DD2460 Software Safety and Security: Part III
Exercises session 2: Type + Jif

Gurvan Le Guernic
adapted from Aslan Askarov

DD2460 (III, E2)
February 22st, 2012

1 Noninterference type systems challenge

Connect to `http://www.vinosv.dk/ifc/` and find the solution to all the challenges.

Advice: backup the (login, password) pairs.

1.1 Challenge 1

No type system.

1.2 Challenge 2

Expressions: $e := 0 \mid 1 \mid x$ (1 is True for conditions).

Type system:

$$\frac{x \in \{h1, h2, h3, h4, h5\}}{\Gamma \vdash x := e \ : \ \texttt{cmd}} \qquad \frac{\Gamma \vdash e \ : \ \texttt{L}}{\Gamma \vdash x := e \ : \ \texttt{cmd}} \qquad \Gamma \vdash \mathsf{skip} \ : \ \texttt{cmd}$$

$$\frac{\Gamma \vdash S_1 \ : \ \texttt{cmd} \qquad \Gamma \vdash S_2 \ : \ \texttt{cmd}}{\Gamma \vdash S_1; S_2 \ : \ \texttt{cmd}} \qquad \frac{\Gamma \vdash S_1 \ : \ \texttt{cmd} \qquad \Gamma \vdash S_2 \ : \ \texttt{cmd}}{\Gamma \vdash \textbf{if} \ (e) \ \{S_1\} \ \textbf{else} \ \{S_2\} \ : \ \texttt{cmd}}$$

$$\frac{\Gamma \vdash S \ : \ \texttt{cmd}}{\Gamma \vdash \textbf{while} \ (e) \ \{S\} \ : \ \texttt{cmd}}$$

1.3 Challenge 3

Expressions: $e := 0 \mid 1 \mid x$ (1 is True for conditions).

Type system:

$$\frac{x \in \{hi \mid i \in [1,5]\}}{\Gamma \vdash x := e \ : \ l} \qquad \frac{\Gamma \vdash e \ : \ \texttt{L}}{\Gamma \vdash x := e \ : \ \texttt{L}} \qquad \Gamma \vdash \mathsf{skip} \ : \ l \qquad \frac{\Gamma \vdash S_1 \ : \ l \qquad \Gamma \vdash S_2 \ : \ l}{\Gamma \vdash S_1; S_2 \ : \ l}$$

$$\frac{\Gamma \vdash e \ : \ \texttt{L} \qquad \Gamma \vdash S_1 \ : \ l \qquad \Gamma \vdash S_2 \ : \ l}{\Gamma \vdash \textbf{if} \ (e) \ \{S_1\} \ \textbf{else} \ \{S_2\} \ : \ l} \qquad \frac{\Gamma \vdash S_1 \ : \ \texttt{H} \qquad \Gamma \vdash S_2 \ : \ \texttt{H}}{\Gamma \vdash \textbf{if} \ (e) \ \{S_1\} \ \textbf{else} \ \{S_2\} \ : \ l}$$

$$\frac{\Gamma \vdash e \ : \ \texttt{L} \qquad \Gamma \vdash S \ : \ l}{\Gamma \vdash \textbf{while} \ (e) \ \{S\} \ : \ l} \qquad \frac{\Gamma \vdash S \ : \ \texttt{H}}{\Gamma \vdash \textbf{while} \ (e) \ \{S\} \ : \ l}$$

1.4 Challenge 4

Expressions: $e := 0 \mid 1 \mid x$ (1 is True for conditions).

Type system:

$$\frac{x \in \{hi \mid i \in [1,5]\}}{\Gamma \vdash x := e : l} \qquad \frac{\Gamma \vdash e : \mathtt{L}}{\Gamma \vdash x := e : \mathtt{L}} \qquad \Gamma \vdash \mathsf{skip} : l \qquad \frac{\Gamma \vdash S_1 : l \quad \Gamma \vdash S_2 : l}{\Gamma \vdash S_1; S_2 : l}$$

$$\frac{\Gamma \vdash e : \mathtt{L} \quad \Gamma \vdash S_1 : l \quad \Gamma \vdash S_2 : l}{\Gamma \vdash \textbf{if } (e) \{S_1\} \textbf{ else } \{S_2\} : l} \qquad \frac{\Gamma \vdash S_1 : \mathtt{H} \quad \Gamma \vdash S_2 : \mathtt{H}}{\Gamma \vdash \textbf{if } (e) \{S_1\} \textbf{ else } \{S_2\} : l}$$

$$\frac{\Gamma \vdash e : \mathtt{L} \quad \Gamma \vdash S : l}{\Gamma \vdash \textbf{while } (e) \{S\} : \mathtt{L}}$$

