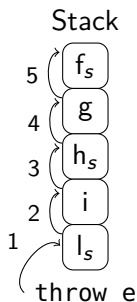


Exceptions



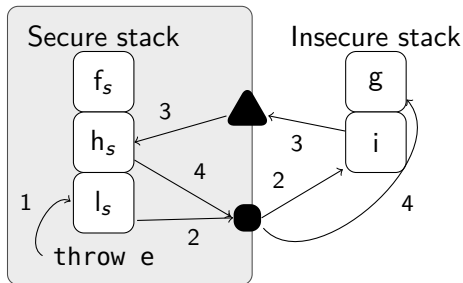
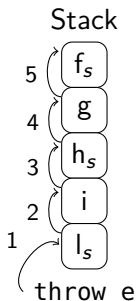
Record passed exceptions

Exceptions



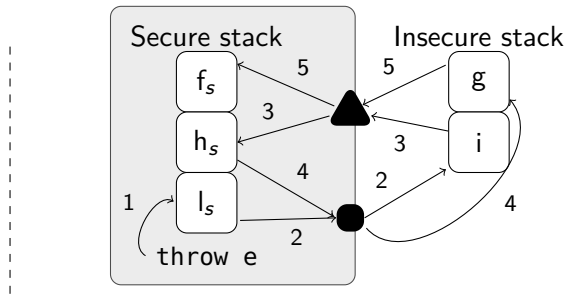
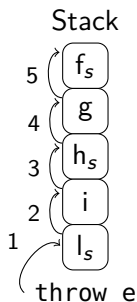
Record passed exceptions
Check that exception could be thrown

Exceptions



Record passed exceptions
Check that exception could be thrown

Exceptions



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

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

$$C_1 \approx^{J+E} C_2 \iff C_1^\downarrow \approx^{A+} C_2^\downarrow$$

Secure Compilation, Formally

$$C_1 \approx^{J+E} C_2 \iff C_1^\downarrow \approx^{A+} C_2^\downarrow$$

Contextual Equivalence

Contextual Equivalence

$$C_1 \simeq^S C_2 \triangleq \forall C. C[C_1] \Downarrow \iff C[C_2] \Downarrow$$

Contextual Equivalence

$$C_1 \simeq^S C_2 \triangleq \forall C. C[C_1] \Downarrow \iff C[C_2] \Downarrow$$

Contextual Equivalence

$$C_1 \simeq^S C_2 \triangleq \forall C. C[C_1] \Downarrow \iff C[C_2] \Downarrow$$

All contexts

Secure Compilation

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

Secure Compilation

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

$$(\forall C. \mathbb{C}[C_1]^\uparrow \iff \mathbb{C}[C_2]^\uparrow) \iff (\forall M. M[C_1^\downarrow]^\uparrow \iff M[C_2^\downarrow]^\uparrow)$$

Secure Compilation

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

VERY COMPLEX!

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

Secure Compilation

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

Secure Compilation

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

Secure Compilation

$$C_1 \simeq^{J+E} C_2 \quad \checkmark \quad C_1^\downarrow \simeq^{A+I} C_2^\downarrow$$

Secure Compilation

$$C_1 \simeq^{J+E} C_2 \Rightarrow C_1^\downarrow \simeq^{A+I} C_2^\downarrow$$

Secure Compilation

$$C_1 \simeq^{J+E} C_2 \Rightarrow$$

$$\begin{array}{c} C_1^\downarrow \simeq^{A+I} C_2^\downarrow \\ \Updownarrow \\ \text{Traces}(C_1^\downarrow) = \text{Traces}(C_2^\downarrow) \end{array}$$

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

$$C_1 \not\approx^{J+E} C_2 \quad \checkmark \quad \text{Traces}(C_1^\downarrow) \neq \text{Traces}(C_2^\downarrow)$$

Secure Compilation

$$C_1 \not\approx^{J+E} C_2 \quad \checkmark$$

$$\begin{array}{c} C_1^\downarrow \simeq^{A+I} C_2^\downarrow \\ \Updownarrow \\ \text{Traces}(C_1^\downarrow) = \text{Traces}(C_2^\downarrow) \end{array}$$

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?

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)

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

PMA Expressiveness

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?

PMA Expressiveness

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 ?

