Integrating Soot into QiLin

Dongjie He

March 2, 2022

QiLin is a pointer analysis framework built on top of Soot (with minor modification). This guide introduces how to build a modified version of Soot that can be successfully integrated into QiLin.

We use the latest version, i.e., `4.2.1`, for illustrating.

1. **Download Source**

The latest version of Soot can be downloaded from `https://github.com/soot-oss/soot/archive/4.2.1.tar.gz`.

```
$ wget https://github.com/soot-oss/soot/archive/4.2.1.tar.gz
$ tar xvf 4.2.1.tar.gz
$ cd soot-4.2.1
```

2. **Modify Soot**

Qilin implements a new context-insensitive pointer analysis which is also named Spark. We reuse many data structures from the original Spark in Soot. To avoid the name clash, we decide to remove the original Spark along with the projects that depend on it. Specifically, we have removed the following two directories:

```
$ rm -rf src/main/java/soot/jimple/spark/
$ rm -rf src/main/java/soot/jimple/toolkits/thread/
```

We have also removed the following 8 files:

```
$ rm src/main/java/soot/jimple/toolkits/callgraph/CallGraphBuilder.java
$ rm src/main/java/soot/jimple/toolkits/callgraph/OnFlyCallGraphBuilder.java
$ rm src/main/java/soot/jimple/toolkits/callgraph/CHATransformer.java
$ rm src/main/java/soot/jimple/toolkits/pointer/CodeBlockRWSet.java
$ rm src/main/java/soot/jimple/toolkits/pointer/InstanceKey.java
$ rm src/main/java/soot/xml/TagCollector.java
$ rm src/main/java/soot/XMLAttributesPrinter.java
$ rm src/main/java/soot/dava/toolkits/base/misc/ThrowFinder.java
```

These files depends on Spark pointer analysis. We will add these file back in the future if we require them.

Now, let us remove codes related to `spark` and `thread` in the following five files.

- `src/main/generated/singletons/soot/Singletons.java`
- `src/main/java/soot/PackManager.java`
- `src/main/java/soot/G.java`
- In `src/main/java/soot/Scene.java`, we replace `SparkField` with `SootField` and then remove all relevant code that may cause compile errors
- We modify `src/main/java/soot/SootField.java` to disable the class implement `SparkField` interface.

These codes will cause compile errors since they have references to files that we have already removed above.

Now, the code could be compile successfully with the following command:

```
$ mvn clean compile assembly:single
```

At last, let us append a few lines of code to `src/main/java/soot/Scene.java` to support QiLin.

(1). We add `addBasicClass("java.io.UnixFileSystem");` into method `private void addSootBasicClasses()` to support a native reflection in QiLin.

(2). We add the following lines to method `protected void addReflectionTraceClasses()` to support the reflection log generated by Tamiflex.

```
} else if (kind.equals("Field.toString") || kind.equals("Method.toString") || kind.equals("Constructor.toString")) {
        classNames.add(signatureToClass(target));
} else if (kind.equals("Field.getName") || kind.equals("Field.getDeclaringClass")) {
        classNames.add(signatureToClass(target));
} else if (kind.equals("Class.getDeclaredField") || kind.equals("Class.getDeclaredMethod")) {
        classNames.add(signatureToClass(target));
} else if (kind.equals("Class.getDeclaredFields") || kind.equals("Class.getDeclaredMethods")) {
        if (!target.startsWith("[")) {
                classNames.add(target);
        }
} else if (kind.equals("Class.getFields") || kind.equals("Class.getMethods")) {
        classNames.add(target);
} else if (kind.equals("Class.getMethod") || kind.equals("Class.getField")) {
        classNames.add(signatureToClass(target));
} else if (kind.equals("Constructor.getModifiers") || kind.equals("Field.getModifiers")) {
        classNames.add(signatureToClass(target));
} else if (kind.equals("Array.newInstance")) {
        // do nothing
} else if (kind.equals("Method.getModifiers") || kind.equals("Method.getName") || kind.equals("Method.toGenericString")) {
        classNames.add(signatureToClass(target));
} else if (kind.equals("Method.getDeclaringClass")) {
        classNames.add(signatureToClass(target));
}
```

We should also fix a bug in soot in resolving non-special call target. In method `getSignaturePolymorphicMethod` of `FastHierarchy`, we need to append these lines:

```
String subsig = SootMethod.getSubSignature(name, parameterTypes, returnType);
candidate = concreteType.getMethodUnsafe(subsig);
if (candidate != null) {
        return candidate;
}
```

3. **Ready to go**

Now, it's ready to compile soot and integrate it into QiLin.

Again, let us use the following line to compile Soot:

```
$ mvn clean compile source:jar assembly:single
```

And then, let us move Soot to `qilin/libs/` and our job is done.

```
$ mv target/sootclasses-trunk-jar-with-dependencies.jar qilin/libs/
$ mv target/sootclasses-trunk-sources.jar qilin/libs/
```