A SYSTEMATIC STUDY OF AUTOMATED PROGRAM REPAIR: FIXING 55 OUT OF 105 BUGS FOR $8 EACH

Claire Le Goues
Michael Dewey-Vogt
Stephanie Forrest
Westley Weimer

http://genprog.cs.virginia.edu
“Everyday, almost 300 bugs appear […] far too many for only the Mozilla programmers to handle.”

– Mozilla Developer, 2005

Annual cost of software errors in the US: $59.5 billion (0.6% of GDP).

PROBLEM: BUGGY SOFTWARE

Average time to fix a security-critical error: 28 days.

10%: Everything Else

90%: Maintenance
HOW BAD IS IT?
Bug Bounty Program

Introduction

The Mozilla Security Bug Bounty Program is designed to encourage security research in Mozilla software and to reward those who help us create the safest Internet clients in existence.

Many thanks to Linus and Mark Shuttleworth, who provided start-up funding for this endeavor.

General Bounty Guidelines

Tarsnap

Online backups for the truly paranoid

Tarsnap Bug Bounties

According to Linus's Law, "given enough eyeballs, all bugs are shallow." This is one of the reasons why the Tarsnap client source code is available; but merely making the source code available doesn't do anything if people don't bother to read it.

For this reason, Tarsnap has a series of bug bounties. Similar bounties offered by Mozilla and Google, the Tarsnap bug bounty is an opportunity for people who find bugs to win cash. Unlike those of the Tarsnap bug bounties aren't limited to security bugs. Deen

Encouraging More Chromium Security Research

Thursday, January 28, 2010

Labels: google, chromium, 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 Isaac Dawson of this week's 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 credit where it's due to the folks at Mozilla for their long-running and successful vulnerability reward program.

Any valid security bug filed through the Chromium bug tracker (under the template "Security Bug") will qualify for consideration. As this is an experimental program, there are some guidelines in the form of questions and answers.

Q) What reward might I get?
A) As per Mozilla, our base reward for eligible bugs is $500. If the panel finds a particular bug particularly severe or particularly clever, we may award $1,000. The panel may also decide to split the award into multiple parts. 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 at 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 else 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.
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. The bounty will be awarded for sg:critical and sg:high severity security bugs that meet the following criteria:

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

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

Web Application and Services Reward Guidelines

The bounty for valid web applications or services related security bugs, we are giving $500 (US) for high severity and, in some cases, may pay up to $3000 (US) for extreme vulnerabilities. We will also include a Mozilla T-shirt. The bounty will be awarded for ws:high 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 Application Security Bounty FAQ for the list of sites which is included under eligibility.

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

<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-harmful 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>

The pre-release bounty value will be awarded for bugs reported in the interval between when a new Tarsnap release is sent to the tarsnap
...REALLY?

Tarsnap:
125 spelling/style
63 harmless
11 minor
+ 1 major

75/200 = 38% TP rate
$17 + 40 hours per TP

which were wrong yet didn’t actually affect the compiled code.

But most importantly, $1265 of bugs gives me the peace of mind of knowing that I’m not the only person who has looked at the Tarsnap code, and if there are more critical bugs like the security bug I fixed in January, they’ve escaped more than just my eyeballs. Worth the money? Every penny.
Tarsnap:
125 spelling/style
63 harmless
11 minor
+ 1 major

75/200 = 38% TP rate
$17 + 40 hours per TP

which were wrong yet didn’t actually affect the compiled code.

But most importantly, $1265 of bugs gives me the peace of mind of knowing that I’m not the only person who has looked at the Tarsnap code, and if there are more critical bugs like the security bug I fixed in January, they’ve escaped more than just my eyeballs. Worth the money? Every penny.

http://genprog.cs.virginia.edu
...REALLY?

which were wrong yet didn't actually affect the compiled code.

But most importantly, $1265 of bugs gives me the peace of mind of knowing that I'm not the only person who has looked at the Tarsnap code, and if there are more critical bugs like the security bug I fixed in January, they've escaped more than just my eyeballs. Worth the money? Every penny.
SOLUTION: PAY STRANGERS
SOLUTION:
PAY STRANGERS
SOLUTION: AUTOMATE
GENPROG: Automatic¹, Scalable, Competitive Bug Repair.


GENPROG: AUTOMATIC\textsuperscript{1}, SCALABLE, COMPETITIVE BUG REPAIR.

