Search-Based Software Testing

Keynote:
Research, Challenges and Opportunities

Westley Weimer
University of Virginia
Meta

- I'm the sort of person who usually skips Keynotes.
- I'm not sure that I have a good argument for why you shouldn't skip mine.
- I'll try to focus on informal notions, background or ideas that may not be as common ... 
- ... rather than formal technical details.
Outline

• My Recent Research
  – Preview ICSE 2012

• The Field

• Challenges and Opportunities
Testing

- A way to gain confidence that a program implementation adheres to its specification (as refined from its requirements).
- Thus, testing often finds bugs ...
Roundabout Motivation

- "Don't come back till you have him!" the Ticktockman said, very quietly, very sincerely, extremely dangerously.

- They used dogs. They used probes. They used cardioplate crossoffs. They used teepers. They used bribery. They used stiktytes. They used intimidation. They used torment. They used torture. They used finks. They used cops. They used search and seizure. They used fallaron. They used betterment incentive. They used fingerprints. They used Bertillon. They used cunning. They used guile. They used treachery. They used Raoul Mitgong, but he didn't help much. They used applied physics. They used techniques of criminology.

- And what the hell: they caught him.

  - “Repent, Harlequin!” Said the Ticktockman (1965)
Catching Bugs

"Don't come back till you have him!" the Ticktockman said, very quietly, very sincerely, extremely dangerously.

They used dogs. They used probes. They used cardioplate crossoffs. They used teepers. They used bribery. They used stiktytes. They used intimidation. They used torment. They used torture. They used finks. They used cops. They used search & seizure. They used fallaron. They used betterment incentive. They used fingerprints. They used Bertillon. They used cunning. They used guile. They used treachery. They used Raoul Mitgong, but he didn't help much. They used applied physics. They used techniques of criminology.

And what the hell: they caught him.

“Repent, Harlequin!” Said the Ticktockman (1965)
A Big Issue: Now What?

Coyote catches Road Runner

Okay, wise guys, you always wanted me to catch him—Now what do I do?
Bug Bounties

Encouraging More Chromium Security Research
Thursday, January 26, 2016
Labels: googlechrome, security

In designing Chromium, we’ve been working hard to make the browser as secure as possible. We’ve made strong improvements with the Integrated Sandboxing and our up-to-date user base. We’re always looking to stay on top of the latest browser security features. We’ve also worked closely with the broader security community to get independent scrutiny and to quickly fix bugs that have been reported.

Some of the most interesting security bugs we’ve fixed have been reported by researchers external to the Chromium project. For example, this same origin policy bypass from Izzat Dawson or this v8 engine bug found by the Mozilla Security team. Thanks to the collaborative efforts of these people and others, Chromium security is stronger and our users are safer.

Today, we are introducing an experimental new incentive for external researchers to participate. We will be rewarding select interesting and original vulnerabilities reported to us by the security research community. For existing contributors to Chromium security—who would likely continue to contribute regardless—this may be seen as a token of our appreciation. In addition, we are hoping that the introduction of this program will encourage new individuals to participate in Chromium security. The more people involved in scrutinizing Chromium’s code and behavior, the more secure our millions of users will be.

Such a concept is not new: we’d like to give serious thanks to the folks at Mozilla for their long-running and successful vulnerability reward program.

Any valid security bug found through the Chromium bug tracker (under the label “Security Bug”) will qualify for consideration. As this is an experimental program, here are some guidelines in the form of questions and answers:

Q: What reward might I get?
   A: As per Mozilla, our base reward for the eligible bugs is $300. If the panel finds a particular bug particularly severe or particularly clever, we may enhance rewards of $1,237. The panel may also decide a single report actually constitutes multiple bugs. As a consumer of the Chromium open source project, Google will be sponsoring the rewards.

Q: What bugs are eligible?
   A: Any security bug may be considered. We will typically focus on High and Critical impact bugs, but any clever vulnerability or any severity might get a reward. Obviously, your bug won’t be eligible if you worked on the code or reviewed the area in question.

Q: How do I find out my bug was eligible?
   A: You will see a provisional comment to that effect in the bug entry once we have triaged the bug.

Q: What if someone also found the same bug?
   A: Only the first report of a given issue that we were previously unaware of is eligible. In the event of a duplicate submission, the earliest filed bug report in the bug tracker is considered the first report.

