Static Analysis of Java Dynamic Proxies

George Fourtounis (gfour@di.uoa.gr)
George Kastrinis (gkastrinis@di.uoa.gr)
Yannis Smaragdakis (yannis@smaragd.org)

University of Athens, Greece

ISSTA’18, July 17, 2018, Amsterdam, Netherlands
Dynamic proxies in Java

- GoF, “Proxy” design pattern:
  “Provide a surrogate or placeholder for another object to control access to it.”
- Proxy objects can be pregenerated at compile-time or dynamically generated at runtime (more flexibility)
- Java 1.3 introduced an API to generate dynamic proxies
Java dynamic proxies API

- “Give me some interfaces and method dispatch logic and I'll dynamically generate a class that implements all such interfaces”
- Method dispatch logic = the “invocation handler”, essentially an interpreter that reflectively handles all attempted method calls
- API + invocation handler = implemented API
- Using dynamic code generation/loading
Example

interface I {
    Object getField();
    float mult(float x, float y);
}

class A implements I {
    Object getField() ... 
    float mult(float x, float y) ... 
}
Example

class AHandler implements InvocationHandler {
    private A a;
    public AHandler(A a) { this.a = a; }
    public Object invoke(Object proxy, Method method, Object[] args) {
        if (method.getName().equals("getField")) return new B();
        else if (method.getName().equals("mult")) {
            float x = ((Float)args[0]).floatValue();
            float y = ((Float)args[1]).floatValue();
            return a.mult(x, y);
        } else return null;
    }
}

class A implements I {
    Object getField() ...
    float mult(float x, float y) ...
}

interface I {
    Object getField();
    float mult(float x, float y);
}
Example

class A implements I {
    Object getField() ...
    float mult(float x, float y) ...
}

class AHandler implements InvocationHandler {
    private A a;
    public AHandler(A a) { this.a = a; }
    public Object invoke(Object proxy, Method method, Object[] args) {
        if (method.getName().equals("getField")) return new B();
        else if (method.getName().equals("mult") {
            float x = ((Float)args[0]).floatValue();
            float y = ((Float)args[1]).floatValue();
            return a.mult(x, y);
        } else return null;
    }
}

interface I {
    Object getField();
    float mult(float x, float y);
}

handler = new AHandler(new A());
proxy = newProxyInstance({I.class}, handler);
What values does “proxy” point to?

```java
I proxy = newProxyInstance(interfaces, handler);
```

Problems:

- `newProxyInstance()` is a black box: dynamic code generation means no statically-available classes
- generated class depends on interfaces and handler, which are runtime values
Dynamic proxies are a problem

In a recent survey of 461 open-source Java projects, Landman et al. find that 21% of them use dynamic proxies

- “very harmful for static analysis”
- “avoid the use of dynamic proxies at any cost”
- “no clear solution seems to be on the horizon”

We have a solution!

- Don't analyze the body of `newProxyInstance()`, model instead the Proxy API semantics.
- To model the API we need:
  - a points-to analysis
  - good support for Java reflection
  - exception analysis
Our solution

- Doop, a static analysis framework for Java
  - analyses written in Datalog
  - already provides a points-to analysis (with several context-sensitivity flavors), a reflection analysis (with substring analysis), and an exception analysis
- Add rules to model dynamic proxies
  - mutually recursive: the new rules use existing analyses but also inform them
A core rule (informally)

```
handler = new AHandler(new A())
proxy = newProxyInstance({I.class}, handler)
If:
  • an instruction i calls newProxyInstance(),
  • the interfaces argument points to an array that contains the Class for interface t_i,
  • the handler argument points to a value obj_handler, and
  • the instruction returns a value in v_ret,
then v_ret points to an object that proxies t_i using obj_handler
```
A core rule (in Doop)

VarPointsTo(v_ret , obj_proxy),
ProxyObjectHandler(obj_proxy, obj_handler) ←
   Call(i, "Proxy.newProxyInstance"),
ActualArg(i, 1, arg_ifaces),
VarPointsTo(arg_ifaces, obj_ifaces),
ArrayContentsPointTo(obj_ifaces , Class_i),
ReifiedType(t_i , Class_i),
ActualArg(i, 2, arg_handler),
VarPointsTo(arg_handler, obj_handler),
AssignRetValue(i, v_ret),
ReifiedProxyInstance(t_i, i, obj_proxy).
A core rule (in Doop)

\[
\begin{align*}
\text{VarPointsTo}(v\_ret, \text{obj\_proxy}), \\
\text{ProxyObjectHandler}(\text{obj\_proxy}, \text{obj\_handler}) & \leftarrow \\
\text{Call}(i, "\text{Proxy.newProxyInstance}") , \\
\text{ActualArg}(i, 1, \text{arg\_ifaces}), \\
\text{VarPointsTo}(\text{arg\_ifaces}, \text{obj\_ifaces}), \\
\text{ArrayContentsPointTo}(\text{obj\_ifaces}, \text{Class\_i}), \\
\text{ReifiedType}(t\_i, \text{Class\_i}), \\
\text{ActualArg}(i, 2, \text{arg\_handler}), \\
\text{VarPointsTo}(\text{arg\_handler}, \text{obj\_handler}), \\
\text{AssignRetValue}(i, v\_ret), \\
\text{ReifiedProxyInstance}(t\_i, i, \text{obj\_proxy}).
\end{align*}
\]
A core rule (in Doop)

