```
@ParameterSecurity({"high", "high"})
@ReturnSecurity("void")
@WriteEffect({"low"})
public void ifElseAssignLocalField(boolean conditionHigh, int thenHigh) {
    int result = SootSecurityLevel.lowId(42);
    if (conditionHigh) {
        result = thenHigh;
    }
    lowField = result;
```

security violation:
lowField ^L := result ^H

Implementing Security Type for Java

supervisors: Prof. Dr. P. Thiemann, Luminous Fennell
presenter: Thomas Vogel
date: 11/06/2013

proglang.informatik.uni-freiburg.de

motivation

pseudo-code:

```
lowValue^L = 42
highValue^H = true

if highValue
   print(lowValue)
```

content

- **team project objectives**
- **theory**
  - security
  - write effects
- **Soot framework**
- **analysis**
  - implementation
  - vulnerabilities / missing
- **conclusion**

TEAM PROJECT OBJECTIVES

team project objectives

- tool for Java developer
- (intra-procedural) data flow analysis
- checks for violations of the *Security Type*
- implementation builds upon a framework

THEORY

security level

- constants
  → weakest security level
- expressions
  → strongest security level of the operands
- locals, fields, methods
  → security level definable
- library fields
  → weakest security level
- library methods
  → strongest security level of the operands

pseudo-code:

```
// constant
42                  // --> ^L

// expression
42 + high^H         // --> ^H

// local
local^H = 42        // --> ^H

// field
field^L = 42        // --> ^L

// method
method()^H          // return
   ...              // --> ^H
```

assignment

- locals
  → no restrictions

- fields
  → security level of assigned value <= level of field

- array
  → security level of assigned value == weakest security level *(restriction)*

⚠ context:
security level of the context must be taken into account

pseudo-code:

```
// locals
val^L = high^H
// --> val^H

// fields
field^H = val^L
// --> field^H

// array
arr^H[0] = val^L
// arr^H[0]--> ^H


// context handling
if high^H
   // only “high” fields
   // can be assigned
   field^H = val^L
```

methods & objects

- methods
  - parameter
    → security level of argument <= level of parameter
  - return statement
    → security level of returned value <= specified return level

⚠ context:

security level of the context must be taken into account for return statement

- objects
  → security level of an instance trumps the level of a field or method

pseudo-code:

```
// method
meth(p1^L, p2^H)^H
   ...
   return p1^L

field^H = meth(v^L, v^L)
// --> ^H

// context
meth()^H
   ...
   if val1^H
      return val2^L

// object
obj^H = new A()
field^H = obj^H.field^L
// --> ^H
```

write effects

- write effects affect a specific security level
  - triggered by assignments or by method invocations
- method write effects
  - all effects which occur inside the method body
- class write effects
  - all effects which occur inside the *static initializer* method body

⚠ context:

the affected security level of a write effects has to be >= security level of the context

SOOT FRAMEWORK

Soot

- Sable Research Group
- framework for Java optimization
  - source-code and byte-code
- provides abstract classes for different kinds of analyses
- Intermediate representation *Jimple*
  - 3 address code
  - `GOTO` instead of `if else`, `for`, `while`, `do`, etc.
- http://www.sable.mcgill.ca/soot/

SECURITY TYPE ANALYSIS

analysis - implementation

- static intra-procedural forward dataflow analysis
- annotations for definition of security level & write effects
- `SecurityLevel`
  - id functions

annotations:

```
@WriteEffects({ … })
public class A {

  @ParameterSecurity({ … })
  @ReturnSecurity(…)
  @WriteEffects({ … })
  public int m(int i) {
    return i;
  }
}
```

implementation of the `SootSecurityLevel.java`:

```
package security;

public class SootSecurityLevel extends SecurityLevel {

  @Override
  public String[] getOrderedSecurityLevels() {
    return new String[] { "high", "low"};
  }

  @ReturnSecurity("high")
  public static <T> T highId(T object) {
    return object;
  }

  @ReturnSecurity("low")
  public static <T> T lowId(T object) {
    return object;
  }
}
```

Demo

analysis - missing

- variable security level
- extended security level hierarchy
- `Exception` handling
- extended arrays
- some statements, e.g. `switch case`
- inheritance

CONCLUSION

conclusion

- static check for security violations
- based on the Soot Framework
- still much to do ...
- https://github.com/peterthiemann/gradual-java

literature

[1]: J.P. Galeotti and A. Gorla, "*Introduction to Soot*", November 2012,
http://www.st.cs.uni-saarland.de/edu/automatedtestingverification12/slides/02-lab-introduction-to-soot.pdf