Q: What about bugs present in Google Chrome but not the Chromium open source project?
   A: Bugs in either build may be eligible. In addition, bugs in plugins that are part of Chromium are eligible.
Show Me The Money

Mozilla reserves the right to not give a bounty payment if we believe the actions of the reporter have endangered the security of Mozilla's end users.

If two or more people report the bug together the reward will be divided among them.

**Client Reward Guidelines**

The bounty for valid critical client security bugs will be $3000 (US) cash reward and a Mozilla T-shirt. The bounty will be awarded for **srg**critical and **s**g**h**igh severity security bugs that meet the following criteria:

- Security bug is present in the most recent supported, beta or release candidate version of Firefox, Thunderbird, Firefox Mobile, or in Mozilla services which could compromise users of those products, as released by Mozilla Corporation or Mozilla Messaging.
- Security bugs in or caused by additional 3rd-party software (e.g., plugins, extensions) are excluded from the Bug Bounty program.

More information about this program can be found in the **Client Security Bug Bounty Program FAQ**.

**Web Application and Services Reward Guidelines**

The bounty for valid web applications or services related security bugs, we are giving a range starting at $500 (US) for high severity and, in some cases, may pay up to $3000 (US) for extraordinary or critical vulnerabilities. We will also include a Mozilla T-shirt. The bounty will be awarded for **ws**critical and **w**s**h**igh security bugs that meet the following criteria:

- Security bug is present in the web properties outlined in the **Web Application Security Bounty FAQ**.
- Security bug is on the list of sites which part of the bounty. See the eligible bugs section of the **Web Application Security Bounty FAQ** for the list of sites which is included under the bounty.

More information about this program can be found in the **Web Application Security Bounty FAQ**.

**Process**

Please file a bug describing the security bug; be sure to check the box near the bottom of the entry form that marks this bug report as confidential. We encourage you to attach a "proof of concept" testcase or

---

<table>
<thead>
<tr>
<th>Bounty value</th>
<th>Pre-release bounty value</th>
<th>Type of bug</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1000</td>
<td>$2000</td>
<td>A bug which allows someone intercepting Tarsnap traffic to decrypt Tarsnap users' data.</td>
</tr>
<tr>
<td>$500</td>
<td>$1000</td>
<td>A bug which allows the Tarsnap service to decrypt Tarsnap users' data.</td>
</tr>
<tr>
<td>$500</td>
<td>$1000</td>
<td>A bug which causes data corruption or loss.</td>
</tr>
<tr>
<td>$100</td>
<td>$200</td>
<td>A bug which causes Tarsnap to crash (without corrupting data or losing any data other than an archive currently being written).</td>
</tr>
<tr>
<td>$50</td>
<td>$100</td>
<td>Any other non-harmless bugs in Tarsnap.</td>
</tr>
<tr>
<td>$20</td>
<td>$40</td>
<td>Build breakage on a platform where a previous Tarsnap release worked.</td>
</tr>
<tr>
<td>$10</td>
<td>$20</td>
<td>&quot;Harmless&quot; bugs, e.g., cosmetic errors in Tarsnap output or mistakes in source code comments.</td>
</tr>
<tr>
<td>$1</td>
<td>$2</td>
<td>Cosmetic errors in the Tarsnap source code or website, e.g., typos in website text or source code comments. Style errors in Tarsnap code qualify here, but usually not style errors in upstream code (e.g., libarchive).</td>
</tr>
</tbody>
</table>
Really?

- **Tarsnap:**
  - 200 candidates
  - 125 spelling/style
  - 63 “harmless”
  - 11 “minor”
  - 1 “major”

- $75/200 = 38\%$ TP rate

- $17 + 40$ hours per TP
GenProg: Automated Program Repair

- **Goal:** Automatically repair defects in off-the-shelf, legacy software.

- **Input:**
  - Unannotated Program Source
  - Deterministic Test Cases (1+ currently fail)

- **Output:**
  - Repaired Program (Patch)
  - That passes all Test Cases
See Also

- Clearview, Jolt
  - Automated fixing of deployed binaries
    - Perkins, Kim, Larsen, Amarasinghe, Bachrach, Carbin, Pacheco, Sherwood, Sidiroglou, Sullivan, Wong, Zibin, Ernst, Rinard

- AutoFix-E
  - Automated fixing of programs with contracts
    - Wei, Pei, Furia, Silva, Buchholz, Meyer, Nordio, Zeller

- AFix
  - Automated fixing of single variable atomicity violations
    - Jin, Song, Zhang, Lu, Liblit

- Debroy and Wong, etc.
GenProg Approach

- **Search** through the space of patches (sequences of edits) until one is found that passes all test cases.

