Static And Dynamic Software Security Analysis

Lab 4

Prof. Jacques Klein & Marco Alecci

2024

For this lab, you will need to use a Linux machine with Java 8 installed and several software to perform static analysis. You can rely on the same Virtual Machine you used in the previous Lab: Virtual Machine

- Image size: 4.21GB, username = password = micsispa
- Lubuntu 19.10, Eclipse 4.14 (with Soot, Jasmin and Heros already imported)
- Soot 4.0.0 and Flowdroid 2020.02 (8cfd133) jars in ~/tools/

In the previous lab, you have seen how to build very simple analyses that explore the content of classes, methods, and statements with Soot. In this lab, you will implement your own DataFlow analysis using Soot. Remember, during the lecture, we have seen that we are dealing with *frameworks*, which means you do not have to reinvent everything. Indeed, Soot already provides this framework so that one can easily implement its own intra-procedural analysis. Read the following blog article: Intra-procedural analysis using Soot. Also, have a look at the `ReachingDefinitionAnalysis.java` file in the `ApkAnalysisTemplate` project to have an example of an intra-procedural analysis template with Soot.

Question 1 (1 point(s))
In class, we discussed Sign Analysis, which can be used to determine the sign of a variable's value at a specific point in the program. Another type of analysis is Odd/Even Analysis, which can be used to determine whether the value of a variable is odd or even. Propose a lattice for this type of analysis.

Question 2 (3 point(s))
Based on the `FlowAnalysis` class, describe the methods you need to override in order to implement your own intra-procedural analysis with Soot. What are they used for?

Question 3 (2 point(s))
How would you implement the odd/even analysis with this framework? What would N and A abstract types be used for in the case of the odd/even analysis?

Question 4 (10 point(s))
Implement the intra-procedural odd/even analysis using Soot. Describe your code.

Question 5 (4 point(s))
To test your implementation of the odd/even analysis, you have to apply it to methods of a program. We will test it on an Android app, consider the same app from the previous lab: APP. The following code gives you a piece of code from which you can get inspired to test your code. Remember that this is an *intra-procedural* analysis. It has to be in a "jtp" transformer (see previous lab). To showcase that your analysis works, apply it to different methods. Then when it is finished, show that for statements involving *int, or long* variables, it outputs the correct result.

```
@Override
protected void internalTransform(Body b, String phase, Map<String, String> opts) {
    SootMethod sm = b.getMethod();
    SignAnalysis sa = new SignAnalysis(new ExceptionalUnitGraph(b));
    for (Unit u: b.getUnits()) {
        System.out.println("Statement: " + u)
        System.out.println("Flow before statement: " + sa.getFlowBefore(u));
        System.out.println("Flow after stateemnt: " + sa.getFlowAfter(u));
}
```

The Reaching Definition Analysis is defined as follows:

Reaching Definitions Analysis

Let m be the method on which to perform the analysis. Let D be the set of all definition statements in method m. Let V be the set of all variables in method m. A definition $d \in D$ of a variable $v \in V$ reaches a program point p if d occurs on some path from the first statement to p and is not followed by any other definition of v on this path.

Question 6 (5 point(s))
This question is not mandatory, it is a bonus. Implement the reaching definition analysis and test it on the APK.