My Life With Bugs, or, Why I Believe in Combinatorial Testing

Paul E. Black
National Institute of Standards and Technology
http://www.nist.gov/
paul.black@nist.gov
Outline

- SAMATE, my current program, briefly
- Some tough and bizarre bugs
- How many conditions do faults need?
- Combinatorial Testing
  - What is it?
  - Why does it work?
  - State of the Art
What is NIST?

- U.S. National Institute of Standards and Technology
- A non-regulatory agency in Dept. of Commerce
- 3,000 employees + adjuncts
- Gaithersburg, Maryland and Boulder, Colorado
- Primarily research, not funding
- Over 100 years in standards and measurements: from dental ceramics to text retrieval, from quantum computers to fire codes, from body armor to DNA forensics, from biometrics to whale blubber
The NIST SAMATE Project

- **Software Assurance Metrics And Tool Evaluation (SAMATE)** project is sponsored in part by DHS
- Began 2004 to help improve software assurance
- Current areas of concentration
  - Source code security analyzers
  - Studies of tool effectiveness
  - Web application scanners
  - *Binary analyzers*
  - *Software labels*

Source Code Security Analyzers

- Examine source code or binary for adherence to guidelines, weaknesses, etc.

- To assess tools, we wrote and collected thousands of test cases in a SAMATE Reference Dataset (SRD)
SRD: an Open Resource

Welcome to the NIST SAMATE Reference Dataset Project

The purpose of the SAMATE Reference Dataset (SRD) is to provide users, researchers, and software security assurance tool developers with a set of known security flaws. This will allow end users to evaluate tools and tool developers to test their methods. These test cases are designs, source code, binaries, etc., i.e. from all the phases of the software life cycle. The dataset includes "wild" (production), "synthetic" (written to test or generated), and "academic" (from students) test cases. This database will also contain real software application with known bugs and vulnerabilities. The dataset intends to encompass a wide variety of possible vulnerabilities, languages, platforms, and compilers. The dataset is anticipated to become a large-scale effort, gathering test cases from many contributors. We have more information about the SRD, including goals, structure, test suite selection, etc.

Browse, download, and search the SRD

Anyone can browse or search test cases and download selected cases. Please click here to browse the test case repository; or download selected or all test cases. To find specific test cases, please click here.

How to submit test cases

We welcome submission of software artifacts with security vulnerabilities. We also welcome samples of avoiding or mitigating such vulnerabilities. A test case consists of one or more files that manifests the security error, and metadata about the file(s), such as the platform,

6 September 2014

Paul E. Black
<table>
<thead>
<tr>
<th>ID</th>
<th>Date</th>
<th>Language</th>
<th>Category</th>
<th>Source</th>
<th>Security Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>58</td>
<td>2005-11-02</td>
<td>Java</td>
<td>Source Code</td>
<td>SecureSoftware</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>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>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>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>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>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>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>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>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>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>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);
            }
        }
    }
}
Outline

- SAMATE, my current program, briefly
- Some tough and bizarre bugs
- How many conditions do faults need?
- Combinatorial Testing
  - What is it?
  - Why does it work?
  - State of the Art
Wrong Date

- IBM System 3/10 - punched card input
- Daily boot-up set system date with a card
- In 1979 the daily run didn’t print any upcoming payment notices …
- Tiny program printed sys data as 1978!
- Checked date card, but it said 1979!
Bad Character

- **Application:** IC design parser
  - Input: computer chip design, 2D, WYSIWYG
  - Output: network list, plain text

- **Failure:** one strange character
  
  \[
  (V\text{-WIRE}\_32 \ (A\_[0..31] \ B\_[0..31])) \ (Y\_[0..31])
  \]
  \[
  ((G\_0 \ (Y\_0) \ T\text{-WIRE} \ (A\_0 \ B\_0)) \ (G\_1 \ (Y\_1) \ T\text{-WIRE} \ (A\_1 \ B\_1))
  \]
  \[
  (G\_8 \ (Y\_8) \ T\text{-GIRE} \ (A\_8 \ B\_8)) \ (G\_9 \ (Y\_9) \ T\text{-WIRE} \ (A\_9 \ B\_9))
  \]
  \[
  (G\_10 \ (Y\_10) \ T\text{-WIRE} \ (A\_10 \ B\_10)) \ (G\_11 \ (Y\_11) \ T\text{-WIRE} \ (A\_11 \ B\_11))
  \]

  - Could not reproduce on my machine; *could* on engineer’s
  - Different places or chars on other runs: memory overwrite?

- **No hint of code overwrite (common in C)**
- **Made a table of where the failure was in output file:** all had same low-order bits in hex

- **Conclusion:** flaky bit bit in output hardware
Lisp Error

- Circuit simulator driven by a built-in Lisp, a language with lists & garbage collection
- Designer had a bizarre error: some computations got wrong answers!
- I could reproduce that error, but it came and went and changed in similar cases
- In an attempt to track it down, I
  - Wrote a “shadow” floating point math module
  - Garbage collected after every operation
  - Changed the heap size
Lisp Error Solution

• Problem: wrong free list set up

• Fix: change one line of code
  
  h->next = free;

  to

  h->next = null;
Testing is like a seatbelt …
Outline

- SAMATE, my current program, briefly
- Some tough and bizarre bugs
- How many conditions do faults need?
- Combinatorial Testing
  - What is it?
  - Why does it work?
  - State of the Art
Software Failure Analysis

- NIST studied software failures in many fields including 15 years of FDA medical device recalls.
- Would pairwise testing find all errors?
- If not, then how many interactions would we need to test to find all errors?
- e.g., failure occurs if
  pressure < 10  \hspace{1cm} (1-way interaction)
  pressure < 10 & volume > 300 \hspace{1cm} (2-way interaction)
What interactions do we need to test to find ALL faults?