AUTOMATED PROGRAM REPAIR

GENPROG: AUTOMATIC, SCALABLE, COMPETITIVE BUG REPAIR.

http://genprog.cs.virginia.edu
AUTOMATED PROGRAM REPAIR

GENPROG: AUTOMATIC, SCALABLE, COMPETITIVE BUG REPAIR.

http://genprog.cs.virginia.edu
AUTOMATED PROGRAM REPAIR

GENPROG: AUTOMATIC, SCALABLE, COMPETITIVE BUG REPAIR.

http://genprog.cs.virginia.edu
Search: random (GP) search through nearby patches.

Approach: compose small random edits.

• Where to change?
• How to change it?
Input:

1

2

3

4

5

6

7

8

9

10

11

12
Input: 🟢🟢🟢❌

Legend:
- High change probability.
- Low change probability.
- Not changed.
An edit is:

- Replace statement X with statement Y
- Insert statement X after statement Y
- Delete statement X
An edit is:

- Replace statement X with statement Y
- Insert statement X after statement Y
- Delete statement X
An *edit* is:

- Replace statement X with statement Y
- Insert statement X after statement Y
- Delete statement X
An edit is:

- Replace statement X with statement Y
- Insert statement X after statement Y
- Delete statement X
An edit is:

- Replace statement X with statement Y
- Insert statement X after statement Y
- Delete statement X
An edit is:

- Replace statement X with statement Y
- Insert statement X after statement Y
- Delete statement X
An edit is:

- Replace statement X with statement Y
- Insert statement X after statement Y
- Delete statement X
An edit is:

- Replace statement X with statement Y
- Insert statement X after statement Y
- Delete statement X
AUTOMATED PROGRAM REPAIR

GENPROG: AUTOMATIC, SCALABLE, COMPETITIVE BUG REPAIR.
AUTOMATED PROGRAM REPAIR

GENPROG: AUTOMATIC, SCALABLE, COMPETITIVE BUG REPAIR.
SCALABLE: SEARCH SPACE
SCALABLE: SEARCH SPACE

Diagram showing a search space with nodes and edges.
SCALABLE: SEARCH SPACE
SCALABLE: SEARCH SPACE

Fix localization: intelligently choose code to move.
SCALABLE: REPRESENTATION

**Input:**

- Node 1
- Node 2
- Node 3
- Node 4
- Node 5

**Naïve:**

1. Node 1
2. Node 2
3. Node 4
4. Node 5

**New:**

1. Node 1
2. Node 2
3. Node 5’
4. Node 4
5. Node 5

- Delete(3)
- Replace(3, 5)

Claire Le Goues, ICSE 2012
http://genprog.cs.virginia.edu
New fitness, crossover, and mutation operators to work with a variable-length genome.

Naïve:

New:

Delete(3)

Replace(3,5)
Fitness:

- Subsample test cases.
- Evaluate in parallel.

Random runs:

- Multiple simultaneous runs on different seeds.
GENPROG: AUTOMATIC, SCALABLE, COMPETITIVE BUG REPAIR.
AUTOMATED PROGRAM REPAIR

GENPROG: AUTOMATIC, SCALABLE, COMPETITIVE BUG REPAIR.
How many bugs can GenProg fix?

COMPETITIVE

How much does it cost?
Goal: systematically test GenProg on a general, indicative bug set.

General approach:

• Avoid overfitting: fix the algorithm.
• Systematically create a generalizable benchmark set.
• Try to repair every bug in the benchmark set, establish grounded cost measurements.
Goal: systematically evaluate GenProg on a general, indicative bug set.

General approach:

• Avoid overfitting: fix the algorithm.
• Systematically create a generalizable benchmark set.
• Try to repair every bug in the benchmark set, establish grounded cost measurements.
CHALLENGE: INDICATIVE BUG SET
Goal: a large set of important, reproducible bugs in non-trivial programs.

Approach: use historical data to approximate discovery and repair of bugs in the wild.
Consider top programs from SourceForge, Google Code, Fedora SRPM, etc:

• Find pairs of viable versions where test case behavior changes.
• Take all tests from most recent version.
• Go back in time through the source control.

Corresponds to a human-written repair for the bug tested by the failing test case(s).
# BENCHMARKS

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Tests</th>
<th>Bugs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fbc</td>
<td>97,000</td>
<td>773</td>
<td>3</td>
<td>Language (legacy)</td>
</tr>
<tr>
<td>gmp</td>
<td>145,000</td>
<td>146</td>
<td>2</td>
<td>Multiple 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 manipulation</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>Language (web)</td>
</tr>
<tr>
<td>python</td>
<td>407,000</td>
<td>355</td>
<td>11</td>
<td>Language (general)</td>
</tr>
<tr>
<td>wireshark</td>
<td>2,814,000</td>
<td>63</td>
<td>7</td>
<td>Network 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>
CHALLENGE: GROUNDED COST MEASUREMENTS
GO
13 HOURS LATER
## SUCCESS/COST

<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</td>
<td>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></td>
</tr>
</tbody>
</table>

$403 for all 105 trials, leading to 55 repairs; $7.32 per bug repaired.
PUBLIC COMPARISON

JBoss issue tracking: median 5.0, mean 15.3 hours.¹

IBM: $25 per defect during coding, rising at build, Q&A, post-release, etc.²

Tarsnap.com: $17, 40 hours per non-trivial repair.³

Bug bounty programs in general:

- At least $500 for security-critical bugs.
- One of our php bugs has an associated security CVE.