Trace Semantics for PMA, Semi-formally

- define states S for programs

Trace Semantics for PMA, Semi-formally

- define states S for programs
- define a semantics for PMA only: $\xrightarrow{i}: S \times S$

Trace Semantics for PMA, Semi-formally

- define states S for programs
- define a semantics for PMA only: $\xrightarrow{i}: S \times S$
- define labels (observables) α

Trace Semantics for PMA, Semi-formally

- define states S for programs
- define a semantics for PMA only: $\xrightarrow{i}: S \times S$
- define labels (observables) α : `call p \bar{r}`

Trace Semantics for PMA, Semi-formally

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

Trace Semantics for PMA, Semi-formally

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

Trace Semantics for PMA, Semi-formally

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

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

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? ✗
 - 1 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

Fully Abstract Trace Semantics

To ensure maximal precision, prove the trace semantics to be fully abstract

Fully Abstract Trace Semantics

To ensure maximal precision, prove the trace semantics to be fully abstract

i.e. there are no other things that we missed

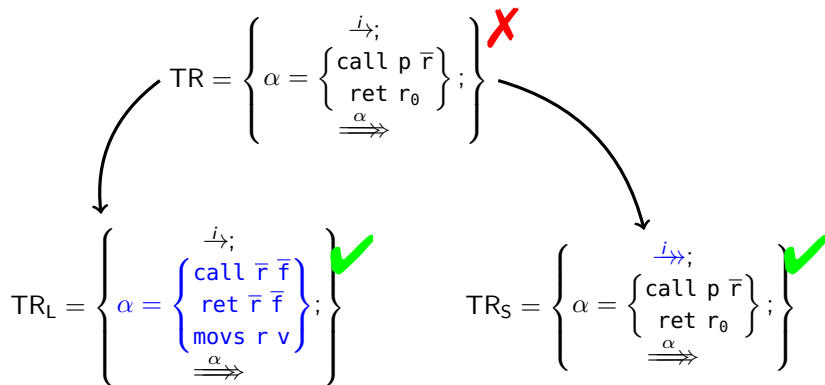
The Spectrum of Full Abstraction (by Curien)

$$\text{TR} = \left\{ \alpha = \left\{ \begin{array}{l} \xrightarrow{i}; \\ \text{call } p \ \bar{r} \\ \text{ret } r_0 \\ \xRightarrow{\alpha} \end{array} \right\}; \right\} \quad \text{X}$$

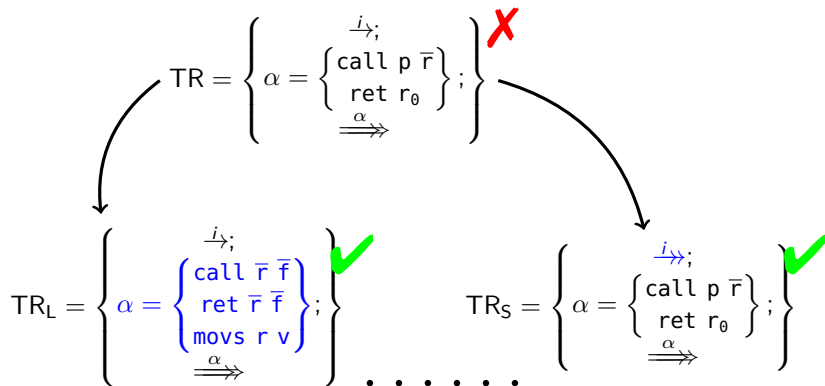
The Spectrum of Full Abstraction (by Curien)

$$\begin{array}{c}
 \text{TR} = \left\{ \alpha = \left\{ \begin{array}{c} \xrightarrow{i}; \\ \text{call } p \ \bar{r} \\ \text{ret } r_0 \\ \xRightarrow{\alpha} \end{array} \right\}; \right\} \quad \text{X} \\
 \downarrow \\
 \text{TR}_L = \left\{ \alpha = \left\{ \begin{array}{c} \xrightarrow{i}; \\ \text{call } \bar{r} \ \bar{f} \\ \text{ret } \bar{r} \ \bar{f} \\ \text{movs } r \ v \\ \xRightarrow{\alpha} \end{array} \right\}; \right\} \quad \checkmark
 \end{array}$$

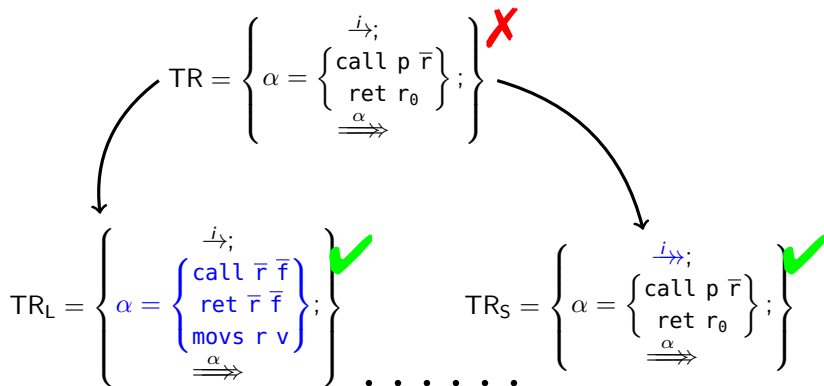
The Spectrum of Full Abstraction (by Curien)



The Spectrum of Full Abstraction (by Curien)



The Spectrum of Full Abstraction (by Curien)



$$TR_X(C_1) = TR_X(C_2) \iff C_1 \simeq C_2$$