

Assignment 3

Due: November 29, 11:59PM EST

Homework 3 is about using Jif to control information flow in Java programs.

If your group is changing, make sure you e-mail Michelle with the new information as soon as possible.

Submission will be via the Blackboard dropbox and must be received by **11:59 PM EST** on **November 29, 2010**. You may resubmit as many times as you like before the deadline; only the last submission will be graded. (**NOTE: Make sure you use “send” and not “upload” with the blackboard dropbox, or your homework will not be submitted properly.**) Please read and follow the instructions on the following pages carefully to ensure you receive maximum credit for your submission.

1 Resources

- Jif Reference manual (<http://www.cs.cornell.edu/jif/doc/jif-3.3.0/manual.html>)
- Jif FAQ (in the VM at `/home/user/hw3/resources/JifFaq.htm`)

2 The assignment

This assignment is in three parts. All parts are due at the same time.

2.1 Part 1 (20%)

This part only uses Java and not Jif.

In part one you are given code for three Java classes: **ExamRoom**, **Question**, and **Exam**, and for the Java interface **IStudent**. This code is found in `/home/user/part1/`. These classes and interface can be described as follows:

- class **ExamRoom** – This file is the starting point of the application. Function `main()` lives here.
- class **Question** – A data structure representing a question. This class has three fields: the text of the question, the array of options, and the answer.
- class **Exam** – This class has the method `runExam()` that runs the exam, and the method `questionPool()` that returns an array of questions.
- interface **IStudent** – This is an interface for a Student class. It has three methods that class Student has to implement.
 - `getAnswer()` – This method accepts the text of the question and an array of answers.
 - `passResult()` – This method accepts an integer which is the total number of points that student has collected.
 - `tellResult()` – In this method, a student prints out the total number that he gets from Examiner via `passResult()`.

The task in the this part is to write a malicious class **Student** (implementing **IStudent**) that leaks Alice’s answers to Bob, allowing students to cheat during the exam. Alice may follow some simple strategy when

choosing the right answer. An example of such a strategy might be just a random guess. In order to compile against the provided class **ExamRoom**, make sure your constructor in the **Student** class accepts a string argument that is a name of the student.

You need to demonstrate the attack by providing the output (as below) that shows that Bob always chooses the same answer as Alice. You must also submit your code. There should be no direct communication between Bob and Alice, or use of static class fields — the attack must exploit a weakness in the Exam class.

```
> java ExamRoom
Starting exam
Alice got question: What cafe in Goteborg offers Kope Luwak coffee?
Options: (0) Blue Mountain Cafe (1) Mauritz Kaffe
::Alice replies 1
Bob got question: What cafe in Goteborg offers Kope Luwak coffee?
Options: (0) Hello, Bob. I think the answer is 1. -Alice (1) Mauritz Kaffe
::Bob replies 1
Alice got question: What's the price of a Kope Luwak espresso?
Options: (0) 100SEK (1) 60SEK
::Alice replies 0
Bob got question: What's the price of a Kope Luwak espresso?
Options: (0) Hello, Bob. I think the answer is 0. -Alice (1) 60SEK
::Bob replies 0
exam finished
Student Alice got 1 points.
Student Bob got 1 points.
done.
```

Explain how Bob obtains the answer from Alice. You must hand in your Student.java code from this part of the lab, and your writeup should explain how the attack works.

This section will be graded by script – we will copy your version of Student.java into a directory with my versions of Exam, ExamRoom, Question and IStudent. Then we will run `java ExamRoom` and match your output with the expected output. Make sure your text matches the example exactly (with the possible exception of which answers Alice chooses) if you want full credit. One quarter of the points for this section will be for a brief but clearly writeup that explains how and why this attack works.

2.2 Using Jif

Once again we are giving you a VM with software and files that you need pre-installed.

Unzip the HW3_VM.tgz file. Using VirtualBox, select File→Import Appliance. Navigate to 18732-HW3.ovf and select it to import. Take all the default options.

The files for part 2 are pre-installed in `/home/user/hw3/resources`, and Jif is installed and accessible via `$JIF`. The profile should be set so that everything you need for Jif is in the path.

There are sample Jif files in `$JIF/tests/` and `$JIF/examples/`. (The files in `$JIF/tests` do not all compile, as they are designed to demonstrate both working Jif code and common errors.) Let's go through the sample `$JIF/tests/Hello.jif`. To test Jif code, you first compile it and then run it, using the following commands:

```
> cd $JIF/tests
> jifc -e -nonrobust -classpath . Hello.jif
> jif -classpath . Hello
Hello, world!
>
```

`jifc` is the compiler. The `-e` flag prints verbose explanations of compiler errors (which you will want). The `-nonrobust` flag makes enforcement of information flow somewhat less strict (which you will also want). The `classpath` flag helps Jif find the pre-existing principals used in the security labels.

2.3 Part 2 (20%)

Part 2 involves going through a set of Jif warmup exercises. For these exercises, you don't need to run any code – just compile different examples of basic Jif mechanics and practice troubleshooting Jif compiler errors.

The exercises are found in the VM at `/home/user/hw3/resources/JifExercises.htm`. Follow the instructions to work through the set of 13 Jif examples.

For this portion of the homework, all you need to turn in is a section of your writeup, no code. In that section of the writeup, include the text of the code you added for each exercise, as well as a short explanation of what the code does and whether or not it compiled successfully. If it didn't compile, include a short description of the error you saw and what the problem was. The goal in your writeup is to walk us through your work on the exercises and show us what you learned by doing them.

