Applied Static Analysis

Software Technology Group
Department of Computer Science
Technische Universität Darmstadt
Dr. Michael Eichberg

I would like to thank Dominik Helm for creating this exercise.

If you have questions regarding OPAL don't hesitate to ask them in the OPAL Gitter channel!

IFDS

You should use the provided template. That project is preconfigured to use the latest snapshot version of OPAL and already contains the initial configuration. The template for this exercise can be found at: https://github.com/stg-tud/apsa/tree/master/2019/10-IFDS/Exercise/template

If the provided template does not compile, you may have an outdated version of OPAL. Please, make sure that you have downloaded the latest snapshot of OPAL. The easiest way to do so is to call "sbt cleanCache" and "sbt cleanLocal". If this doesn't help go to the your local .ivy cache and delete all jars relates to OPAL– they can be found in the respective sub folder. After that, when you compile the template, you should see that OPAL downloads the latest versions from the SNAPSHOT repository!

> ⚠️ Always ensure that you use the latest snapshot version. You can clean the latest (snapshot) version that you have downloaded using the command `sbt cleanCache cleanLocal` in your project's root folder.

An integrated JavaDoc of the latest snapshot version of OPAL that spans all sub-projects can be found at: www.opal-project.de

For further details regarding the development of static analysis using OPAL see the OPAL tutorial.

Logging of Sensitive Information

Develop an IFDS taint analysis using OPAL's IFDS support (`org.opalj.tac.fpcf.analyses.AbstractIFDSAnalysis`) which finds violations of the following rule taken from The CERT Oracle Secure Coding Standard for Java:

> FIO13-J: Do not log sensitive information outside a trust boundary.

Examples can be found in the `test` directory.
We consider `User#getName` as a source of sensitive information and `Logger#log` as a logging operation outside a trust boundary, i.e., a sink for the taint analysis.

The analysis should use all public methods of the project as entry points.

> To run the analysis against the provided test case, use `run -cp=<Path>/test`.

Task

Test your analysis using the provided tests. It should find violations in all methods named `noncompliant` (there are 7 of them).
If you find an additional false positive in `CorrelatedCalls#compliant` be prepared to explain why.

Hints

1. In OPAL's IFDS implementation, the flow functions are always given a set of valid facts. Still, you have to ensure that every fact you derive depends on at most one fact in the given input. If you want to remove a fact, that decision may not depend on any other fact. Otherwise, your analysis is not distributive and

may fail. Don’t forget to include all unchanged facts in your results as well.

2. Use case classes as subtypes of `Fact` to model different types of taints, including local variables, elements of arrays, static and instance fields. What information does each kind of value need? (Start implementing your analysis by just considering local variables then extend the analysis; consider arrays as the last step!)
3. You may report violations simply by printing information about the violation to the command line.
4. You can make use of the `callToReturnFlow` method to introduce tainted values when `User#getName` is called as well as to report violations if the argument of `Logger#log` is tainted.
5. If violations are reported twice, report only if the successor statement `succ` is in a basic block (`.node.isBasicBlock` as opposed to an exit node). That is because the flow functions are invoked for every possible successor of the current statement which might include terminating the method because of an exception.