

An Information Flow Verifier for Small Embedded Systems

Dorina Ghindici, Gilles Grimaud and Isabelle Simplot-Ryl

LIFL CNRS UMR 8022
University of Lille 1, France

WISTP'07, Heraklion, Crete, Greece, May 10th, 2007

Security issues for Java-enabled embedded systems

- Open, multi-applicative, embedded systems, dynamic class loading, overloading ...
- Enforcing security: sensitive data manipulated should be protected and accesible only to authorized users
- Current JVM security mechanisms **do not adress data propagation**

Example

```
package applet1;
class B {
    /* secret x */
    void m(int x, A a){
        ...
        this.p += a.foo(x);
    }
}
```

```
package applet2;
class Abis extends A {
    /* shared */
    public int leak;
    int foo(int x){
        ?
    }
}
```

Security issues for Java-enabled embedded systems

- Open, multi-applicative, embedded systems, dynamic class loading, overloading ...
- Enforcing security: sensitive data manipulated should be protected and accesible only to authorized users
- Current JVM security mechanisms **do not adress data propagation**

Example

```
package applet1;
class B {
    /* secret x */
    void m(int x, A a){
        ...
        this.p += a.foo(x);
    }
}
```

```
package applet2;
class Abis extends A {
    /* shared */
    public int leak;
    int foo(int x){
        this.leak = x;
    }
}
```

Data propagation = Information flow

Challenge: Ensuring confidentiality

Class B invokes an untrusted method $A.foo$ passing as argument a secret value.

Information flow

- Data annotated with security levels (*high/secret* and *low/public*)
- $p = s; \quad p \rightarrow s$
- Non-interference: secret, confidential value s should not be induced by the reader of public, low value p .

Data propagation = Information flow

Challenge: Ensuring confidentiality

Class B invokes an untrusted method $A.foo$ passing as argument a secret value.

Information flow

- Data annotated with security levels (*high/secret* and *low/public*)
- $p = s; \quad p \rightarrow s$
- Non-interference: secret, confidential value s should not be induced by the reader of public, low value p .

Example

Example

```
void m(int x, A a) {  
    while( x > 0 ) {  
        this.p += a.foo(x);  
        this.s += x--;  
    }  
}
```

Policy

s: secret/high
p: public/low

Methods signatures

- Contain potential flows generated by the execution of m

$$S_{foo} = \{R \xrightarrow{v} p_1\} \quad S_m = \{this^p \xrightarrow{v} x, this^s \xrightarrow{v} x\}$$

Example

Example

```
void m(int x, A a) {  
    while( x > 0 ) {  
        this.p += a.foo(x);  
        this.s += x--;  
    }  
}
```

Policy

s: secret/high
p: public/low

Methods signatures

- Contain potential flows generated by the execution of m

$$S_{foo} = \{R \xrightarrow{v} p_1\} \quad S_m = \{this^p \xrightarrow{v} x, this^s \xrightarrow{v} x\}$$

Approach

- Compositional, flow-sensitive, context-insensitive
- Intra and inter procedural analysis

Experimental results

	/method
Iter inter	3
Iter intra	1.5
CPU run	91.13ms
Avg memory	2.64Kb
Max memory	32.55Kb

Performance issues

Analysis impracticable
for a device having
limited resources

Approach

- Compositional, flow-sensitive, context-insensitive
- Intra and inter procedural analysis

Experimental results

	/method
Iter inter	3
Iter intra	1.5
CPU run	91.13ms
Avg memory	2.64Kb
Max memory	32.55Kb

Performance issues

Analysis impracticable
for a device having
limited resources

Approach

- Compositional, flow-sensitive, context-insensitive
- Intra and inter procedural analysis

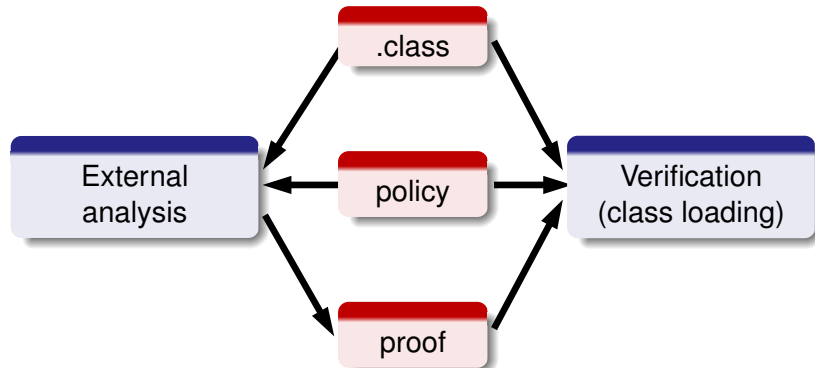
Experimental results

	/method
Iter inter	3
Iter intra	1.5
CPU run	91.13ms
Avg memory	2.64Kb
Max memory	32.55Kb