- An “Edit” is:
  - Delete statement X
  - Insert statement X after statement Y
  - Replace statement X with statement Y

- The “Search” is:
  - Genetic Algorithm
Genetic Algorithm

- Genome = sequence of edits
- Mutate = add a new edit
- Crossover = uniform
- Fitness = weighted sum of tests passed
On The Shoulders of Giants

- **Search Space Reduction**
  - Fault Localization guides search
  - When mutating, apply edits to likely places

- **Test Suite Prioritization**
  - Evaluate fitness on a *sample* of the tests
  - Confirm a candidate repair via *retest-all*
  - Time-Aware Test Suite Prioritization
  - Impact Analysis
Parallelism

• Can evaluate in parallel
  - One candidate on test X and test Y
  - Separate candidates
  - Separate runs of the entire process

• Reduce time-to-first-fix
  - With many runs in parallel
  - Use public *cloud computing* infrastructure
  - Directly measure the cost ($) of each fix
Results that Generalize

- **Straw Program Repair Evaluation**
  - I took my 10 favorite programs, identified a bug in each one, and voila, our technique fixed all of those bugs.

- **Research Question**
  - What fraction of defects can these techniques actually repair?
Systematic Benchmark Selection

• Intuition: Take the last 100 bugs from project X. How many can we fix?

• Process for each Program X:
  - Take all tests from latest version of X.
  - Find all compile-able, run-able versions A,B such that B passes tests that A does not

• Consider top programs X from Sourceforge, Google Code, Fedora SRPM, etc.

• Fix all algorithm parameters before finding or inspecting benchmarks
From So Many To So Few

- Opaque or non-automated GUI testing
  - Firefox, Eclipse, OpenOffice
- Inaccessible or small version control histories
  - bash, cvs, openssh
- Few viable versions for recent tests
  - valgrind
- Require incompatible automake, libtool
  - gmp
- Non-deterministic tests ...
105 bugs, 5 MLOC, 10k Tests

**Subject C programs, test suites and historical defects:** Tests were taken from the most recent version available in May, 2011; Defects are defined as test case failures fixed by developers in previous versions.

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Tests</th>
<th>Defects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fbc</td>
<td>97,000</td>
<td>773</td>
<td>3</td>
<td>legacy coding</td>
</tr>
<tr>
<td>gmp</td>
<td>145,000</td>
<td>146</td>
<td>2</td>
<td>precision math</td>
</tr>
<tr>
<td>gzip</td>
<td>491,000</td>
<td>12</td>
<td>5</td>
<td>data compression</td>
</tr>
<tr>
<td>libtiff</td>
<td>77,000</td>
<td>78</td>
<td>24</td>
<td>image processing</td>
</tr>
<tr>
<td>lighttpd</td>
<td>62,000</td>
<td>295</td>
<td>9</td>
<td>web server</td>
</tr>
<tr>
<td>php</td>
<td>1,046,000</td>
<td>8,471</td>
<td>44</td>
<td>web programming</td>
</tr>
<tr>
<td>python</td>
<td>407,000</td>
<td>355</td>
<td>11</td>
<td>general coding</td>
</tr>
<tr>
<td>wireshark</td>
<td>2,814,000</td>
<td>63</td>
<td>7</td>
<td>packet analyzer</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td><strong>5,139,000</strong></td>
<td><strong>10,193</strong></td>
<td><strong>105</strong></td>
<td></td>
</tr>
</tbody>
</table>

- Bugs severe enough to merit checked-in tests
- Bugs 3/5 or higher Devel-reported severity
55/105 Repaired for $8 Each