2.4 Part 3 (60%)

In part 3 you need to use Jif to re-implement the exam scenario, using information flow control to prevent the cheating that was possible in part 1.

- Write classes `Exam`, `Question`, and `Student`, and interface `IStudent`, in Jif. These classes can remain mostly the same as the Java versions, with modifications as necessary to provide proper flow control.
- Parameterize class `Student` and interface `IStudent` over student principal `S`.
- Use provided principals `Alice`, `Bob`, and `Examiner` for label annotations.
- Use provided skeleton `ExamRoom.jif` to get started.
- As previously, `IStudent` should contain three functions.
 - The method `getAnswer()` accepts a text of the question and array of answers. Question text and options should be labeled as `{Examiner: S}`, where `S` is principal parameter corresponding to a student. The return label of the `getAnswer()` method should be labeled so that it depends on the question and at the same time specifies that student owns the part of the answer and allows only examiner to read it.
 - The method `passResult()` should be called by the examiner (in `Exam`). The label of the argument of this function should correspond to the security policy that student owns a data and only examiner is allowed to read a data.
 - The method `tellResult()` should declassify the result and print it to the screen.

2.4.1 Running your part 3 code

To run code for part 3, unzip the part 3 tarball into a new directory (we'll put it at `/home/user/hw3` for this example). You should have the principal files `Alice.jif`, `Bob.jif`, and `Examiner.jif` installed there, as well as the skeleton of `ExamRoom.jif`. You will add additional files for `Exam`, `Student`, `Question`, and `IStudent`.

You must compile the principal files first, because the resulting class files must exist in order for references to them in `ExamRoom.jif` to exist. Use `jifc -e -nonrobust Alice.jif` and then repeat for `Bob.jif` and `Examiner.jif`. The principal classes will be created in the subdirectory `jif/principals/`.

Compile your code using `jifc -e -nonrobust -classpath . ExamRoom.jif` (to compile `ExamRoom` and everything it depends on; you could also compile an individual `.jif` file as needed). When all of your files compile, run the program using `jif -classpath . ExamRoom`. The expected output looks as follows:

```
> jif -classpath . ExamRoom
Starting exam.
Exam finished.
Student Alice got 1 points.
Student Bob got 0 points.
done.
>
```

(The scores may vary if your students score differently.)

2.4.2 Hints

- Use class parameterization. Depending on your solution you may or may not consider parameterizing class `Question`.
- Identify what variables contain sensitive information and how restrictive their labels should be. Assign confidentiality labels to class fields.
- Identify side effects that you have in your methods. This will tell you how begin-labels should look. If your method has side effects and may terminate in different ways, this will also affect end-labels.
- Use `NullPointerException` analysis to avoid ignoring exceptions and achieve cleaner code.
- You may be able to use the `Declassifier` pattern for array declassification. Below is an example of this class with a function for declassification of string arrays. Caution: This pattern is very sensitive to the current environment (this and `caller_pc`), and it can be hard to make it work correctly. If you try it and get stuck, it may be easier to think of a different way of solving the problem that doesn't require array declassification.

```
class Declassifier[principal P, label L] {
  public static String{L}[] {L}
  declassifyStringArray{L} (String{P:}[] {P:} x_0)
  where caller (P) {
    String{P:}[] {L} x = declassify(x_0, {L});
    if (x == null)
      return null;
  }
}
```

```

        String{L}[][]{L} y = new String[x.length];
    try {
        for (int i = 0; i < x.length; i++)
            y[i] = declassify(x[i], {L});
    } catch (ArrayIndexOutOfBoundsException ignored) {
    } catch (ArrayStoreException ignored) {
    }
    return y;
}
}

```

This class is parameterized over a principal P whose authority is required for declassification and a label L to which the data is downgraded. In this pattern, the function `declassifyStringArray()` accepts an array of strings labeled high and returns the low copy of it. Sample usage is:

```

String{Alice:}[][]{Alice:} highArray = ...
String{}[][]{} lowArray = Declassifier[Alice, {}].declassifyStringArray(highArray);

```

- Printing is side-effect in Jif and is rejected by type system unless you specify correct begin-label. Reuse code from `ExamRoom.main` function in order to print output to the console.
- More examples of Jif programs can be found in directories `$JIF/tests` and `$JIF/examples/battleship`.

2.4.3 Submission for part 3

Your writeup should include:

- How does your solution work?
- What labels do you use in your program? What do they protect?
- How do you use declassification? Could there be less declassification in your program? What data is declassified? Who declassifies it? Where in the program does it occur? When does it occur?
- Does your code prevent attacks that you found in part 1?

To test your code, I will recompile it and then run it using the commands listed above.

3 Submission and grading

Turn in **one** gzip tarball named **hw3_gXX.tgz**. Put all your files in the directory `gXX`, then create the tarball as follows:

```
> tar -czvf hw3_gXX.tgz gXX/
```

When the tarball is unzipped, it should result in a folder named `gXX` containing all your files. Inside this folder, we should find `gXX_hw3.pdf`, containing your writeup for all three parts.

We should also find a subfolder, `part1`, containing your version of `Student.java`.

We should also find a subfolder, `part3`, containing your code from part three, including the following .jif files: `Alice`, `Bob`, `Examiner`, `ExamRoom`, `Student`, `Question`, `IStudent`, `Exam`.

Part 2 will be graded entirely on the writeup; Parts 1 and 3 will be one-fourth writeup, three-fourths code.