An Analysis of Patch Plausibility and Correctness for Generate-and-Validate Patch Generation Systems

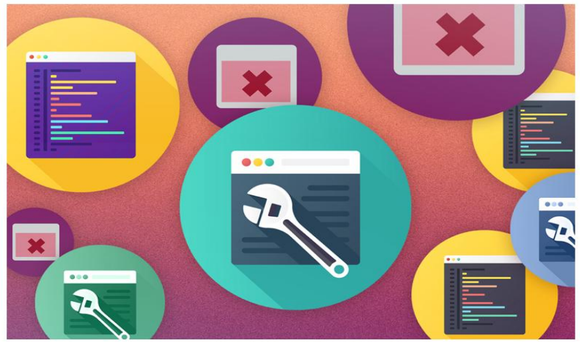
Paper Authors: Zichao Qi, Fan Long, Sara Achour, and Martin Rinard

Paper Presentation by:
Pulkit Bansal - 40321488

Introduction

- **What is Automatic Patch Generation?**
  - Generate-and-validate patch repair systems (GenProg, RSRepair, AE) attempt automatic patch generation.
- **Challenge:**
  - Many patches fail to produce correct outputs due to weak acceptance and validation criteria.
  - Existing systems generate incorrect or misleading patches, making debugging harder.
- **Solution:**
  - Traditional generate-and-validate methods struggle with complex patches.
  - Functionality deletion is a simpler, more focused approach.
- **Objective of the Paper:**
  - Introduce Kali, a patch generation system that removes functionality instead of modifying code.

Motivation



Automatic Patch Generation

Learning from How Developers Fix Bugs

Time-Consuming Debugging:

- Debugging inefficiencies arise due to weak test suites in GenProg, RSRepair, and AE.
- Most patches require manual inspection, as only 2/105 (1.9%) of GenProg patches were correct. (Figure 1, Page 30)

- **Inefficiency of Flat Delta Debugging:**
  - 104/110 (94.5%) of GenProg's plausible patches were just functionality deletions. (Section 5, Page 29)
  - Weak validation methods allow incorrect patches to pass test cases, misleading debugging efforts.
- **Real-World Problems:**
  - Systems generate patches for large real-world projects, but many introduce security vulnerabilities by removing key checks. (Section 5.2, Page 29)
- **Need for a Structured Approach:**
  - Kali, a functionality deletion-based system, achieves similar or better correctness rates than complex systems. (Figure 1, Page 30)

Problem Statement

- **Incorrect Patch Generation:**
  - GenProg, RSRepair, and AE generate patches that fail correctness checks.
  - Only 5/414 GenProg patches (1.2%) were correct. (Section 3, Page 28)
- **Weak Validation Mechanisms:**
  - Many patches pass test cases but remain incorrect due to weak test suites.
  - Example: 37/55 (67%) GenProg patches produced incorrect outputs despite validation. (Page 25, Section 1.1)
- **Security and Reliability Risks:**
  - 104/110 GenProg patches (94.5%) only deleted functionality. (Section 5, Page 29)
  - Functionality deletion leads to buffer overflows, security flaws, and logic failures. (Page 29, Section 5.2)

Related Work

- **GenProg, RSRepair, AE:**
  - Modify program structures via mutation-based genetic algorithms.
  - Flaw: Relies on weak test cases, leading to incorrect patches. (Section 1.1, Page 25)
- **ClearView (Binary Patching System):**
  - Uses learned invariants to generate patches that help systems survive defects.
  - Flaw: Limited correctness; only 4/10 ClearView patches were correct. (Section 7, Page 31)
- **Key Issue:**
  - Test suite limitations prevent automated repair systems from ensuring patch correctness. (Section 3, Page 28)

System	Approach	Success Rate	Limitations
GenProg	Mutation-based patching	**1.2% correct** (5/414)	Overfits to weak test cases
RSRepair	Similar to GenProg	**2/24 correct patches**	Still suffers from weak validation
AE	Heuristic-based mutation	**3/105 correct patches**	Large search space, inefficient
Kali	Functionality deletion	**3/105 correct patches**	Limited to deletion-only fixes

Proposed Solution – Kali

- **What is Kali?**
  - A functionality deletion-based patch generation system. (Section 1.4, Page 26)
  - Removes code instead of modifying logic, making patch generation simpler and more transparent.
- **Key Findings:**
  - Kali generates at least as many correct patches as GenProg, RSRepair, and AE. (Section 1.8, Page 27)
  - More effective in identifying defective functionality than traditional systems.
- **Efficiency:**
  - Kali's search space is ≤1700 patches per defect, while GenProg's is tens of thousands. (Page 30, Figure 1)

