From Runtime Failures to Patches: Study of Patch Generation in Production

Thomas Durieux

June 12, 2019

Defended the 25 September 2018 at Lille
Advisors: Martin Monperrus & Lionel Seinturier
Thesis initiator: Youssef Hamadi Ecole Polytechnique
Partnership between INRIA & Microsoft Research
Chromium is taking on average 48 days for handling blocking issues\(^1\)

\(^1\)Valdivia Garcia and Shihab, “Characterizing and predicting blocking bugs in open source projects”, *MSR’14*
Automatic Patch Generation
Automatic Patch Generation

Buggy Application  Repair Strategy  Oracle (e.g., Crash)

---

Test-based Automatic Patch Generation

Buggy Program

GenProg\(^3\), Nopol\(^4\), CapGen\(^5\), ...

Regression Oracle:
- Green circle: Passing Tests
- Red circle: Failing Tests

---

\(^3\)Le Goues et al., “GenProg: A generic method for automatic software repair”, *TSE’12*

\(^4\)Xuan et al., “Nopol: Automatic repair of conditional statement bugs in Java programs”, *TSE’16*

\(^5\)Wen et al., “Context-Aware Patch Generation for Better Automated Program Repair”, *ICSE’18*
Test-based Automatic Patch Generation

Uses the test suite as the specification of the program.

<table>
<thead>
<tr>
<th>Status</th>
<th>Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Test Feature 1</td>
</tr>
<tr>
<td></td>
<td>Test Feature 2</td>
</tr>
<tr>
<td></td>
<td>Test Feature 3</td>
</tr>
</tbody>
</table>
Test-based Automatic Patch Generation

Uses the test suite as the specification of the program.

**Common practice:** Developer reproduces a bug with a test

<table>
<thead>
<tr>
<th>Status</th>
<th>Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Test Feature 1</td>
</tr>
<tr>
<td></td>
<td>Test Feature 2</td>
</tr>
<tr>
<td></td>
<td>Test Feature 3</td>
</tr>
<tr>
<td></td>
<td><strong>Reproduced Bug-X</strong></td>
</tr>
</tbody>
</table>
Test-based Automatic Patch Generation

Uses the test suite as the specification of the program.

**Goal:** Patch generation techniques make all the tests passing

<table>
<thead>
<tr>
<th>Status</th>
<th>Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Test Feature 1</td>
</tr>
<tr>
<td></td>
<td>Test Feature 2</td>
</tr>
<tr>
<td></td>
<td>Test Feature 3</td>
</tr>
<tr>
<td></td>
<td><strong>Reproduced Bug-X</strong></td>
</tr>
</tbody>
</table>
Problem 1: Automatic patch generation techniques rely on a failing test-case to reproduce the bug.
Solution 1: To connect the automatic patch generation techniques to the production environment where real bugs happen on a daily basis.
Demo

Error in the field.\textsuperscript{6}

\textsuperscript{6}Screencast: durieux.me/bikiniproxy.mp4
Outline

Automatic Patch Generation

BikiniProxy: Patch Generation for JavaScript Client-side applications

   BikiniProxy Architecture

   BikiniProxy Evaluation

Itzal: Patch Generation for Server-side Applications

   Itzal Architecture

   Itzal Evaluation

Conclusion
BikiniProxy is a HTTP proxy that handles JavaScript errors by rewriting the JavaScript and HTML HTTP requests.
Browser: e.g. Firefox or Chrome
Web server: traditional HTTP server
BikiniProxy – Architecture

User → Browser → Web Server

- image.png
- style.css
- script.js
- page.html

page.html: web resource.
JS Error: JS error faced by the user in the browser.
**Proxy:** BikiniProxy that handles failures by rewriting the resources.
rewritten-page.html: web page with BikiniProxy framework.
**BikiniProxy Backend**: stores the errors faced by the User.

**Goal**: Collect JavaScript errors.
**BikiniProxy – Architecture**

**BikiniProxy Backend**: Send the known errors for a given page.

**patched-* .js**: web resource’s rewritten by BikiniProxy.

**Goal**: Handle the known errors.
BikiniProxy – Repair Strategies

JavaScript Strategies

1. **HTTP/HTTPS Redirector** changes HTTP to HTTPS
2. **HTML Element Creator** creates HTML elements
3. **Library Injector** injects missing libraries

Generic Strategies

4. **Line Skipper** adds a precondition to the buggy statement
5. **Initialize Variable** initializes a null variable
Evaluation Protocol

1. Create a benchmark of JavaScript production errors
2. Evaluate BikiniProxy with the benchmark
DeadClick: a Benchmark of JavaScript Errors