2 Jif

Before starting, setup your environment as follows:

1. go into the directory into which you will do the exercises: `cd ...`
2. create a symbolic link to: /opt/jif/jif-3.3.1/tests (`ln -s /opt/jif/jif-3.3.1/tests`)
3. compile the following files:
   - Alice.jif: `jifc -nonrobust tests/jif/principals/Alice.jif`
   - root.jif: `jifc -nonrobust tests/jif/principals/root.jif`

To run the Jif compiler: `jifc -nonrobust [-classpath ".;$HOME/.jif"] file.jif`

Files generated by the compiler are created into: `$HOME/.jif`

2.1 Program number one

Write down the following code to the file T.jif

```
public class T {
}
```

Compile it with: `jifc -nonrobust T.jif`

2.2 Variable declaration

Add to the class `T` an integer variable `secret` of security level {Alice−>} and an integer variable `pbl` of security level {}.

Compile the program with: `jifc -nonrobust T.jif` or `jifc -nonrobust -classpath ".;$HOME/.jif" T.jif`

Note that if you don't specify a label of a declared class field it is assumed to be {} by default. Try to remove the label annotation for variable `pbl` and compile the program again.

2.3 Method declaration

Add an empty void method `foo()` to the class `T` as follows.

```
public void foo() {
}
```

Compile the program as previously.

2.4 Methods without side-effects

In the method `foo()` declare a local variable `x` of type `int` and initialize it with the value of `this.pbl`.

```
int x = this.pbl;
```

Compile the program.

2.5 Direct flows

The simplest form of an information flow is an assignment statement. An assignment where security level of the left hand side is not higher or equal than the security level of a right hand side is leakage. An example of such a statement is:

```
int {} y = this.secret;
```

Add this statement to the function `foo()` and compile the program. You should expect an error message from compiler similar to the following:

```
T.jif:7: Label of local variable initializer not less restrictive than the
    label for local variable y
int {} y = this.secret;
           ^---------^

1 error.
```

One can get a more detailed error description with `-explain` compiler flag.

```
>jifc -nonrobust -explain T.jif
T.jif:7: Unsatisfiable constraint:
     init.nv <= label of local variable y
      {Alice->; _<-_; this; caller_pc} <= {caller_pc; _->_; _<-_}
     in environment
     [{this} <= {caller_pc}]

     Label Descriptions
     ------------------
     - init.nv = label of successful evaluation of initializing expression
     - init.nv = {Alice->; _<-_; this; caller_pc}
     - label of local variable y = {caller_pc; _->_; _<-_}
     - this = label of the special variable "this"
     - caller_pc = The pc at the call site of this method (bounded above by
    {*->})

     More information is revealed by the successful evaluation of the
    intializing expression than is allowed to flow to the local variable y.
int {} y = this.secret;
           ^---------^

1 error.
```

Label {`this`} is a label of the current class instance. This error message can be shortly interpreted as follows: the last line of the error message indicates the statement that causes the problem. Note that the assignment statement succeeds only if the evaluation of the right hand side succeeds. The right hand side in this statement is this.secret. In order for the class field secret to be evaluated successfully, the current instance of the class should be non-null which reveals "{`this`}" amount of information about

this. In addition, variable secret is labeled as {Alice->}. The join of these two labels {Alice->; this} constitutes the label of successful evaluation of initializing expression. It is denoted as init.nv. The constraint which Jif fails to satisfy while type checking the program is:

```
init.nv <= label of local variable y
 {Alice->; _<-_; this; caller_pc} <= {caller_pc; _->_; _<-_}
```

This points out to the insecure flow. To fix the flow one can label y as {Alice-> ; this}.

Note that if you declare y as

```
int y = this.secret
```

and compile the program no errors will be reported - here Jif automatically infers correct labels for us. Although this is a useful feature, be aware that it may sometimes confuse you.