- $403 for all 105 trials, leading to 55 repairs

<table>
<thead>
<tr>
<th>Program</th>
<th>Defects Repaired</th>
<th>Cost per Non-Repair</th>
<th>Cost Per Repair</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Hours US$</td>
<td>Hours US$</td>
</tr>
<tr>
<td>fbc</td>
<td>1 / 3</td>
<td>8.52</td>
<td>5.56</td>
</tr>
<tr>
<td>gmp</td>
<td>1 / 2</td>
<td>9.93</td>
<td>6.61</td>
</tr>
<tr>
<td>gzip</td>
<td>1 / 5</td>
<td>5.11</td>
<td>3.04</td>
</tr>
<tr>
<td>libtiff</td>
<td>17 / 24</td>
<td>7.81</td>
<td>5.04</td>
</tr>
<tr>
<td>lighttpd</td>
<td>5 / 9</td>
<td>10.79</td>
<td>7.25</td>
</tr>
<tr>
<td>php</td>
<td>28 / 44</td>
<td>13.00</td>
<td>8.80</td>
</tr>
<tr>
<td>python</td>
<td>1 / 11</td>
<td>13.00</td>
<td>8.80</td>
</tr>
<tr>
<td>wireshark</td>
<td>1 / 7</td>
<td>13.00</td>
<td>8.80</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td><strong>55 / 105</strong></td>
<td><strong>11.22h</strong></td>
<td><strong>1.60h</strong></td>
</tr>
</tbody>
</table>
Public Comparisons

- **JBoss issue tracking:**
  - Median 5.0 hours, mean 15.3 hours

- **Tarsnap.com**
  - $17 per non-trivial repair

- **IBM**
  - $25 per defect during coding
  - (rising at build, Q&A, post-release, etc.)

- **One of the php bugs we fixed has an associated security CVE**
A Systematic Study of Automated Program Repair:
Fixing 55 out of 105 bugs for $8 Each

<table>
<thead>
<tr>
<th>Program</th>
<th>Defects Repaired</th>
<th>Cost per Non-Repair Hours US$</th>
<th>Cost Per Repair Hours US$</th>
<th>LOC</th>
<th>Tests</th>
<th>Defects</th>
</tr>
</thead>
<tbody>
<tr>
<td>fbc</td>
<td>1 / 3</td>
<td>8.52</td>
<td>5.56</td>
<td>97,000</td>
<td>773</td>
<td>3</td>
</tr>
<tr>
<td>gmp</td>
<td>1 / 2</td>
<td>9.93</td>
<td>6.61</td>
<td>145,000</td>
<td>146</td>
<td>2</td>
</tr>
<tr>
<td>gzip</td>
<td>1 / 5</td>
<td>5.11</td>
<td>3.04</td>
<td>491,000</td>
<td>12</td>
<td>5</td>
</tr>
<tr>
<td>libtiff</td>
<td>17 / 24</td>
<td>7.81</td>
<td>5.04</td>
<td>77,000</td>
<td>78</td>
<td>24</td>
</tr>
<tr>
<td>lighttpd</td>
<td>5 / 9</td>
<td>10.79</td>
<td>7.25</td>
<td>62,000</td>
<td>295</td>
<td>9</td>
</tr>
<tr>
<td>php</td>
<td>28 / 44</td>
<td>13.00</td>
<td>8.80</td>
<td>1,046,000</td>
<td>8,471</td>
<td>44</td>
</tr>
<tr>
<td>python</td>
<td>1 / 11</td>
<td>13.00</td>
<td>8.80</td>
<td>407,000</td>
<td>355</td>
<td>11</td>
</tr>
<tr>
<td>wireshark</td>
<td>1 / 7</td>
<td>13.00</td>
<td>8.80</td>
<td>2,814,000</td>
<td>63</td>
<td>7</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td><strong>55 / 105</strong></td>
<td><strong>11.22h</strong></td>
<td><strong>1.60h</strong></td>
<td><strong>5,139,000</strong></td>
<td><strong>10,193</strong></td>
<td><strong>105</strong></td>
</tr>
</tbody>
</table>

Preprint, Code, Benchmarks, VMs
http://genprog.cs.virginia.edu
Outline

• My Recent Research
  – Preview ICSE 2012

• The Field

• Challenges and Opportunities
What is SBST?

- Search-Based Software Testing
- Well, “Software Testing” is well-established
- So what's “Search-Based”?
- Let's ask a luminary in the field.
Authoritative Answer
Are We Pining for the Fjords?

International Workshop on Search-Based Software Testing

Number of Accepted Papers

- 2008
- 2009
- 2010
- 2011
- 2012
SE Fields Using “Search-Based”

1976-2010 Percentage of Paper Number

- Testing and Debugging: 52%
- Management: 10%
- Design Tools and Techniques: 9%
- Distribution, Maintenance, and Enhancement: 9%
- Requirements/Specifications: 5%
- General Aspects: 5%
- Software/Program Verification: 3%
- Others: 7%

- Yuanyuan Zhang, SEBASE Repository
- 1022 SE+Search Publications
Search-Based Software Testing

- Is there still a strong case for a separate SBST identity (from SBSE and Software Testing)?
  - It's not clear to me that there is.
- Search-Based approaches are more respectable.
- There are good scientific and engineering challenges associated with making “mainstream” SE arguments.
Challenges and Opportunities

- Benchmark Selection
- Oracles and Specifications
- Extracting Human Intent
- Leveraging Cloud Computing & Crowdsourcing
- “Fix” Localization
- Embracing the Problem
- Lifting Input Assumptions & Sensitivity
Benchmark Selection

- The TRIANGLE program and the small SIEMENS benchmarks are not adequate.