- Maximum interactions for fault triggering for these applications was 6
  - Wallace, Kuhn 2001 – medical devices
    98% of flaws were pairwise interactions, no fault required > 4-way interactions to trigger
  - Kuhn, Reilly 2002 – web server, browser;
    no fault required > 6-way interactions to trigger
  - Kuhn, Wallace, Gallo 2004 – large NASA distributed database;
    no fault required > 4 interactions to trigger

- Reasonable evidence that maximum interaction strength for fault triggering is relatively small
Maximum interactions for fault triggering for 4 domains

% triggered vs. Interactions for Medical, NASA, Server, and Browser domains.
Outline

- SAMATE, my current program, briefly
- Some tough and bizarre bugs
- How many conditions do faults need?

Combinatorial Testing
  - What is it?
  - Why does it work?
  - State of the Art
A simple example
How Many Tests Would It Take?

- There are 10 effects, each can be on or off
- All combinations is $2^{10} = 1,024$ tests
  too many to visually check …

- Let’s look at all 3-way interactions …
Now How Many Would It Take?

- There are $\binom{10}{3} = 120$ 3-way interactions.
- Naively $120 \times 2^3 = 960$ tests.
- Since we can pack 3 triples into each test, we need no more than 320 tests.
- Each test exercises many triples:
  
  $\begin{array}{cccccccc}
  0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\
  \end{array}$

  We oughta be able to pack a lot in one test, so what’s the smallest number we need?
All Triples Take Only 13 Tests
A Real-World Example

No silver bullet because:
Many values per variable
Need to abstract values
But we can still increase information per test

Plan: flt, flt+hotel, flt+hotel+car
From: CONUS, HI, AK, Europe, Asia …
To: CONUS, HI, AK, Europe, Asia …
Compare: yes, no
Date-type: exact, 1to3, flex
Depart: today, tomorrow, 1yr, Sun, Mon …
Return: today, tomorrow, 1yr, Sun, Mon …
Adults: 1, 2, 3, 4, 5, 6
Minors: 0, 1, 2, 3, 4, 5
Seniors: 0, 1, 2, 3, 4, 5
Does It Really Work?

- Traffic Collision Avoidance System (TCAS) module
  - Used in previous testing research
  - 41 versions seeded with errors
  - 12 variables: 7 boolean, two 3-value, one 4-value, two 10-value
  - All flaws found with 5-way coverage
Outline

- SAMATE, my current program, briefly
- Some tough and bizarre bugs
- How many conditions do faults need?
- Combinatorial Testing
  - What is it?
  - Why does it work?
  - State of the Art
Why? A Geometric Intuition

[Diagram showing geometric intuition with checkboxes for Superscript, Emboss, Subscript, and Hidden, with a focus on the term combinatorial testing]
Naïve Test Approach

- Test all off, all on, each one on
  - 7 tests total
How Combinatorial Tests Look
Outline

● SAMATE, my current program, briefly
● Some tough and bizarre bugs
● How many conditions do faults need?
● Combinatorial Testing
  – What is it?
  – Why does it work?
  – State of the Art
Combinatorial Testing Requires a Lot of Tests

- For $n$ variables with $\nu$ values each, the number of $k$-way combinations is:
  $$\binom{n}{k} \nu^k$$

- The test set is a covering array
- Finding a covering array is NP hard
- Assume 30 parameters with 5 values each. All 4-way combinations are covered by 3,800 tests
Lots of Tests (cont.)

<table>
<thead>
<tr>
<th>$k$</th>
<th># test cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-way</td>
<td>156</td>
</tr>
<tr>
<td>3-way</td>
<td>461</td>
</tr>
<tr>
<td>4-way</td>
<td>1,450</td>
</tr>
<tr>
<td>5-way</td>
<td>4,309</td>
</tr>
<tr>
<td>6-way</td>
<td>11,094</td>
</tr>
</tbody>
</table>

![Bar chart showing the number of test cases for different values of $k$.]
Combinatorial Testing Research

- Huge increases in performance & scalability
- Proof-of-concept demonstrations
- Applied modeling and simulation
Summary

- Combinatorial testing makes sense where
  - More than ~8 variables and less than 300 - 400
  - Logical or numeric interaction of variables
- New algorithms make large-scale combinatorial testing possible
- Beta release of open source tools in December
- New public catalog of covering arrays

http://csrc.nist.gov/acts
Seeking Participants

- Contribute test cases to **SRD**
- Comment on specifications and tests
- Join SAMATE email list with ideas on
  - Static binary analyzers
  - Software labels
- Use Combinatorial Test Generation Tools

Paul E. Black
SAMATE Project Leader
paul.black@nist.gov

Rick Kuhn
Combinatorial Testing
kuhn@nist.gov
Society has 3 options:

- Learn how to make software that **works**
- Limit size or authority of software
- Accept failing software