<table>
<thead>
<tr>
<th>Crawling statistics</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td># Visited pages</td>
<td>96174</td>
</tr>
<tr>
<td># Pages with errors</td>
<td>4282 (4.5%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benchmark statistics</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td># Pages with reproduced errors</td>
<td>555</td>
</tr>
<tr>
<td># Errors</td>
<td>826</td>
</tr>
<tr>
<td># Errors per page</td>
<td>1-10 (avg. 1.49)</td>
</tr>
<tr>
<td>Average page size</td>
<td>1.98mb</td>
</tr>
</tbody>
</table>

DeadClick is the first benchmark of reproducible JavaScript errors.
1. Access each web page of DeadClick with BikiniProxy enabled
2. Collect the triggered errors
3. Compare the errors with the DeadClick errors
## BikiniProxy – Evaluation Results

<table>
<thead>
<tr>
<th>Error Type</th>
<th># handled error</th>
</tr>
</thead>
<tbody>
<tr>
<td>53 error types</td>
<td>248/826 (30%)</td>
</tr>
<tr>
<td>xxx is not defined</td>
<td>184/307 (60%)</td>
</tr>
<tr>
<td>Cannot read property xxx of null</td>
<td>42/176 (24%)</td>
</tr>
<tr>
<td>xxx is not a function</td>
<td>11/111 (10%)</td>
</tr>
<tr>
<td>Unexpected token x</td>
<td>2/61 (3%)</td>
</tr>
<tr>
<td>Cannot set property xxx of null</td>
<td>11/24 (46%)</td>
</tr>
<tr>
<td>Invalid or unexpected token</td>
<td>0/21 (0%)</td>
</tr>
<tr>
<td>Unexpected identifier</td>
<td>0/15 (0%)</td>
</tr>
<tr>
<td>Script error for: xxx</td>
<td>2/10 (20%)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

BikiniProxy is able to handle 30% of the errors.
BikiniProxy – Conclusion

BikiniProxy has been presented at ISSRE’18 and gas been nominated for the best paper award.

Key Novelties

• First proxy-based repair technique
• New repair strategies for JavaScript errors
• First benchmark of JavaScript field errors
Problem 2: Automatic generated patches can alter the state of the applications.
Solution 2: To shadow the production application in a sandboxed environment for patch generation techniques.
Outline

Automatic Patch Generation

BikiniProxy: Patch Generation for JavaScript Client-side applications
  BikiniProxy Architecture
  BikiniProxy Evaluation

Itzal: Patch Generation for Server-side Applications
  Itzal Architecture
  Itzal Evaluation

Conclusion
Client: e.g. a browser
Server: e.g. a web server
**Shadower**: intercepts and duplicates the requests
Patch Service: generates patches that fix the requests
Failure Oracle: detects if a request is passing or failing
**Itzal – Architecture**

---

**Client**

- **Shadower**

**Server**

**Failure Oracle**

**Patch Service**

**Sandbox**

**Regression Service**

**Regression Oracle**

---

**Regression**: executes passing request on patched server

**Regression Oracle**: compares the output of the original server and the patched server
**Reporting**: communicates the patches to the developers (Dashboard, Pull Request, ...)

---

**Shadower**

**Server**

**Failure Oracle**

**Patch Service**

**Sandbox**

**Regression Service**

**Regression Oracle**

**Reporting**

**Developer**
Itzal – Evaluations

Three evaluations:

Evaluation 1: Patch Generation Service
   - Assert that the **patch generation service** can generate patches from a failing execution.

Evaluation 2: Regression Service
   - Assert that the **regression service** can detect behavior changes between a valid and an invalid patch.

Evaluation 3: Itzal Architecture
   - Assert that all the services of Itzal work together by evaluating it with two **cases studies**.
**Goal**: Assert that the **patch generation service** can generate patches from a failing execution.

1. Collect 34 null pointer exception bugs from six benchmarks
2. Repair the bugs with NPEFix and Exception-Stopper
3. Verify that the generated patches handle the buggy request
## Itzal – Evaluation 1 Results

<table>
<thead>
<tr>
<th>Repair Strategies</th>
<th>NPEFix</th>
<th>Exception-Stopper</th>
</tr>
</thead>
<tbody>
<tr>
<td># Valid</td>
<td>23 118</td>
<td>198</td>
</tr>
<tr>
<td># Invalid</td>
<td>31 060</td>
<td>592</td>
</tr>
</tbody>
</table>

34 bugs from 14 applications

NPEFix and Exception-Stopper can generate patches from a failing request.
**Goal**: Assert that the **regression service** can detect invalid patches.

