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

Thomas Durieux

June 12, 2019

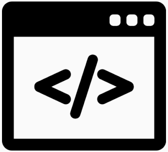
Defended the 25 September 2018 at Lille
Advisors: Martin Monperrus & Lionel Seinturier
Thesis initiator: Youssef Hamadi *Ecole Polytechnique*
Partnership between *INRIA & Microsoft Research*

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*

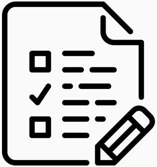
Automatic Patch Generation

Automatic Patch Generation[2]



Buggy Application

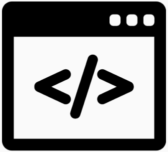
Repair Strategy



Oracle (e.g. Crash)

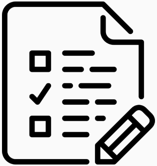
[2]Monperrus, "Automatic software repair: a bibliography", *CSUR'18*.

Test-based Automatic Patch Generation



Buggy Program

GenProg[3], Nopol[4], CapGen[5], ...



Regression Oracle:

● Passing Tests

Failure Oracle:

● 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.

Status	Tests
●	Test Feature 1
●	Test Feature 2
●	Test Feature 3

Test-based Automatic Patch Generation

Uses the test suite as the specification of the program.

Common practice: Developer reproduces a bug with a test

Status	Tests
●	Test Feature 1
●	Test Feature 2
●	Test Feature 3
●	**Reproduced Bug-X**

Test-based Automatic Patch Generation

Uses the test suite as the specification of the program.

Goal: Patch generation techniques make all the tests passing

Status	Tests
●	Test Feature 1
●	Test Feature 2
●	Test Feature 3
●	**Reproduced Bug-X**

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.

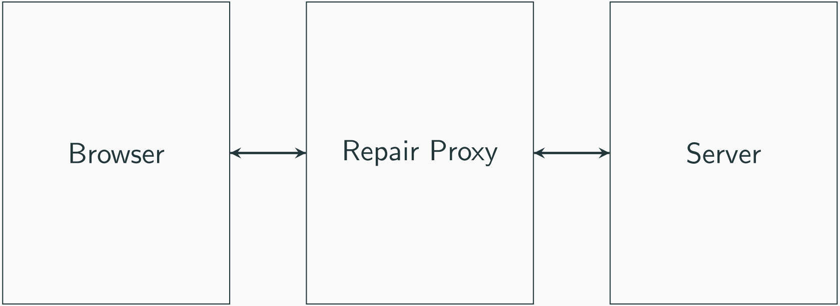
Error in the field.[6]

[6] Screencast: `durieux.me/bikiniproxy.mp4`

Outline

BikiniProxy

BikiniProxy is a HTTP proxy that handles JavaScript errors by rewriting the JavaScript and HTML HTTP requests.

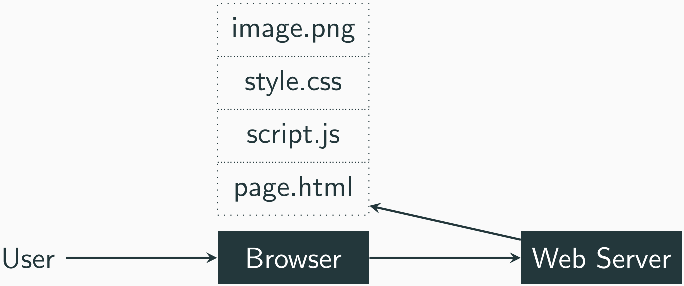


BikiniProxy – Architecture

Browser: e.g. Firefox or Chrome

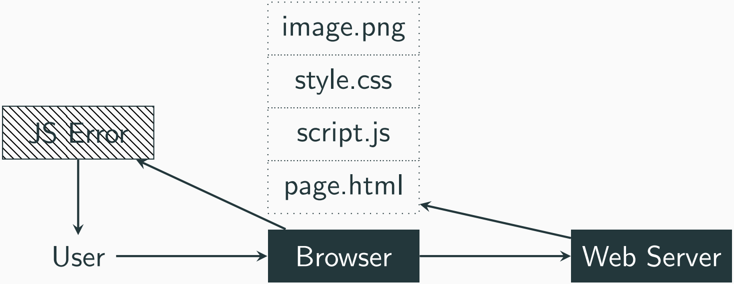
Web server: traditional HTTP server

BikiniProxy – Architecture



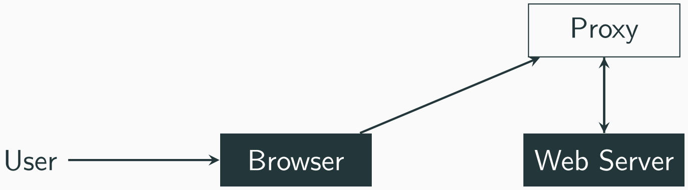
page.html: web resource.

