Finding Security Errors in Java Applications Using Lightweight Static Analysis

Benjamin Livshits
Computer Science Lab
Stanford University
Vulnerability Research Focus

- Static analysis for vulnerability detection
- Until recently, a large portion of server-side software was written in C/C++
- Vulnerabilities come from poor language and API design:
  - Buffer overruns
  - Format string violations
- More profound:
  - Time-of-check-time-of-use errors (TOCTOU)
Security Errors in Java are Emerging

- Situation is changing…
- More and more Web-based applications are written in Java
- Web-based applications are good vulnerability targets
- New categories of errors in this domain

SQL Injections

LDAP injection

HTTP response splitting

Bad session stores

Cross-site scripting

Forceful browsing
Finding Errors with Static Analysis

Our approach:
- Static Analysis has been proven useful for finding security errors in C programs
- Apply to Java to find new categories of errors

What we did:
- Created user-friendly code analysis tools
- Based on Eclipse, an open-source Java IDE
- Easy to run on your own code
- Focused on two types of errors so far
  - Bad session stores
  - SQL injections
- We look at these two error patterns next…
Focus on Two Error Patterns

Bad session store

Object o = ...
HttpSession s = ...
s.setAttribute("name", o);

- A common pattern in servlets leading to errors
- HttpSession need to be saved to disk
- Object o must implement java.io.Serializable
- Bad API design
- Can lead to crashes and DOS attacks

SQL injection

String query =
    request.getParameter("name");
java.sql.Statement stmt = ...
stmt.executeQuery(query);

- Unchecked input passed to backend database
- Carefully crafted input containing SQL will be interpreted by database
- Can be used by the malicious user to
  - read unauthorized info,
  - delete data,
  - even execute commands,
  - etc.
Our Tools…

- Bad session stores
- Identify all sources of user information
- SQL Injections
- Identify all sinks where sensitive data can flow
- Filter out sinks that take constant strings
- Help to follow data from sources to sinks
- Report errors
- Report errors

Look at the type of the 2nd argument of `setAttribute`:
- `setAttribute(..., expr);

Do a type check for `expr` that don’t implement `java.io.Serializable`

Report errors
```java
if(values != null) {
    MessageFormat mf = new MessageFormat(sql);
    for(int i = 0; i < values.length; i++) {
        if(values[i] instanceof String) {
            mf.setFormat(i, 1);
        } else if(values[i] instanceof Boolean) {
            values[i] = (Boolean)values[i].booleanValue() ? new Long(1) : new Long(0);
        } else if(values[i] instanceof java.util.Date) {
            mf.setFormat(i, dbDF);
        } else if(values[i] instanceof java.sql.Date) {
            mf.setFormat(i, dbDF);
        } else if(values[i] instanceof java.lang.Integer) {
            mf.setFormat(i, dbInt);
        } else if(values[i] instanceof java.lang.Long) {
            mf.setFormat(i, dbInt);
        }
    }
    sql = mf.format(values);
    //System.out.println("sql=" + sql);
    st.execute(sql);
    st.close();
    return nextId;
}
```
Benchmarks

- 10 Web-based applications
- Widely deployed and vulnerable to attacks
- Most blogging tools
- Quite large – 10s of KLOC
- Rely on very large J2EE libs

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>LOC</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapleblog</td>
<td>2,156</td>
<td>36</td>
</tr>
<tr>
<td>personalblog</td>
<td>2,317</td>
<td>38</td>
</tr>
<tr>
<td>blueblog</td>
<td>4,142</td>
<td>38</td>
</tr>
<tr>
<td>blogwelder</td>
<td>4,901</td>
<td>33</td>
</tr>
<tr>
<td>javablog</td>
<td>5,184</td>
<td>79</td>
</tr>
<tr>
<td>snipsnap</td>
<td>9,671</td>
<td>1,331</td>
</tr>
<tr>
<td>blojsom</td>
<td>14,382</td>
<td>30</td>
</tr>
<tr>
<td>jboard</td>
<td>17,368</td>
<td>138</td>
</tr>
<tr>
<td>pebble</td>
<td>30,319</td>
<td>169</td>
</tr>
<tr>
<td>roller</td>
<td>47,044</td>
<td>267</td>
</tr>
</tbody>
</table>

Total 137 K 2,159
Results for Bad Session Stores

- Found 14 errors
- 8 false positives
- 37% false pos rate
- Why false positives?
  - Declared types are too wide
  - Can improve with better type info from pointer analysis

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>All</th>
<th>Bad</th>
<th>Errors</th>
<th>False pos.</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapleblog</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>personalblog</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>blueblog</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>blogwelder</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>javablog</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>snipsnap</td>
<td>28</td>
<td>12</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>blojsom</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>jboard</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pebble</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>roller</td>
<td>24</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

| Total         | 75  | 22  | 14     | 8          |
Results for SQL Injections

- Found 6 errors
- Can find “low-hanging” errors
- Easy when sources and sinks are “close”
- Often they are very far apart
- Many require more elaborate analysis

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Sources</th>
<th>All sinks</th>
<th>Unsafe sinks</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapleblog</td>
<td>8</td>
<td>16</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>personalblog</td>
<td>29</td>
<td>35</td>
<td>27</td>
<td>1</td>
</tr>
<tr>
<td>blueblog</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>blogwelder</td>
<td>115</td>
<td>24</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>javablog</td>
<td>12</td>
<td>42</td>
<td>38</td>
<td>0</td>
</tr>
<tr>
<td>snipsnap</td>
<td>195</td>
<td>33</td>
<td>33</td>
<td>1</td>
</tr>
<tr>
<td>blojsom</td>
<td>12</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>jboard</td>
<td>3</td>
<td>18</td>
<td>17</td>
<td>3</td>
</tr>
<tr>
<td>pebble</td>
<td>109</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>roller</td>
<td>81</td>
<td>45</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>560</strong></td>
<td><strong>216</strong></td>
<td><strong>188</strong></td>
<td><strong>6</strong></td>
</tr>
</tbody>
</table>
Summary

Created lightweight interactive tools for finding security errors in Java

Found a total of 20 errors

However, there are
  - false positives and
  - “unknowns” – potential errors our tools can’t address

Conclusion:
  - Our tools are good for finding simpler errors
  - Hard errors often require a stronger analysis of data propagation
  - Working on a pointer analysis-based approach