2.6 Indirect flows

In the function foo() add a code that has an indirect flow. An example of an implicit flow from variable secret towards some public(low) variable z is

```
int {} z = 0;
if (secret > 0) z = 1;
```

The flow in this code occurs when assignment to publicly labeled z happens in the context that depends on the high variable secret. Compile the program to see the error message. Even though assignment statement z=1; does not contain an insecure flow by itself, the context in which it occurs depends on the high condition secret. Therefore, the pc-label in the branch of the if statement is incremented to {Alice->}. Hence the implicit flow from {Alice->} to {}.

Correct the label of z.

2.7 Methods with side-effects

Add an assignment statement to the method foo() that modifies the class field secret.

```
this.secret = 1;
```

How much information does this side-effect reveal?

Compile the program. Because assignment to a class field is a side-effect, it needs to be bounded by the begin-label of the method. You should expect an error message similar to the following.

```
T.jif:10: Label of right hand side not less restrictive than the label for
    field secret
this.secret = 1;
^---------^

T.jif:10: Effect of assignment to field secret is not bounded below by the PC
    bound.
this.secret = 1;
^---------^
```

More details:

```
T.jif:10: Unsatisfiable constraint:
     rhs.nv <= label of field secret
```

```
      {caller_pc; this} <= {Alice->}
     in environment
     [{this} <= {caller_pc}]

     Label Descriptions
     ------------------
     - rhs.nv = label of successful evaluation of right hand of assignment
     - rhs.nv = {caller_pc; this}
     - label of field secret = {Alice->}
     - caller_pc = The pc at the call site of this method (bounded above by
    {*->})
     - this = label of the special variable "this"

     More information is revealed by the successful evaluation of the right
    hand side of the assignment than is allowed to flow to the field secret.
this.secret = 1;
^---------^

T.jif:10: Unsatisfiable constraint:
     Li <= label of field secret
      {*->} <= {Alice->}
     in environment
     [{this} <= {caller_pc}]

     Label Descriptions
     ------------------
     - Li = Lower bound for side-effects
     - Li = {*->}
     - label of field secret = {Alice->}
     - this = label of the special variable "this"
     - caller_pc = The pc at the call site of this method (bounded above by
    {*->})

     Assignment to the field secret is a side effect which reveals more
    information than this method is allowed to; the side effects of this
    method must be bounded below by the method's PC bound, Li.
this.secret = 1;
^---------^
```

This error message in its turn can be interpreted as follows: the last line of the message indicates the statement that causes the error. Lower bounds for side effects denoted as Li is essentially the begin-label of this method. Since no begin-label specified for this method, the method is assumed to be side-effect free. Hence, any side effect in this method is rejected by the compiler. In particular, the statement `this.secret = 1` which has level `{Alice->}` is rejected.

Adjust the signature of the method with the begin-label of this side-effect and recompile. You shouldn't get any errors. Next, add another assignment statement

```
this.pbl = 0;
```

This statement modifies another class field but its level is lower than the level of the previous assignment. Thus, the program should be rejected by the compiler. Check it by compiling the program. Finally, relax the begin-label of the method foo so that the program is accepted by Jif.

2.8 Array declaration

In the method foo() declare two public arrays of public integers.

```
int {}[]{} larr1 = ...;
int {}[]{} larr2 = ...;
```

Initialize those arrays with 10 elements such that the i^{th} element of the array is equal to i+1. For `larr1`, do this when declaring it; for `larr2`, do this in a loop.

Setting array elements may throw ArrayIndexOutOfBoundsException. Unlike Java, Jif requires runtime exceptions to be caught. In this exercise there are at least two approaches to deal with this exception.

1. Declare it in the method header;
2. Catch and ignore it.

Try both of them and compile your programs. Note that, when you declare that this method throws an exception you would need to specify the level of that exception in the end-label of the method. Why do you think you need that?

Next, declare an array which is confidential with respect to Alice:

```
int {Alice->}[]{Alice->} harr;
```

Initialize this array the same way you have initialized `larr2`; handle the exception in the both ways described above.

2.9 Declaring a method with arguments

Create a new class `LArray`. In this class, declare a string array larr;

```
public class LArray {
  String {Alice->}[]{Alice->} larr;
}
```

Write a method `getAt()` that accepts an integer argument i and returns the element of the array larr at the i^{th} position. Use the following header for the declaration of the method getAt

```
public String{Alice->;i} getAt(int i):{...}
  throws ArrayIndexOutOfBoundsException, NullPointerException { ... }
```

The two exception classes that you have to handle in this method are ArrayIndexOutOfBoundsException (IOB) and NullPointerException (NPE). The reason for IOB is the same as before. NPE may be thrown if the variable larr is not initialized or is null. Declare these exceptions in the method header and set the end-label appropriately.