- Augment them with real-world programs (e.g., open source), larger programs from SIR, or programs from other repositories (iBugs, etc.).

- Opportunity: If your algorithm really scales with the “tests” or “test coverage” and not the subject LOC, this is a free way for you to look more impressive while convincing a general SE audience.
Benchmark Selection

  - 350 papers and theses, 1977 - 2009
  - Table IX: Programs Used In Empirical Studies
  - Only 3 used a program of size 100,000 LOC
  - Median size < ~1000 LOC

- Mutation Testing != SBST, but ...
  - SBST Abs #: “LOC” 0, “lines” 0, “thousand” 0, “million” 0, etc. General SE will use LOC.

- Challenge yourself: +1 order of magnitude
Oracles and Specifications

- Test input generation is no longer sufficient. Test oracle generation is akin to specification mining and anomaly intrusion detection.
- There are now a number of great projects for test input generation (CUTE, AUSTIN, PEX, DART, etc.).
  - These work great with implicit or universal specifications (e.g., “don't segfault”).
- SBST Abs #: “oracle” 2, “specification” 3, “requirements” 1.
Specification Mining

- Ill-named task in SE/PL:
  - Given a program's source code and an indicative workload, output some partial correctness specifications.

- Analogy: Learn the rules of English from high school student essays.

- Reasonable post-DAIKON examples:
Human Intent

- Non-executable artifacts should play a larger role in search-based software testing.
- Imagine that you are trying to localize a fault, learn a specification, or prioritize a test suite, and you have two pieces of code:
  - One highly readable, rarely-touched, written by an expert developer, full of comments.
  - Another less readable, often churned, written by a novice, full of duplicate code.
Measuring Human Intent

- Halstead and Software Science may be poor choices, but other options are available.


- SBST Abs #: docume* 0, comment 0, human 2.

- Measuring intent or quality involves humans but is much cheaper than a full-blown human study: instead, sift historical data to get human judgments “for free”.
Cloud Computing

- Cloud computing (or similar) should be used to put a monetary value on research costs or benefits, where applicable.
- For some tasks (test suite reduction or prioritization) this may not be necessary, but for others (fault localization, bug repair) this seems increasingly relevant.
  - Many layers to “total effort saved”
  - Challenge yourself: be precise about one.
- SBST Abs #: cloud 0, dollar 0, money 0, cost 8
Crowdsourcing

- Crowdsourcing should be used as an efficient way to gather human study data.
- Services like Amazon's Mechanical Turk are increasingly used in SE human studies.
- Care must be taken to control or account for expertise and “gaming the system”.
- SBST Abs #: “human s” 0, “irb” 0, “crowd” 0, human 2.
“Fix” Localization

• **We should use tests to detect possible fixes, not just to detect faults.**

• Explicit personal bias on this one.

• Information used for test prioritization, test reduction, test generation, etc., could be used to say “If you are failing test X, you should look at Y” more often than it currently is.

• SBST Abs #: fix 1!, locali* 0, repair 2.
Input Assumptions & Sensitivity

- We should try to reduce (or measure the impact of) the assumptions of our projects.

- If you are a grad student looking for a project, removing an assumption made by another project is a reasonable first step.
  - Example: “Assumes single-threaded code” or “Assumes tracing from requirements to tests.” Increases utility, carves out niche.

- If not: Sensitivity Analysis. (Abs #: sens.* 0)
  - Example: If the tracing is off by 10%, is the reduction off by 20%? 2000%?
Conclusion

- **Automated Program Repair**
  - We can repair 55/105 bugs in 5 MLOC with 10,000+ tests for $8 each, on average.

- **Search-Based Software Testing venue**
  - Time to go mainstream?

- **Challenges and Opportunities**
  - Benchmarks, Specifications, Intent, Cloud Computing, Crowdsourcing, Fix Localization, Assumptions and Sensitivity