Introduction to Soot

CS6235

What is Soot?

- At its core, Soot is a Java Optimization Framework.
- It is a product out of the Sable research group from McGill University.
- Open source.

What can Soot do?

What do *you* want Soot to do?

- Use it as a simple parser?
- Use one of its many out of the box optimizations?
- Use one of its many out of the box analyses?
- Or extend it to implement your own program analysis/optimization algorithm?

How to obtain Soot

Soot is available both as source code for you to build, and prebuilt jars ready for use.

Latest prebuilt releases are available here - https://repo1.maven.org/maven2/org/soot-oss/soot/

The most basic Soot invocation

This simple command will invoke Soot on a provided main-class, run a few default whole-program optimizations, and emit the (transformed) classes in an output directory of choice.

```
$ java -jar soot-4.3.0-jar-with-dependencies.jar -pp -app -cp . -d out Main
```

Soot Intermediate Representations (IRs)

One of Soot's biggest strengths is that it offers four different purpose-built Intermediate Representations (IRs) to support different purposes of analysis. Each of them are at a different level of abstraction:

- **Baf** - a streamlined, stack-based representation of bytecode designed to be easy to manipulate.
- **Jimple** - a 3-address representation of bytecode designed to be suitable for optimizations.
- **Shimple** - an SSA version of Jimple
- **Grimp** - a version of Jimple suited for code decompilation and inspection.

Jimple is Simple

Quick demo..

A slight modification of the previous command emits Jimple instead of class files:

```
$ java -jar soot-4.3.0-jar-with-dependencies.jar -pp -app -cp .  -f J -d out
Main
```

On Jimple

- Jimple can be created directly in Soot, or from source code, or even from bytecode (class files).
- Jimple statements correspond to Soot Units.
- Jimple has about 15 different statements - for example, AssignStmt, IfStmt, InvokeStmt, ReturnStmt, ReturnVoidStmt, etc.
- Most (if not all) code manipulation you will perform will be via Jimple - so it pays to be familiar with Jimple.

Some Buzzwords to Keep in Mind

You will see these classes being used frequently in Soot-based applications:

- The **Scene** - represents the complete environment housing the analysis.
- The **SootClass** - represents a single class, think of it as the unit of transaction representing a class in Soot.
- The **SootMethod** - represents a single method, think of it as the unit of transaction representing a method in Soot.
- The **SootField** - you get the idea.
- The **Body** - represents a method body, and comes in different flavors - corresponding to different IRs.

Essentially, these are data structures that you will most likely use while manipulating code for analysis.

Introduction to Extending Soot

..and manipulating the IR.

Another quick demo..

Essential Resources

- Soot Command Line Args reference
- Soot Survivor's Guide
- Soot Github repo