VarPointsTo(v_ret, obj_proxy),
ProxyObjectHandler(obj_proxy, obj_handler) ←
   Call(i, "Proxy.newProxyInstance"),
   ActualArg(i, 1, arg_ifaces),
   VarPointsTo(arg_ifaces, obj_ifaces),
   ArrayContentsPointTo(obj_ifaces, Class_i),
   ReifiedType(t_i, Class_i),
   ActualArg(i, 2, arg_handler),
   VarPointsTo(arg_handler, obj_handler),
   AssignRetValue(i, v_ret),
   ReifiedProxyInstance(t_i, i, obj_proxy).
A core rule (in Doop)

VarPointsTo(v_ret, obj_proxy),
ProxyObjectHandler(obj_proxy, obj_handler) ←
   Call(i, "Proxy.newProxyInstance"),
   ActualArg(i, 1, arg_ifaces),
   VarPointsTo(arg_ifaces, obj_ifaces),
   ArrayContentsPointTo(obj_ifaces, Class_i),
   ReifiedType(t_i, Class_i),
   ActualArg(i, 2, arg_handler),
   VarPointsTo(arg_handler, obj_handler),
   AssignRetValue(i, v_ret),
   ReifiedProxyInstance(t_i, i, obj_proxy).
A core rule (in Doop)

VarPointsTo(v_ret, obj_proxy),
ProxyObjectHandler(obj_proxy, obj_handler) ←
  Call(i, "Proxy.newProxyInstance"),
  ActualArg(i, 1, arg_ifaces),
  VarPointsTo(arg_ifaces, obj_ifaces),
  ArrayContentsPointTo(obj_ifaces, Class_i),
  ReifiedType(t_i, Class_i),
  ActualArg(i, 2, arg_handler),
  VarPointsTo(arg_handler, obj_handler),
  AssignRetValue(i, v_ret),
  ReifiedProxyInstance(t_i, i, obj_proxy).
A core rule (in Doop)

VarPointsTo(v_ret, obj_proxy),
ProxyObjectHandler(obj_proxy, obj_handler) ←
   Call(i, "Proxy.newProxyInstance"),
   ActualArg(i, 1, arg_ifaces),
   VarPointsTo(arg_ifaces, obj_ifaces),
   ArrayContentsPointTo(obj_ifaces, Class_i),
   ReifiedType(t_i, Class_i),
   ActualArg(i, 2, arg_handler),
   VarPointsTo(arg_handler, obj_handler),
   AssignRetValue(i, v_ret),
   ReifiedProxyInstance(t_i, i, obj_proxy).
Looks simple!

- If you already have all the other analyses
- And interfacing with them is easy
- Mutual recursion ($\text{VarPointsTo}$ in last slide)
- In total, 29 Datalog rules (also taking care of corner cases, such as argument boxing, special `java.lang.Object` methods, proxy spec exceptions)
Evaluation 1: XCorpus

XCorpus, a suite of Java programs containing:

- binaries ready for static analysis
- entry points with good code coverage
- report about calls to newProxyInstance()

Taking the XCorpus report as ground truth, does our analysis resolve calls to proxies?

Evaluation 1: XCorpus

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>XCorpus reported</th>
<th>Doop reachable</th>
<th>Opt-handled</th>
<th>Def-handled</th>
<th>Invocation handler edges</th>
<th>Analysis time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Opt-Reflective</td>
<td></td>
</tr>
<tr>
<td>aoi-2.8.1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>–</td>
<td>6,911</td>
<td>206min, timeout (4hr)</td>
</tr>
<tr>
<td>batik-1.7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>411</td>
<td>8min, 87min</td>
</tr>
<tr>
<td>castor-1.3.1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>–</td>
<td>9,384</td>
<td>26min, timeout (4hr)</td>
</tr>
<tr>
<td>drools-7.0.0.Beta6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>–</td>
<td>15,205</td>
<td>143min, timeout (4hr)</td>
</tr>
<tr>
<td>guava-21.0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>5,350</td>
<td>11min, 42min</td>
</tr>
<tr>
<td>jedit-4.3.2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4,516</td>
<td>29min, 213min</td>
</tr>
<tr>
<td>jhotdraw-7.5.1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>880</td>
<td>11min, 183 min</td>
</tr>
<tr>
<td>jrat-0.6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>3 min, 8 min</td>
</tr>
<tr>
<td>mockito-core-2.7.17</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>13</td>
<td>4 min, 8 min</td>
</tr>
<tr>
<td>picocountainer-2.10.2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>881</td>
<td>2 min, 213 min</td>
</tr>
<tr>
<td>pmd-4.2.5</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>–</td>
<td>9</td>
<td>19 min, timeout (4hr)</td>
</tr>
<tr>
<td>quartz-1.8.3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>12</td>
<td>7 min, 25 min</td>
</tr>
<tr>
<td>squirrel_sql-3.1.2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10min, timeout (4hr)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>20</strong></td>
<td><strong>19</strong></td>
<td><strong>19</strong></td>
<td><strong>11</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Def- vs. Opt-reflective: full reflection vs. naive reflection support for scalability
- Both miss the same benchmark (squirrel) due to lack of coverage by the XCorpus entry points
Evaluation 2: okhttp/guice

- Ground truth from manual inspection
- OkHttp, a popular HTTP library
  - OpenJDK/Android portability with dynamic proxies
  - we analyze okhttp-mockwebserver (its test server)
- Google Guice, dependency injection (DI) framework
  - we analyze guice-jndi (standalone test JNDI client)
  - many call-graph edges, as in XCorpus picocontainer (another DI library)
Conclusion

- Dynamic proxies are no longer a source of unsoundness in static analysis!
- We can analyze code with proxies using limited support of reflection
- Writing analyses in mutual-recursive style is easy
Thank you!