Performance issues

Analysis impracticable
for a device having
limited resources

Lightweight information flow verification



Signature verification

Example

```
0: void m(int x, A a) {  
1:   while( x > 0 ) {  
2:     this.p += a.foo(x);  
3:     this.s += x--;  
4:   }  
5: }
```

Proof elements

- JVM state for target
bytecodes: Q_1^p, Q_4^p
- Signature: S_m^p
- External signatures:
 S_{foo}
- Fields policy

Class validation

- Compatible JVM states: $Q_1^c \leq Q_1^p, Q_4^c \leq Q_4^p$
- Compatible signatures: $S_m^c \leq S_m^p$

Signature verification

Example

```

0: void m(int x, A a) {
1:   while( x > 0 ) {
2:     this.p += a.foo(x);
3:     this.s += x--;
4:   }
5: }
    
```

Proof elements

- JVM state for target
 bytecodes: Q_1^p, Q_4^p
- Signature: S_m^p
- External signatures:
 S_{foo}
- Fields policy

Class validation

- Compatible JVM states: $Q_1^c \leq Q_1^p, Q_4^c \leq Q_4^p$
- Compatible signatures: $S_m^c \leq S_m^p$

Signature management

Two dictionaries:

- **Temp** : temporary dictionary
- **Dico**: certified dictionary

Scenario

```
load class C
  external A.foo with  $S_{foo}^2$ 
load class D
  external A.foo with  $S_{foo}^1$ 
load class A
  A.foo with  $S_{foo}$ 
```

Class validation

$$S_{foo} \leq (S_{foo}^2 \sqcap S_{foo}^1)$$

- **Temp** : $S_{foo}^2 \sqcap S_{foo}^1$
- **Dico** : S_{foo}

Example

```
class C
```

```
A.foo:  $S_{foo}^2$ 
```

```
class D
```

```
A.foo:  $S_{foo}^1$ 
```

```
class A
```

```
A.foo:  $S_{foo}$ 
```

Signature management

Two dictionaries:

- **Temp** : temporary dictionary
- **Dico**: certified dictionary

Scenario

load class C

external A.foo with S_{foo}^2

load class D

external A.foo with S_{foo}^1

load class A

A.foo with S_{foo}

Class validation

$$S_{foo} \leq (S_{foo}^2 \sqcap S_{foo}^1)$$

- **Temp** : $S_{foo}^2 \sqcap S_{foo}^1$
- **Dico** : S_{foo}

Example

class C

A.foo: S_{foo}^2

class D

A.foo: S_{foo}^1

class A

A.foo: S_{foo}

Signature management

Two dictionaries:

- **Temp** : temporary dictionary
- **Dico**: certified dictionary

Scenario

```
load class C
  external A.foo with  $S_{foo}^2$ 
load class D
  external A.foo with  $S_{foo}^1$ 
load class A
  A.foo with  $S_{foo}$ 
```

Class validation

$$S_{foo} \leq (S_{foo}^2 \sqcap S_{foo}^1)$$

- **Temp** : $S_{foo}^2 \sqcap S_{foo}^1$
- **Dico** : S_{foo}

Example

class C

A.foo: S_{foo}^2

class D

A.foo: S_{foo}^1

class A

A.foo: S_{foo}

Signature management

Two dictionaries:

- **Temp** : temporary dictionary
- **Dico**: certified dictionary

Scenario

```
load class C
  external A.foo with  $S_{foo}^2$ 
load class D
  external A.foo with  $S_{foo}^1$ 
load class A
  A.foo with  $S_{foo}$ 
```

Class validation

$$S_{foo} \leq (S_{foo}^2 \sqcap S_{foo}^1)$$

- **Temp** : $S_{foo}^2 \sqcap S_{foo}^1$
- **Dico** : S_{foo}

Example

class C

A.foo: S_{foo}^2

class D

A.foo: S_{foo}^1

class A

A.foo: S_{foo}

Signature management

Two dictionaries:

- **Temp** : temporary dictionary
- **Dico** : certified dictionary

Scenario

```
load class C
  external A.foo with  $S_{foo}^2$ 
load class D
  external A.foo with  $S_{foo}^1$ 
load class A
  A.foo with  $S_{foo}$ 
```

Class validation

$$S_{foo} \leq (S_{foo}^2 \sqcap S_{foo}^1)$$

- **Temp** : $S_{foo}^2 \sqcap S_{foo}^1$
- **Dico** : S_{foo}

Example

class C

A.foo : S_{foo}^2

class D

A.foo : S_{foo}^1

class A

A.foo : S_{foo}

Implementation

- The verifier as a JVM plug-in
 - KVM: built-in
 - Class loader hierarchy: user-defined class loader

Example

```
public class SafeClassLoader extends ClassLoader {
    Dictionary dico;
    ...
}
```