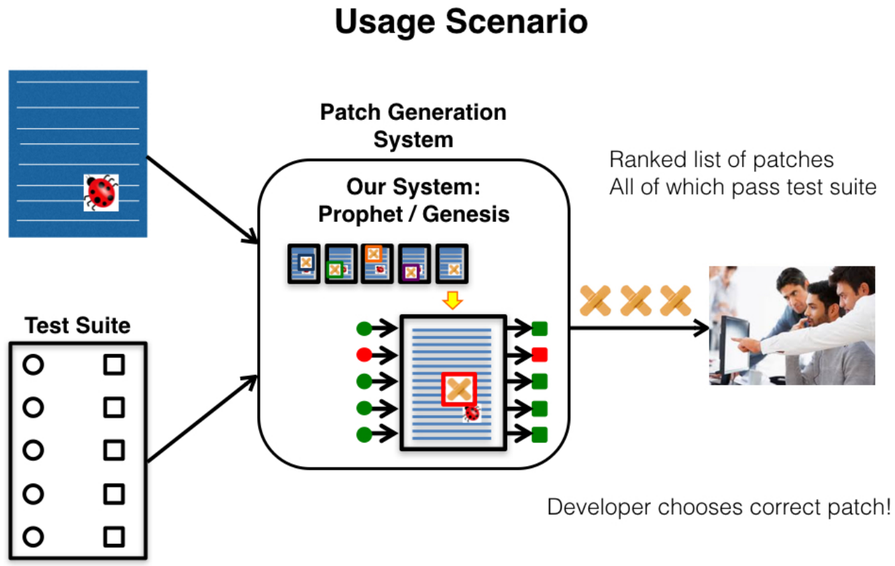
Plausible Patches Vs Correct Patches

- **Plausible Patches** – Patches that **pass test cases** but are **not necessarily correct**.
- **Correct Patches** – Patches that **truly fix the defect** and maintain program functionality.

📌 **High Plausibility, Low Correctness:**
•Most patches generated by GenProg, RSRepair, and AE were plausible but incorrect.
•Example: GenProg produced 104 plausible patches, but only 2 were actually correct.

📌 **Kali's Functionality Deletion Approach:**
•Kali matched or outperformed all systems in correct patches.
•Kali's plausible patches were ALL correct (27/27), unlike GenProg and RSRepair.

📌 **Security Risks of Plausible Patches:**
•Many plausible patches introduced security vulnerabilities, such as buffer overflows. *(Page 29, Section 5.2)*



Weak Proxies

- **Weak proxies** refer to flawed validation methods that assume a patch is correct **if it passes test cases**, rather than verifying if it truly fixes the defect.
- Weak Proxies are a problem because a patch might **pass existing test cases** but still be **semantically incorrect.**

Overfitting to Incomplete Test Suites

- Many test suites do **not cover all edge cases**, allowing **incorrect patches to pass validation.**
- **Result: False positives**, where bad patches are accepted as fixes.

Security Risks of Weak Proxies

- **Functionality deletion patches often pass tests** because removing code prevents failures.
- **Example:** A libtiff patch removed a crucial check, reintroducing **buffer overflow vulnerability** CVE-2006-2025.

Impact on Automated Patch Generation

Incorrect patches still make it to production due to weak validation methods.

Most "plausible patches" are not actually fixes but **just remove failing functionality** to pass tests.

Need for stronger validation techniques

Evaluation & Experimental Results

- **Dataset Used:**
  - 105 real-world defects from the GenProg benchmark set. (Section 6.1, Page 30)
- **Key Results:**
  - **Correct Patches:**
    - Kali: 3/105 defects
    - GenProg: 2/105 defects
    - AE: 3/105 defects
  - RSRepair: 2/24 defects (Page 30, Figure 1)
- **Plausible Patches (Functionality Deletion-Based):**
  - GenProg: 104/110
  - RSRepair: 37/44
  - AE: 22/27
  - Kali: 27/27 (Page 29, Section 5.2)
- **Performance:**
  - Kali finds patches in tens of minutes, GenProg takes hours. (Section 6.2, Page 30)

