Fuzzing the PHP Interpreter via Dataflow Fusion

A novel fuzzing framework for detecting memory errors in the PHP interpreter

</>

>200 Verified Bugs

PHP Official Toolchain

Yuancheng Jiang, Chuqi Zhang, Bonan Ruan, Jiahao Liu, Manuel Rigger, Roland Yap, Zhenkai Liang

School of Computing, National University of Singapore

NUS National University of Singapore

The PHP Security Challenge

PHP's Dominance in Web Development

- Cornerstone of the modern web, powering **over 70%** of websites globally
- Used for personal blogs to major e-commerce platforms
- Extensive **C codebase** (over a million lines) presents a significant attack surface

Overlooked Security Risks

- Existing PHP researches: application-level security (*e.g.*, SQL Injection)
- Memory corruption bugs in C most common cause of critical vulnerabilities

CVE-2023-3824

In PHP version 8.0.* when loading phar file, while reading PHAR directory entries, insufficient length checking may lead to a **stack buffer overflow**, leading potentially to memory corruption or RCE.

CVE-2024-8929

In PHP versions 8.1.* a hostile MySQL server can cause the client to **disclose the content of its heap** from other SQL requests and possible other data belonging to different users of the same server.

Research Question: how to improve the low-level security of PHP ecosystem?

Fuzzing

Fuzzing: well-known effective approach to discover bugs

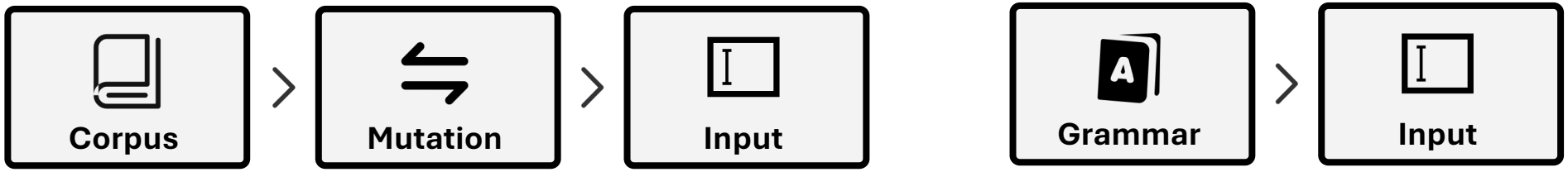
- Generating a diversity of inputs
- Driving software into unexpected states

Challenge: how to automatically generate diverse inputs?

- Mutation-based Input Generation
  - Starts from a corpus of existing inputs (seeds) and produces variants by applying random or heuristically guided mutations to those seeds.
- Grammar-based Input Generation
  - Uses a formal grammar (e.g., a BNF or ANTLR definition) of the input language to generate syntactically valid inputs from scratch.

Program Input Generation

Mutation- and Grammar-based Fuzzing



Mutation-based Fuzzing

Grammar-based Fuzzing

Common Weakness in Generation Programs

- Limited semantic diversity (although having high syntax correctness)
- Hard to reach deep logics when fuzzing compilers (*e.g.*, clang) or interpreters (*e.g.*, PHP)

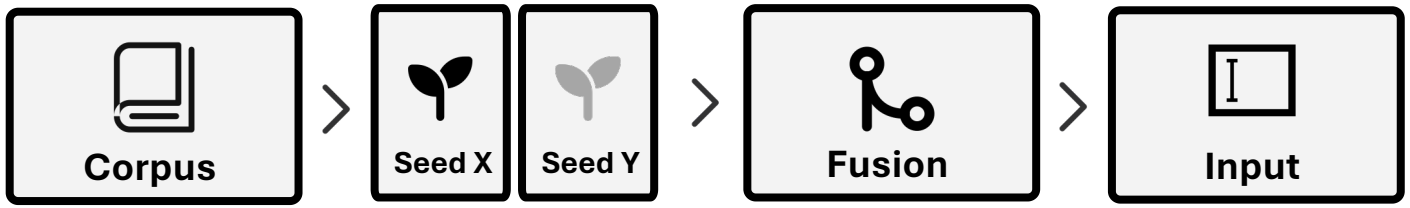
Research Question: how to generate inputs with diverse code semantics?

- Mutation-based fuzzing alters one input at one time

Can we take two or more inputs at one time and do further transformations?

Input Fusion

Fusion-based Fuzzing



What could be effective way to merge inputs with diverse semantics?

- Seed concatenation? It makes no big difference from executing them separately

Recall Seed Selection

- Known **Proof** of Concepts from bug reports, **unit** tests, *etc*.
- **Minimized** code snippet to test one single feature or bug reproduction

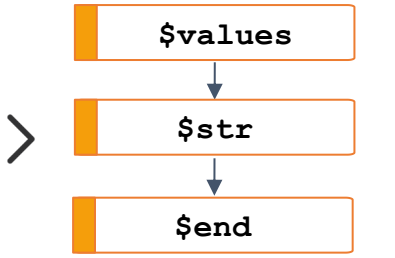
Seeds are typically unit-level program snippets designed to test a single functionality or regression in the codebase. They generally (*e.g.*, 96.1% in PHP) execute sequentially (*i.e.*, without branches). This implies that control flow has minimal impact on their code semantics.

