Road to Confidence in IT Systems: SAMATE’s SATE and SRD projects

Paul E. Black
paul.black@nist.gov
Software Assurance Metrics And Tool Evaluation (SAMATE) project

Current Areas of Concentration

- Static source code security analyzers
- Static Analysis Tool Exposition (SATE)
- SAMATE Reference Dataset (SRD)
- Vote tools evaluation methodology
- Studies

http://samate.nist.gov/
SAMATE’s Niche

Mature Process

Resilient Execution Environment

Confidence in IT Systems

\[ C = f(p, s, e) \]

Study, Review, and Test the Software
What is a Static Analyzer?

- Static analyzers check programs for bugs.
- They can’t catch everything
  - e.g. sending data: should it be encrypted?
How Good are Static Analyzers?

- Does this analyzer find all bugs? 
  - without too many false positives …
- Does it find all the bugs we know about?
- How well does it do for important bugs?
- How much is my confidence increased by using it?
- Is running it a good use of my time?
Static Analysis Tool Exposition
SATE

● Goals:
  – Gather large test sets to enable empirical research
  – Encourage improvement of tools
  – Speed adoption of tools by objectively demonstrating their use on real software

● Steps
  – We choose open source programs with security implications
  – Participants run tools and return reports
  – We analyze reports
  – Everyone shares observations at a workshop
  – We release final reports and data later

http://samate.nist.gov/SATE.html
SATE Overview

- We held four SATEs since 2008.
- Test sets were in C/C++, Java, and PHP and consisted of about 26,000,000 LoC.
- 19 teams on 4 continents participated.
- We received a combined 128,043 warnings*

- We better understand subtleties of finding, designating, and counting weaknesses.
How can We Test Static Analyzers?

- Come up with a small set of test programs with known bugs.
- The result of running a tool on the set is correlated with the result on large, real-world programs.
SAMATE Reference Dataset (SRD)

- Public repository of over 60,000 test cases in C++, C, Java, C#, PHP, etc. covering 147 classes of weaknesses.

- User can search by language, weakness, code construct, etc.

- Contributions from Fortify, Defence R&D Canada, Klocwork, MIT Lincoln Laboratory, Praxis, Secure Software, and others.

samate.nist.gov/SRD
<table>
<thead>
<tr>
<th>Issue</th>
<th>Date</th>
<th>Language</th>
<th>Source Type</th>
<th>Author</th>
<th>Severity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>58</td>
<td>2005-11-02</td>
<td>Java</td>
<td>Source Code</td>
<td>SecureSoftware</td>
<td>C</td>
<td>Not using a a random initialization vector with Cipher Block ...</td>
</tr>
<tr>
<td>71</td>
<td>2005-11-07</td>
<td>Java</td>
<td>Source Code</td>
<td>SecureSoftware</td>
<td>C</td>
<td>Omitting a break statement so that one may fall through is often ...</td>
</tr>
<tr>
<td>1552</td>
<td>2006-06-22</td>
<td>Java</td>
<td>Source Code</td>
<td>Jeff Meister</td>
<td>C</td>
<td>Tainted input allows arbitrary files to be read and written.</td>
</tr>
<tr>
<td>1553</td>
<td>2006-06-22</td>
<td>Java</td>
<td>Source Code</td>
<td>Jeff Meister</td>
<td>C</td>
<td>Tainted input allows arbitrary files to be read and written. ...</td>
</tr>
<tr>
<td>1554</td>
<td>2006-06-22</td>
<td>Java</td>
<td>Source Code</td>
<td>Jeff Meister</td>
<td>C</td>
<td>Two file operations are performed on a filename, allowing a filenamer</td>
</tr>
<tr>
<td>1567</td>
<td>2006-06-22</td>
<td>Java</td>
<td>Source Code</td>
<td>Jeff Meister</td>
<td>C</td>
<td>The credentials for connecting to the database are hard-wired ...</td>
</tr>
<tr>
<td>1568</td>
<td>2006-06-22</td>
<td>Java</td>
<td>Source Code</td>
<td>Jeff Meister</td>
<td>C</td>
<td>The credentials for connecting to the database are hard-wired ...</td>
</tr>
<tr>
<td>1569</td>
<td>2006-06-22</td>
<td>Java</td>
<td>Source Code</td>
<td>Jeff Meister</td>
<td>C</td>
<td>The credentials for connecting to the database are hard-wired ...</td>
</tr>
<tr>
<td>1570</td>
<td>2006-06-22</td>
<td>Java</td>
<td>Source Code</td>
<td>Jeff Meister</td>
<td>C</td>
<td>An exception leaks internal path information to the user.</td>
</tr>
<tr>
<td>1571</td>
<td>2006-06-22</td>
<td>Java</td>
<td>Source Code</td>
<td>Jeff Meister</td>
<td>C</td>
<td>An exception leaks internal path information to the user. (fixed ...</td>
</tr>
<tr>
<td>1579</td>
<td>2006-06-22</td>
<td>Java</td>
<td>Source Code</td>
<td>Jeff Meister</td>
<td>C</td>
<td>Tainted output allows log entries to be forged.</td>
</tr>
</tbody>
</table>
public class File1_bad extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
    {
        res.setContentType("text/html");
        ServletOutputStream out = res.getOutputStream();
        out.println("<HTML><HEAD><TITLE>Test</TITLE></HEAD><BODY><blockquote><pre>");

        String name = req.getParameter("name");
        String msg = req.getParameter("msg");
        if(name != null) {
            try {
                File f = new File("/tmp", name);  // BAD *
                if(msg != null) {
                    FileWriter fw = new FileWriter(f);  // BAD */
                    fw.write(msg, 0, msg.length());
                    fw.close();
                    out.println("message stored");
                } else {
                    String line;
                    BufferedReader fr = new BufferedReader(new FileReader(f));
                    while((line = fr.readLine()) != null)
                        out.println(line);
                }
            } catch(Exception e) {
                throw new ServletException(e);
            }
        }
    }
}