As for NPE, one can use NPE analysis and return null if the larr is null in the beginning of the method. The method header then becomes:

```
public String{Alice->;i} getAt(int i):{...}
  throws ArrayIndexOutOfBoundsException { ... }
```

Note that the NPE analysis works for local variables only. Therefore, in order to use it we need to declare a local variable and initialize it with the class field.

```
String{Alice->}[]{Alice->;this} larr = this.larr;
if (larr == null) return null;
...
```

Complete the method getAt and compile your program. Try with the declaration of the local variable larr.

2.10 Writing a setter method

Setter methods usually have both argument labels and side-effects. Write a method `setAt()` that accepts two arguments: `i` of type `int{Alice->}` and `s` of type `String{Alice->}`. This method assigns `s` to the i^{th} element of the `larr`. Start with the following header and modify it as needed:

```
public void setAt{Alice->}(int{Alice->} i, String{Alice->} s)
  throws ArrayIndexOutOfBoundsException, ArrayStoreException, NullPointerException {
  ...
}
```

Complete the method `setAt` and compile the program.

2.11 Adding constructor

Write a constructor for the class LArray that accepts an integer argument `n` with the label {`Alice->`}. In the constructor add a statement to initialize the array larr:

```
this.larr = new String[n];
```

Before *automatically* dealing with compilation errors, *think* about the purpose of the constructor.

2.12 Using the class LArray in another class

Declare an instance of `LArray` in the class `T` and initialize it. Write a boolean function `testLArray()` that uses functions `setAt` and `getAt`. In this function create an instance of `LArray` of size 10, set the values of every i^{th} element to the string representation of (i+1) using `setAt()` (use `Integer.toString(i+1)` to convert `int` to `String`). Use `getAt()` to check whether the returned values are correct. The function `testLArray` should return true if all the elements match their set values and false otherwise.

2.13 Class parametrization

In the current version of `LArray`, the `larr` variable is labeled with Alice's principal. One often wants the data structure to be more flexible. Class parametrization is useful in this case. Parametrize the class `LArray` over a label `L`. This parameter should correspond to the label of array's element and size. More precisely:

1. In the class declaration of `LArray`, add label parameter `L`.
2. Use `L` and/or {`L`} in the body of `LArray` instead of the label {`Alice->`}.
3. Use parametrized version of class `LArray` in the class `T`, while still ensuring security level {`Alice->`} for the `LArray` manipulated in `T`.
4. Rewrite `testLArray` so that it works with the parametrized class `LArray`.

2.14 Declassification and outputs

Jif controls outputs, which may reveal secret information. In order to retrieve an output stream on which you can write public information insert the code of the following method into your class `T`.

```
import java.io.PrintStream;
import jif.runtime.Runtime;

...
```

```
public PrintStream[{}]{_<-_} setupOutputStream() {
  PrintStream[{}] out = null;
  try {
    Runtime runtime = Runtime.getRuntime();
    out = runtime==null?null:runtime.stdout(new label {});
  }
  catch (SecurityException e) {}

  return out;
}
```

To use this stream, use the following lines where needed:

```
PrintStream[{}] out = ....setupOutputStream();
if (out != null) out.println("Testing␣the␣LArray␣class:");
```

Write a main method into your class `T` in order to test `LArray`. This main method should output a first message stating that your are testing `LArray`, create a `T` instance, call `testLArray()` on it, and finally output the result of the call to `testLArray()`. For the main method use the following header:

```
public static void main{}(String[] args) { ... }
```

In order to output the result of `testLArray()`, you will see that you need to declassify its label from {`Alice->`} to {} (the upper bound on what can be output on the stream). In order to declassify an expression `e` from label `L1` to label `L2`, use the following expression:

```
declassify(e, L1 to L2)
```

As your are removing Alice's policy when declassifying, the method (and by transitivity the class) needs Alice's authority. To provide it, add `authority (Alice)` to `T` and `main` as follows:

```
public class T authority (Alice) {
  ...

  public static void main{}(String[] args) where authority(Alice) { ... }
}
```

Once your program compiles, run it with the command: `jif [-classpath ".;$HOME/.jif"] T`

2.15 Password checking

The last exercise tests your `LArray` class. Download the Password class from the course outline. It is a small password checker utility that should compile with the `LArray` class that you have written.