1. Take two e-commerce applications
2. Inject bugs in the e-commerce applications
3. Generate patches with NPEFix
4. Create synthetical production traffic for the e-commerce applications
5. Compare the regression oracles effectiveness to detect behavior change in the applications
Visual behavior:

- **HTTP Status** $HTTP_{status} \neq 5xx$
- **HTTP Content** $Response_{patched} == Response_{original}$

Program behavior:

- **Execution trace at method level**
  $Method_{patched} \sim Method_{original}$
- **Execution trace at block level**
  $Block_{patched} \sim Block_{original}$
Regression oracles can detect behavior changes by observing the application behavior.
Goal: Assert that all the services of Itzal work together by evaluating it with two cases studies.

1. Find null pointer exceptions in e-commerce applications
2. Identify the workflow to reproduce the bugs
3. Setup the application in Itzal architecture
4. Replay the buggy requests and synthetical requests
5. Collect the generated patches
NPE in cart when no price is defined in shipping strategy

#231

jvelo opened this issue on Dec 9, 2014 · 0 comments

jvelo commented on Dec 9, 2014

No description provided.

jvelo added the bug label on Dec 9, 2014

jvelo self-assigned this on Dec 9, 2014

jvelo added a commit that closed this issue on May 29, 2015

- Fixes #231 NPE in cart when no price is defined in shipping strategy

jvelo closed this in ce04282 on May 29, 2015

Assignees
- jvelo

Labels
- bug

Projects
- None yet

Milestone
- No milestone

Notifications
You’re not receiving notifications from
Itzal – Evaluation 3 Architecture

![Architecture Diagram]

- **Workload**
- **Mayocat**
- **Shadower**
- **Patch Service**

HTTP$_{status} \neq 5xx$
### Itzal – Evaluation 3 Results

<table>
<thead>
<tr>
<th>Repair Strategy</th>
<th># Valid</th>
<th># Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPEFix</td>
<td>105</td>
<td>182</td>
</tr>
</tbody>
</table>

Valid generated patch by Itzal for Mayocat

```java
@@ FlatStrategyPriceCalculator.java
@@ -37,2 +37,5 @@
+ if (carrier.getPerItem() == null) {
+     return null;
+ } }

    price = price.add(carrier.getPerItem().multiply(BigDecimal.valueOf(numberOfItems)));
```
Itzal – Conclusion

Itzal has been presented at ICSE NIER’17.

Key Novelties

- Patch generation in production.
- Patch regression with production inputs.
- Shadowing the production environment to a repair environment to not introduce regression in the application.
Outline

Automatic Patch Generation

BikiniProxy: Patch Generation for JavaScript Client-side applications
  BikiniProxy Architecture
  BikiniProxy Evaluation

Itzal: Patch Generation for Server-side Applications
  Itzal Architecture
  Itzal Evaluation

Conclusion
Conclusion

Automatic patch generation in production is feasible with:

- BikiniProxy: a patch generation technique for JavaScript client-side applications
- Itzal: a patch generation architecture for server-side applications
All the presented artifacts are open-science. They are available on GitHub:

https://github.com/spirals-team/
Summary

Test-based Automatic Patch Generation

Buggy Program

GenProg\textsuperscript{3}, Nopol\textsuperscript{4}, CapGen\textsuperscript{5}, ...

Regression Oracle:
- Passing Tests
- Failing Tests

BikiniProxy – Architecture

BikiniProxy Backend: Send the known errors for a given page.

patched-*.js: web resource’s rewritten by BikiniProxy.

Goal: Handle the known errors.

Conclusion

This thesis is the first work to show that automatic patch
generation in production is feasible with:

- BikiniProxy: a patch generation technique for JavaScript
  client-side applications
- Itzal: a patch generation architecture for server-side
  applications

\textsuperscript{3}Le Goues et al., “GenProg: A generic method for automatic software repair”, TSE’12
\textsuperscript{4}Xuan et al., “Nopol: Automatic repair of conditional statement bugs in Java
  programs”, TSE’16
\textsuperscript{5}Wen et al., “Context-Aware Patch Generation for Better Automated Program
  Repair”, ICSE’18

Itzal – Architecture

Reporting: communicates the patches to the developers
(Dashboard, Pull Request, ...)

BikiniProxy – Architecture

BikiniProxy Backend

Proxy

image.png
style.css
patched-script.js
rewritten-page.html

User

Browser

Web Server

Patch Service

Sandbox

Regression Service

Regression Oracle

Server

Client

Shadower

Patch Service

Sandbox

Regression Service

Regression Oracle

Developer

Reporting

rewritten-page.html

BikiniProxy Backend

User

Browser

Web Server