Confined Types

Boris Bokowski,
FU Berlin / GMD-First

Jan Vitek
Purdue University

Thesis

Weak encapsulation fails to protect representations

Solutions…

- Islands, J. Hogg, *OOPSLA'91*
- Balloons, P. Almeida, *ECOOP'97*
- Sandwich types, Genius, Trapp and Zimmermann, *TiC'98*
- Flexible alias protection, Noble, Potter, Vitek, *ECOOP'98*
- IWAOOS'99 – *wwwcui.unige.ch/~ecoopws*

Presentation Overview

- **Motivating Example**
  Writing secure code in Java
- **Confined Types**
  Static checking of strong encapsulation
- **Discussion**
  Reusability, Alternatives, Patterns

Motivating Example

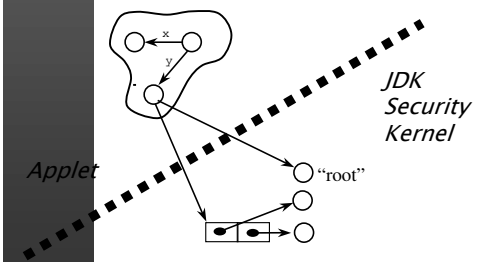
Writing Secure Code in Java

or

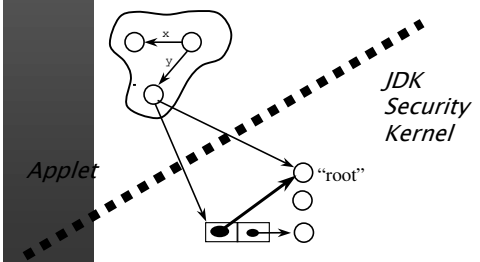
The HotJava 1.0 Signature Bug.

Princeton SIP team, 1997

Class Signing



Class Signing



The "Bug"

```
public class Identity  { … }

{
  …
  private Identity[] signers ;

  public Identity[] getSigners() {
     return signers;
  }
```

Ad hoc "Fix"

```
public class Identity  { … }

{
  …
  private Identity[] signers ;

  public Identity[] getSigners() {
     return signers.clone();
  }
```

Systematic Solution with Confined Types

```
class Identity implements ConfinedType
```

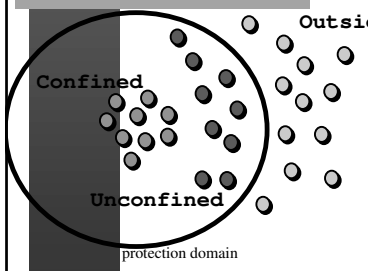
Identify reference transfers that break encapsulation boundaries.

Prevent transfer of references to confined objects.

Confined Types

Static enforcement of strong encapsulation

Strong encapsulation



Strong encapsulation enforces the constraint that *Confined* objects can not be used by *Outside* objects.

***Unconfined* objects play the role of interface between *Outside* and *Confined*.**

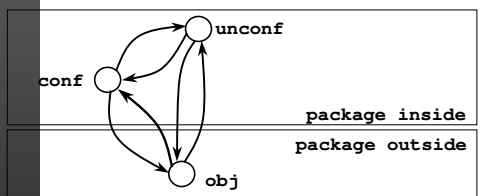
Confined Types in Java

- **Confinement declared by implementing the empty interface `ConfinedType`**
- **`Package`s act as protection domains**
- **Assumptions:**
  - **no run-time overhead for confinement**
    - ***compile-time checking***
  - **confined classes backwards compatible**
    - ***no language changes, Java 2 compatible***
    - ***outside classes not modified, reuse JDK as is***
  - **whole confining packages are checked**

Confined Types in Java

- ***Confinement constraints*** **prevent reference transfer of confined objects to outside packages**
- **Confinement enforced by static checking confinement constraints on Java source code of confining packages (only)**

Confined Types in Java



Preventing Transfers from the Outside

Outside classes can:

- create conf objects
- read conf fields of unconf objects
- invoke methods of unconf returning conf objects

Constraints on confining package:

- *Confined types are package-private*
- *Only private methods can return confined types*
- *Only private fields can have confined types*
- *Arrays of confined objects are confined*

Preventing Transfers from the Inside

Inside classes can:

- widen a conf type to an unconf type
- throw a conf exception

Constraints on confining package:

- *Widening from type A to type B is forbidden if A is conf and B is unconf*
- *Subtypes of Throwable may not be declared confined*

Hidden Widening

```
class C {
   public static C lastSeen;
   public void break() {
      lastSeen = this ;
      …
```

package outside
package inside

```
class Password extends C implements ConfinedType
```

Anonymous methods

Observation:

Hidden widening is a problem only if the inherited method leaks `this` ***to the outside***

Anonymous methods **constrain the implementation of outside classes that are parents of confined classes**

Concrete syntax:

```
/*:anon*/ public Object foo(){
```

Anonymous methods

Constraints:

- `this` *may not be used as (1) an argument, (2) on the RHS of an assignment, (3) as a return value, (4) as target for a non-anonymous method call*
- *Constructors called from an anonymous constructor must be anonymous*
- *Native methods are not anonymous*

Hidden widening (2)

Constraints on confining package:

- *Methods invoked on a confined object must either be defined in a confined class or be anonymous*
- *Subclasses of Thread may not be confined*

Anon and Backwards Compatibility

P
c
i
A
a
%

Discussion

Confinement and Reusability

- **Unconf class can be subclassed and used without restrictions**
- **Conf classes can be subclassed and used in their confining package**
- **Outside classes can be subclassed and made confined**

```
class PrivKey extends Key implements ConfinedType{}
```

but it is necessary to make their methods anon.

Confinement and Polymorphism

- **Generic data structures that depend on subtyping and widening are not a good match with confined types**
- **Instead use parametric polymorphism as in [Odersky and Wadler 97]**
- **Heterogeneous translation scheme in the implementation**

Guards and Facades

- Li Gong's *Guarded Objects* and the GoF *Façade Pattern* can be implemented by an unconf object providing an interface to one or more conf objects
- Confined types provide systematic enforcement

Other Notions of Protection Domain

- Protection domains need not be packages:
  - Sets of classes
  - Nested domains
  - Sets of objects (Flexible Alias Protection)

  but we can not use package protection mechanisms

Conclusion

- A programming discipline to obtain strong encapsulation in Java
- Confined types do not require language extensions or significant library modifications
- Static and modular checking

SOURCE CODE...

http://www.inf.fu-berlin.de/~bokowski/ConfinedTypes