³http://www.tarsnap.com/bugbounty.html
CONCLUSIONS/CONTRIBUTIONS

GenProg: scalable, automatic bug repair.

• Algorithmic improvements for scalability: fix localization, internal representation, parallelism.

Systematic study:

• Indicative, systematically-generated set of bugs that humans care about.
• Repaired 52% of 105 bugs in 96 minutes, on average, for $7.32 each.

Benchmarks/results/source code/VM images available:

• http://genprog.cs.virginia.edu
I LOVE QUESTIONS.

(Examples: “Which bugs can GenProg fix?” “What happens if you run for more than 13 hours/change the probability distributions/pick a different crossover/etc?” “How do you know the patches are any good?” “How do your patches compare to human patches?” ...
WHICH BUGS...?

Slightly more likely to fix bugs where the human:

- restricts the repair to statements.
- touched fewer files.

As **fault space** decreases, success increases, repair time decreases.

As **fix space** increases, repair time decreases.
FINDING BUGS IS HARD

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
  • Earlier versions of gmp

No bugs
  • GnuCash, openssl

Non-deterministic tests ...
EXAMPLE: PHP BUG #54372

Relevant code: function zend_std_read_property in zend_object_handlers.c

Note: memory management uses reference counting.

Problem: this line:

449. zval_ptr_dtor(object)

If object points to $this and $this is global, its memory is completely freed, even though we could access $this later.

1. class test_class {
2.  public function __get($n) {
3.      return $this; %$
4.  }
5.  public function b() {
6.      return;
7.  }
8.  }
9.  global $test3;
10. $test3 = new test_class();
11. $test3->a->b();

Expected output: nothing

Buggy output: crash on line 9.
EXAMPLE: PHP BUG #54372

Human:

```php
% 449c449,453
< zval_ptr_dtor(&object);
> if (*retval != object)
> { // expected
>    zval_ptr_dtor(&object);
> } else {
>    Z_DELREF_P(object);
> }
```

GenProg:

```c
% 448c448,451
> Z_ADDROF_P(object);
> if (PZVAL_IS_REF(object))
> {
>    SEPARATE_ZVAL(&object);
> }
> zval_ptr_dtor(&object)
```
Is automatically-patched code more or less maintainable?

Approach: Ask 102 humans maintainability questions about patched code (human vs. GenProg).

Results:

- No difference in accuracy/time between human accepted and GenProg patches.
- Automatically-documented GenProg patches result in higher accuracy and lower effort than human patches.

## PATCH REPRESENTATION

<table>
<thead>
<tr>
<th>Program</th>
<th>Fault</th>
<th>LOC</th>
<th>Repair Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcd</td>
<td>infinite loop</td>
<td>22</td>
<td>1.07</td>
</tr>
<tr>
<td>uniq-utx</td>
<td>segfault</td>
<td>1146</td>
<td>1.01</td>
</tr>
<tr>
<td>look-utx</td>
<td>segfault</td>
<td>1169</td>
<td>1.00</td>
</tr>
<tr>
<td>look-svr</td>
<td>infinite loop</td>
<td>1363</td>
<td>1.00</td>
</tr>
<tr>
<td>units-svr</td>
<td>segfault</td>
<td>1504</td>
<td>3.13</td>
</tr>
<tr>
<td>deroff-utx</td>
<td>segfault</td>
<td>2236</td>
<td>1.22</td>
</tr>
<tr>
<td>nullhttpd</td>
<td>buffer exploit</td>
<td>5575</td>
<td>1.95</td>
</tr>
<tr>
<td>indent</td>
<td>infinite loop</td>
<td>9906</td>
<td>1.70</td>
</tr>
<tr>
<td>flex</td>
<td>segfault</td>
<td>18775</td>
<td>3.75</td>
</tr>
<tr>
<td>atris</td>
<td>buffer exploit</td>
<td>21553</td>
<td>0.97</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td>6325</td>
<td>1.68</td>
</tr>
</tbody>
</table>