Evaluation & Experimental Results

Defect	GenProg	RSRepair	AE	Kali			
				Result	Search Space	Search Time	Type
fbc-5458-5459	Plausible‡	-	Plausible	Plausible	737	2.4m	SL†
gmp-14166-14167	Plausible‡	Plausible	Plausible‡	Plausible	1169	19.5m	DP
gzip-3fe0ca-39a362	Plausible‡	Plausible‡	Plausible	Plausible	1241	28.2m	SF (119)*
gzip-a1d3d4-f17cbd	No Patch	-	Plausible‡	No Patch			
libtiff-0860361d-1ba75257	Plausible	Implausible	Plausible	Plausible	1525	16.7m	SL*
libtiff-5b02179-3dfb33b	Plausible	Implausible	Plausible	Plausible	1476	4.1m	DP
libtiff-90d136e4-4c66680f	Implausible	Implausible	Plausible	Plausible	1591	45.0m	SL†
libtiff-d13be72c-ccadf48a	Plausible	Plausible	Plausible	Plausible	1699	42.9m	SL*
libtiff-ee2ce5b7-b5691a5a	Implausible	Implausible	Plausible	Plausible	1590	45.1m	SF(10)*
lighttpd-1794-1795	Plausible	-	Plausible	Plausible	1569	5.9m	
lighttpd-1806-1807	Plausible	Plausible	Plausible	Plausible	1530	55.5m	SF(21)†
lighttpd-1913-1914	Plausible	Plausible	No Patch	Plausible	1579	158.7m	SL*
lighttpd-2330-2331	Plausible	Plausible	Plausible	Plausible	1640	36.8m	SF(19)†
lighttpd-2661-2662	Plausible	Plausible	Plausible	Plausible	1692	59.7m	DP
php-307931-307934	Plausible	-	Plausible	Plausible	880	9.2m	DP
php-308525-308529	No Patch	-	Plausible	Plausible	1152	234.0m	SL†
php-309111-309159	No Patch	-	Correct‡	No Patch			
php-309892-309910	Correct	Correct	Correct	Correct	1498	20.2m	C
php-309986-310009	Plausible	-	Plausible	Plausible	1125	10.4m	SF(27)*
php-310011-310050	Plausible‡	-	Plausible	Plausible	971	12.9m	SL*
php-310370-310389	No Patch	-	No Patch	Plausible	1096	12.0m	DP
php-310673-310681	Plausible	-	Plausible	Plausible	1295	89.00m	SL*
php-311346-311348	No Patch	-	No Patch	Correct	941	14.7m	C
python-69223-69224	No Patch	-	Plausible‡	No Patch			
python-69783-69784	Correct	Correct	Correct	Correct	1435	16.1m	C
python-70098-70101	No Patch	-	Plausible‡	Plausible	1233	6.8m	SL*
wireshark-37112-37111	Plausible	Plausible‡	Plausible	Plausible	1412	19.6m	SL†
wireshark-37172-37171	No Patch	-	Plausible	Plausible	1459	10.9m	SL†
wireshark-37172-37173	No Patch	-	Plausible	Plausible	1459	10.9m	SL†
wireshark-37284-37285	No Patch	-	Plausible	Plausible	1482	11.5m	SL†

Figure 1: Experimental Results

Limitations & Technical Insights

- **Functionality Deletion is a Limited Strategy** – Kali is effective only when removing code is a viable solution; it cannot modify or rewrite logic, making it unsuitable for many real-world defects. (Page 27, Section 1.8)
- **Dependence on Existing Test Suites** – If the test suite is incomplete or weak, incorrect patches may still pass validation, leading to false positives. (Page 28, Section 3)
- **Not Ideal for Complex or Semantic Bugs** – Bugs requiring logical restructuring, variable modifications, or algorithmic corrections cannot be fixed with Kali's deletion-based approach. (Page 26, Section 1.8)
- **Potential for Security Vulnerabilities** – If Kali removes critical checks or error-handling mechanisms, it may unintentionally introduce security risks instead of fixing defects. (Page 29, Section 5.2)
- **Lack of Generalization to All Programming Languages** – Kali has been primarily evaluated on C programs, and its effectiveness on other languages (e.g., Java, Python) remains unclear. (Page 27, Section 1.8)
- **Weak Proxies Mislead Patch Evaluation** – Many generate-and-validate systems consider a patch valid if it passes test cases, but this approach is flawed as it does not ensure actual correctness.
- **Correctness vs. Plausibility** – The study differentiates between plausible patches (which pass test cases) and correct patches (which fix the defect correctly), emphasizing that many existing methods fail to ensure correctness.
- **Scalability Advantage of Kali** – Since Kali removes functionality instead of modifying code, its search space is significantly smaller and more efficient compared to GenProg and RSRepair.
- **Challenge in Handling Multi-Line or Structural Changes** – Kali operates at a fine-grained level, meaning it struggles with multi-line modifications or cases where restructuring is necessary.

Conclusion & Future Work

- **Summary & Impact**
  - Most patches from GenProg, RSRepair, and AE were incorrect due to weak validation criteria. (Page 25, Section 1.1)
  - Kali matches or outperforms existing systems by focusing on functionality deletion. (Page 30, Figure 1)
  - Stronger validation mechanisms are needed to ensure patches do not introduce new vulnerabilities. (Page 28, Section 3)
- **Future Work**
  - Integrating Semantic Analysis – Improve patch correctness by understanding code logic. (Page 27, Section 1.8)
  - Enhancing Test Suites – Strengthen validation mechanisms to prevent incorrect patches from passing. (Page 28, Section 3)
  - Expanding Kali's Scope – Apply Kali to Java, Python, and binary patching for broader impact. (Page 27, Section 1.8)

Thank You