- Extends the delegation model to the signatures lookup

Loading scenario

Scenario

Sc_2 loads class C
external $Sc_2.A.foo$ with S_{foo}^2
 Sc_1 loads class D
external $Sc_1.A.foo$ with S_{foo}^1
 Sc_1 loads class A
 $Sc_1.A.foo$ with S_{foo}

Class validation

$Sc_1.S_{foo} \leq Sc_1.S_{foo}^1$
 $Sc_1.S_{foo} \leq Sc_2.S_{foo}^2$

SCL: Sc_1

Classes loaded: D, A
Dico: S_{foo}
Temp Dico: S_{foo}^1

SCL: Sc_2

Classes loaded: C
Dico:
Temp Dico: S_{foo}^2

SCL: Sc_3

Classes loaded:
Dico:
Temp Dico:

Loading scenario

Scenario

Sc_2 loads class C
external $Sc_2.A.foo$ with S_{foo}^2
 Sc_1 loads class D
external $Sc_1.A.foo$ with S_{foo}^1
 Sc_1 loads class A
 $Sc_1.A.foo$ with S_{foo}

Class validation

$Sc_1.S_{foo} \leq Sc_1.S_{foo}^1$
 $Sc_1.S_{foo} \leq Sc_2.S_{foo}^2$

SCL: Sc_1

Classes loaded: **D, A**
Dico: S_{foo}
Temp Dico: S_{foo}^1

SCL: Sc_2

Classes loaded: **C**
Dico:
Temp Dico: S_{foo}^2

SCL: Sc_3

Classes loaded:
Dico:
Temp Dico:

Loading scenario

Scenario

Sc_2 loads class C
external $Sc_2.A.foo$ with S_{foo}^2
 Sc_1 loads class D
external $Sc_1.A.foo$ with S_{foo}^1
 Sc_1 loads class A
 $Sc_1.A.foo$ with S_{foo}

Class validation

$Sc_1.S_{foo} \leq Sc_1.S_{foo}^1$
 $Sc_1.S_{foo} \leq Sc_2.S_{foo}^2$

SCL: Sc_1

Classes loaded: **D, A**
Dico: S_{foo}
Temp Dico: S_{foo}^1

SCL: Sc_2

Classes loaded: **C**
Dico:
Temp Dico: S_{foo}^2

SCL: Sc_3

Classes loaded:
Dico:
Temp Dico:

Loading scenario

Scenario

Sc_2 loads class C
external $Sc_2.A.foo$ with S_{foo}^2
 Sc_1 loads class D
external $Sc_1.A.foo$ with S_{foo}^1
 Sc_1 loads class A
 $Sc_1.A.foo$ with S_{foo}

Class validation

$Sc_1.S_{foo} \leq Sc_1.S_{foo}^1$
 $Sc_1.S_{foo} \leq Sc_2.S_{foo}^2$

SCL: Sc_1

Classes loaded: **D, A**
Dico: S_{foo}
Temp Dico: S_{foo}^1

SCL: Sc_2

Classes loaded: **C**
Dico:
Temp Dico: S_{foo}^2

SCL: Sc_3

Classes loaded:
Dico:
Temp Dico:

Loading scenario

Scenario

Sc_2 loads class C
 external $Sc_2.A.foo$ with S_{foo}^2
 Sc_1 loads class D
 external $Sc_1.A.foo$ with S_{foo}^1
 Sc_1 loads class A
 $Sc_1.A.foo$ with S_{foo}

Class validation

$Sc_1.S_{foo} \leq Sc_1.S_{foo}^1$
 $Sc_1.S_{foo} \leq Sc_2.S_{foo}^2$

SCL: Sc_1

Classes loaded: **D, A**
 Dico: S_{foo}
 Temp Dico: S_{foo}^1

SCL: Sc_2

Classes loaded: **C**
 Dico:
 Temp Dico: S_{foo}^2

SCL: Sc_3

Classes loaded:
 Dico:
 Temp Dico:

Experimental Results

Experimental results

	/method	/method
Iter inter	3	1
Iter intra	1.5	1
CPU run	91.13ms	6.92ms
Avg memory	2.64Kb	0.43Kb
Max memory	32.55Kb	2.44Kb

Proof size

Initial .class size	18.56Kb
Annotated .class	28.58Kb
Dico	4.01%
Proof	39.73%

Summary and Future Work

Summary

- Lightweight information flow verification
- Linear time and constant memory
- Acceptable time penalty and memory consumption
- First verifier for small embedded systems

Future Work

- Implicit flow
- Cut the size of the elements embedded within the code by optimizing the encoding of proof and signatures

Summary and Future Work

Summary

- Lightweight information flow verification
- Linear time and constant memory
- Acceptable time penalty and memory consumption
- First verifier for small embedded systems

Future Work

- Implicit flow
- Cut the size of the elements embedded within the code by optimizing the encoding of proof and signatures