BikiniProxy – Architecture



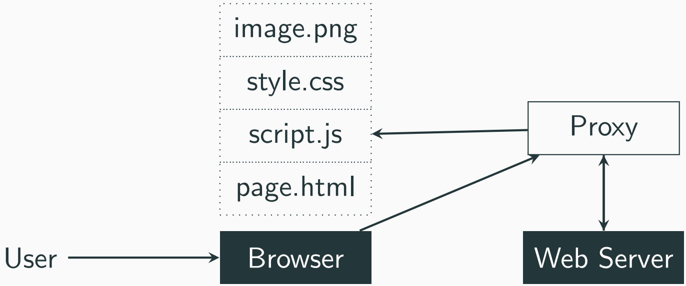
JS Error: JS error faced by the user in the browser.

BikiniProxy – Architecture

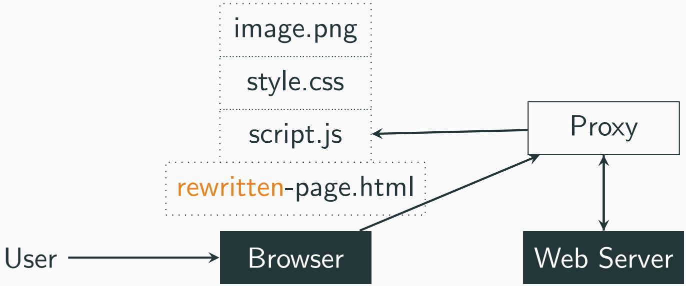


Proxy: BikiniProxy that handles failures by rewriting the resources.

BikiniProxy – Architecture

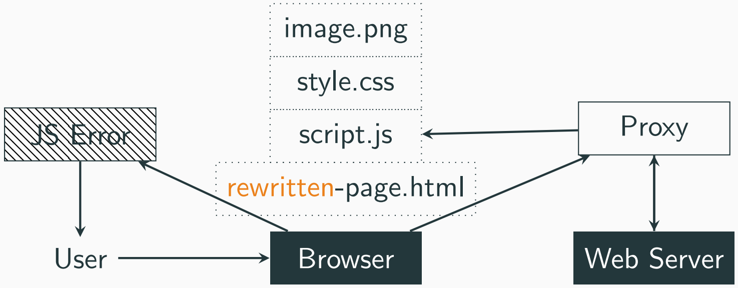


BikiniProxy – Architecture

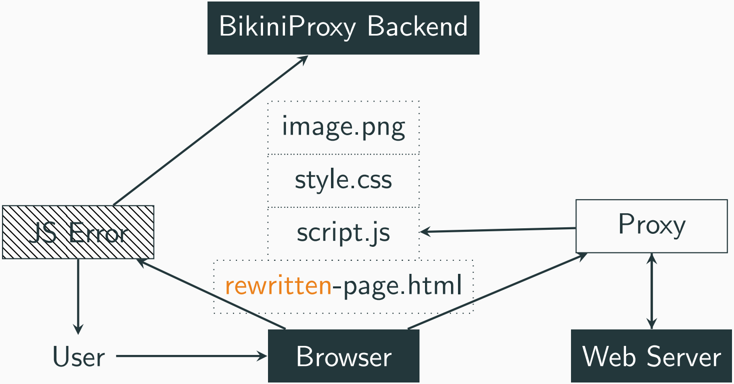


rewritten-page.html: web page with BikiniProxy framework.

