Program Analysis for Security

Suhabe Bugrara
Stanford University

Web Application Code

1. javax.sql.Connection con = . . .;
2. javax.servlet.http.HttpServletRequest request = . . .;
3. String username = request.getParameter("username");
4. String query = "SELECT * FROM Users " + " WHERE name = '" + username + "'";
5. con.execute(query);

Program Analyzers

Soundness, Completeness

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Soundness</strong></td>
<td>If the program contains an error, the analysis will report a warning.</td>
</tr>
<tr>
<td><strong>Completeness</strong></td>
<td>If the analysis reports an error, the program will contain an error.</td>
</tr>
</tbody>
</table>

Complete Incomplete

- Undecidable
  - Reports all errors
  - Reports no false alarms
- Decidable
  - Reports all errors
  - May report false alarms
- Undecidable
  - May not report all errors
  - Reports no false alarms
- Decidable
  - May not report all errors
  - May report false alarms

Manual testing only examines small subset of behaviors
Sound Over-approximation of Behaviors

Software

Traditional Static Analysis

Does this program ever crash?

Try analyzing without approximating...

infeasible path! ... program will never crash

non-termination! ... therefore, need to approximate

\[ d_{out} = f(d_{in}) \]
What is the space of dataflow elements, $\Delta$?
What is the least upper bound operator, $\sqcup$?

Try analyzing with "signs" approximation...

Try analyzing with "path-sensitive signs" approximation...

Tainted Object Propagation

```
String username = req.getParameter("username");
StringBuffer buf = new StringBuffer();
buf.append("SELECT * FROM Users ");
buf.append("WHERE name = " + username);
buf.append(""");
String query = buf.toString();
con.execute(query);
```
Tainted Object Propagation

<table>
<thead>
<tr>
<th>Term</th>
<th>Descriptor</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Object</td>
<td>Method</td>
<td>HttpServletRequest.getParameter(String)</td>
</tr>
<tr>
<td>Parameter</td>
<td></td>
<td>return</td>
</tr>
<tr>
<td>Access path</td>
<td></td>
<td>ε</td>
</tr>
<tr>
<td>Sink Object</td>
<td>Method</td>
<td>Connection.execute(String)</td>
</tr>
<tr>
<td>Parameter</td>
<td></td>
<td>first argument</td>
</tr>
<tr>
<td>Access path</td>
<td></td>
<td>ε</td>
</tr>
<tr>
<td>Derivation</td>
<td>Method</td>
<td>StringBuffer.append(String)</td>
</tr>
<tr>
<td>From Parameter</td>
<td></td>
<td>this</td>
</tr>
<tr>
<td>To Access Path</td>
<td></td>
<td>ε</td>
</tr>
</tbody>
</table>

Security Violation

Complication of Aliasing

String username = req.getParameter("username");
StringBuffer buf1 = new StringBuffer();
StringBuffer buf2 = buf1;
buf1.append(username);
String query = buf2.toString();
con.execute(query);

No security violation found!

Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object creation</td>
<td>o1: T v = new T();</td>
<td>Creates a new heap object o1 of type T, and makes variable v point to o1</td>
</tr>
<tr>
<td>copy</td>
<td>v = w;</td>
<td>Makes v point to whatever heap object w currently points to</td>
</tr>
<tr>
<td>field store</td>
<td>v.f = w;</td>
<td>Let v point to heap object o1. Let w point to heap object o2. Makes the field f in o1 now point to o2</td>
</tr>
<tr>
<td>field load</td>
<td>v = w.f;</td>
<td>Let w point to heap object o2. Makes v point to whatever the field f in o2 points to.</td>
</tr>
</tbody>
</table>

Pointer Analysis

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>points-to (v, o)</td>
<td>variable v can point to heap object o</td>
</tr>
<tr>
<td>heap-points-to (o1, o2)</td>
<td>field f of heap object o1 can point to heap object o2</td>
</tr>
</tbody>
</table>

Predicate Rules

1: "v = w;" 
   points-to (v, o) 
   points-to (w, o) 
2: "v.f = w;" 
   points-to (v, o) 
   points-to (w, o) 
   heap-points-to (o1, o2) 
3: "v = w.f;" 
   points-to (v, o) 
   points-to (w, o) 
   heap-points-to (o1, o2) 
4: "v = w;" 
   points-to (v, o) 
   points-to (w, o)
Context Sensitivity

```
class DataSource {
    private String url;
    public DataSource(String url) {
        this.url = url;
    }
    String getUrl() {
        return this.url;
    }
}
```

```
String passedUrl = request.getParameter("...");
DataSource ds1 = new DataSource(passedUrl);
String localUrl = "http://localhost/";
DataSource ds2 = new DataSource(localUrl);
String s1 = ds1.getUrl();
String s2 = ds2.getUrl();
StringBuilder buf1 = new StringBuilder();
buf1.append(s2);
String query = buf1.toString();
Connection con = ...;
con.execute(query);
```
1: int symbolic_bad_abs (int x) {
  2: add_constraints(x >= INT_MIN, x <= INT_MAX);
  3: ret = new symbol;
  4: if (fork() == child) {
  5: add_constraints(x < 0, ret = -x);
  6: return ret;
  7: } else
  8: add_constraints(x >= 0);
  9: if (fork() == child) {
10: add_constraints(x = 12345678, ret = -x);
11: return ret;
12: } else
13: add_constraints(x != 12345678);
14: add_constraints(ret = x);
15: return ret;
16: }

1: int main (void) {
  2: unsigned i, t, a[4] = { 1, 3, 5, 2};
  3: make_symbolic(a);
  4: if (i >= 4)
  5: exit(0);
  6: char *p = (char *) a + i * 4;
  7: *p = *p – 1;
  8: t = a[*p] / a[i];
  9: if (t == 2)
10: assert (i == 1);
11: else
12: assert (i == 3);
13: }

Questions