BikiniProxy – Architecture



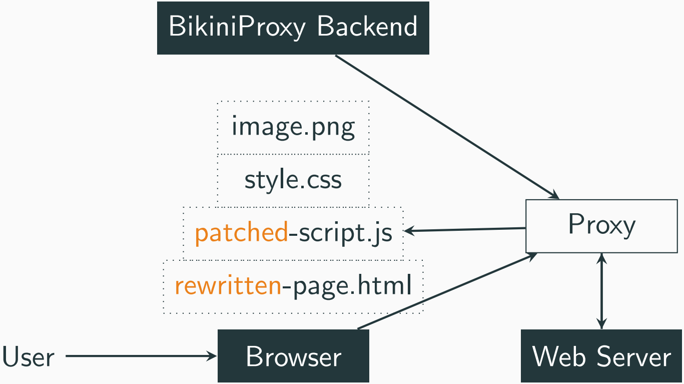
BikiniProxy – Architecture



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

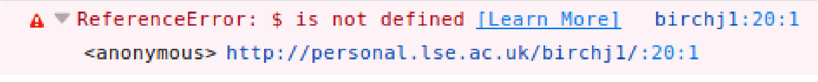
1. **HTTP/HTTPS Redirector** changes HTTP to HTTPS
2. **HTML Element Creator** creates HTML elements
3. **Library Injector** injects missing libraries

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

Crawling statistics	Value
# Visited pages	96174
# Pages with errors	4282 (4.5%)
Benchmark statistics	**Value**
# Pages with reproduced errors	555
# Errors	826
# Errors per page	1-10 (avg. 1.49)
Average page size	1.98mb

DeadClick is the first benchmark of reproducible JavaScript errors.

BikiniProxy – Evaluation Protocol

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

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

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.

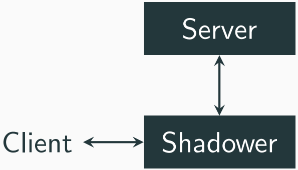
Outline

Itzal – Architecture

$$\text{Client} \longleftrightarrow \boxed{\text{Server}}$$

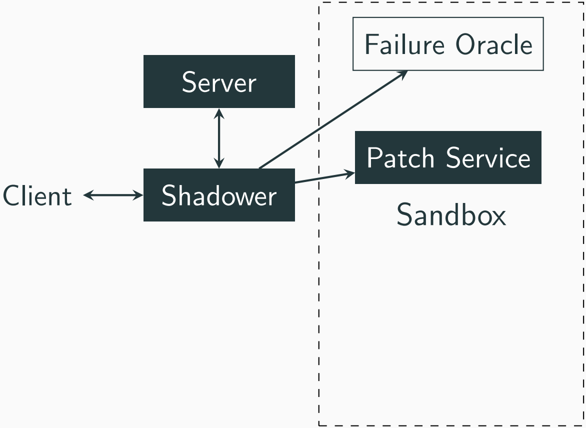
Client: e.g. a browser
Server: e.g. a web server

Itzal – Architecture



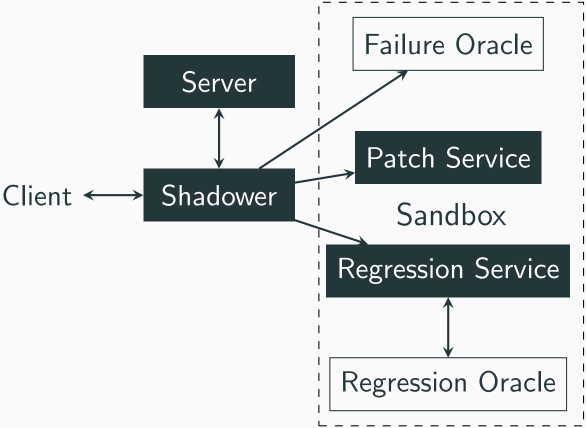
Shadower: intercepts and duplicates the requests

Itzal – Architecture



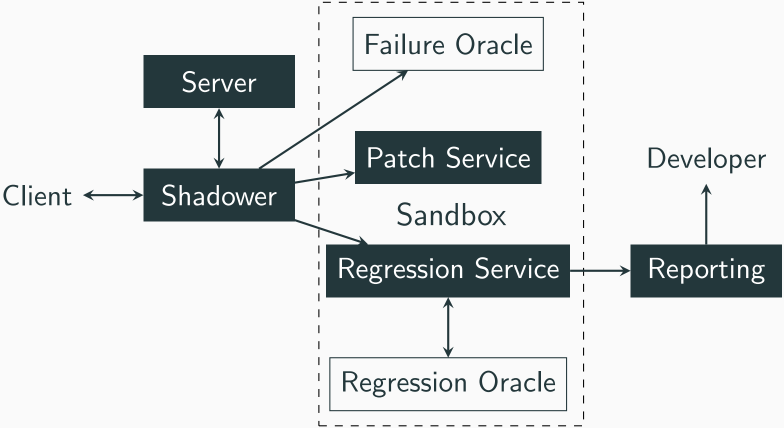
Patch Service: generates patches that fix the requests
Failure Oracle: detects if a request is passing or failing

Itzal – Architecture



Regression: executes passing request on patched server
Regression Oracle: compares the output of the original server and the patched server

Itzal – Architecture



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

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**.

Itzal – Evaluation 1 Protocol

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

	Repair Strategies			
	NPEFix		Exception-Stopper	
	# Valid	# Invalid	# Valid	# Invalid
34 bugs from 14 applications	23 118	31 060	198	592

NPEFix and Exception-Stopper can generate patches from a failing request.

Itzal – Evaluation 2 Protocol

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

Itzal – Evaluation 2 Regression Oracles

Visual behavior:

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

Program behavior:

- **Execution trace at method level**
  $Method_{patched} \simeq Method_{original}$
- **Execution trace at block level**
  $Block_{patched} \simeq Block_{original}$

Itzal – Evaluation 2 Results

	Differences				Is Valid Patch?
Patches	HTTP status	HTTP content	Trace Method	Trace Block	
Patch 1	⊗	⊗	⊗	⊗	Yes
Patch 2	⊗	●	●	●	No
Patch 3	⊗	●	●	●	No
...	...	...	...	...	
80 patches	16 ●	42 ●	39 ●	42 ●	23

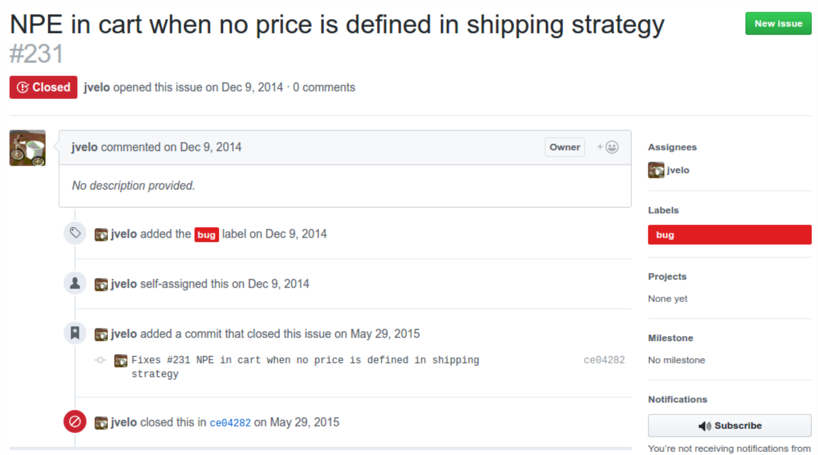
Regression oracles can detect behavior changes by observing the application behavior.

Itzal – Evaluate 3 Protocol

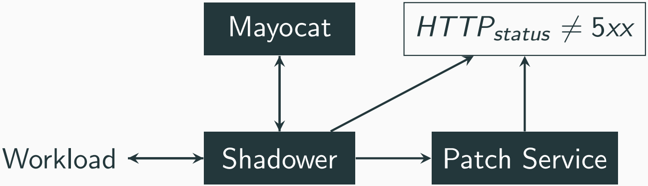
Goal: Assert that all the services of Itzal work together by evaluating it with two **cases studies**.

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

Itzal – Evaluation 3 Case Study



Itzal – Evaluation 3 Architecture



Itzal – Evaluation 3 Results

Repair Strategy	# Valid	# Invalid
NPEFix	105	182

Valid generated patch by Itzal for Mayocat

```
@@ 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

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

Open-science

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[3], Nopol[4], CapGen[5], ...

Regression Oracle: 🟢 Passing Tests

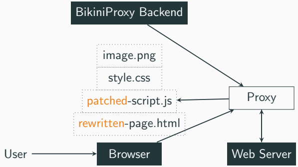
Failure Oracle: 🔴 Failing Tests

[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*

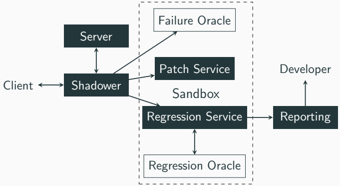
[5] Wen et al., "Context-Aware Patch Generation for Better Automated Program Repair", *ICSE'18*

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.

Itzal – Architecture



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

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