Data Flow as a Representation

Data Flow

- Definition: how values move from their points of definition to their points of use

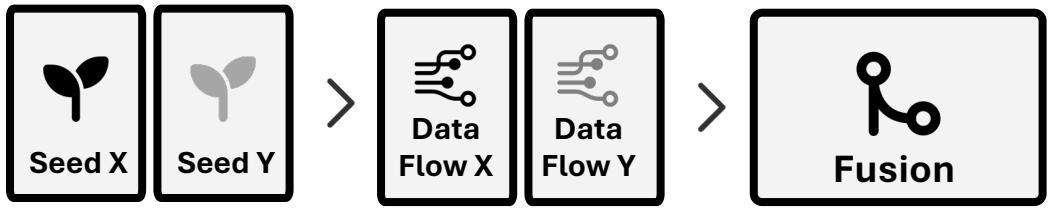
```php
<?php
  /* Test Case A */
  $values = array("Hello World", "\n\t1\r\n");
  foreach($values as $str) {
    $enc = base64_encode($str);
  }
```



data flow of PHP program

Seed Fusion to Data Flow Fusion

- Data flow can well represent code semantics of seeds due to the simple control flow
- Seed fusion becomes data flow fusion

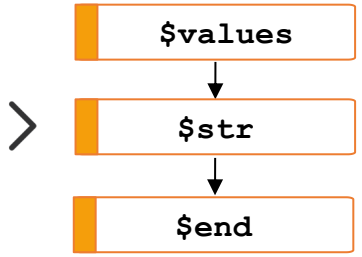


Data Flow Fusion

Data Flow Fusion

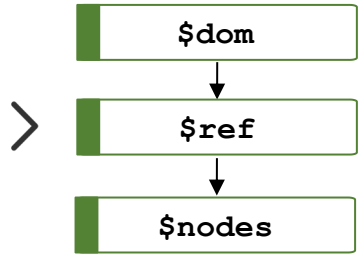
- Randomly interleaving data flows among two or more seed programs

```php
<?php
  /* Test Case A */
  $values = array("Hello World", "\n\t1\r\n");
  foreach($values as $str) {
    $enc = base64_encode($str);
  }
```



Test Case A: base64 encoding

```php
<?php
  /* Test Case B */
  $dom = new DOMDocument;
  $dom->loadXML('<tag>value</tag>');
  $ref = $dom->documentElement->firstChild;
  $nodes = $ref->childNodes;
```



Test Case B: DOM operations

Data Flow Fusion

Data Flow Fusion

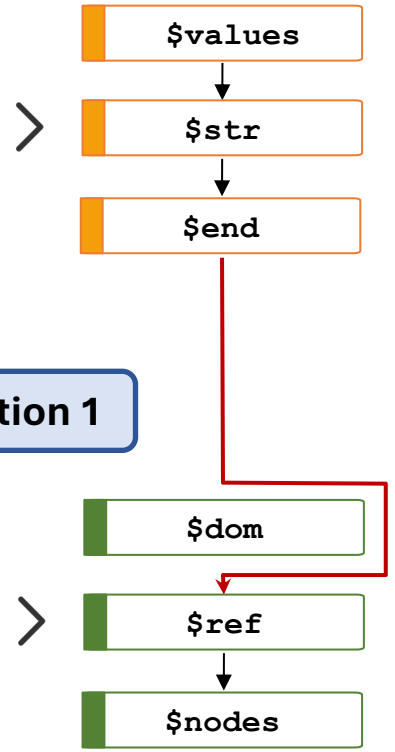
- Randomly interleaving data flows among two or more seed programs

```php
<?php
  /* Test Case A */
  $values = array("Hello World", "\n\t1\r\n");
  foreach($values as $str) {
    $enc = base64_encode($str);
  }
```

Test Case A: base64 encoding

```php
<?php
  /* Test Case B */
  $dom = new DOMDocument;
  $dom->loadXML('<tag>value</tag>');
  $ref = $dom->documentElement->firstChild;
  $nodes = $ref->childNodes;
```

Test Case B: DOM operations



Dataflow fusion option 1

```php
<?php
  /* Fused Test 01 */
  $values = array(..);
  foreach($values as $str) {
    $enc = base64_encode($str);
  }

  $fusion = $enc;

  $dom = new DOMDocument;
  $dom ->loadXML ('<tag>value</tag>');
  $ref = $fusion-> documentElement->..;
  $nodes = $ref->childNodes;
```

Observation:
See? Now we have a new test case to fuzz the DOM operation on base64 object!

Data Flow Fusion

Data Flow Fusion

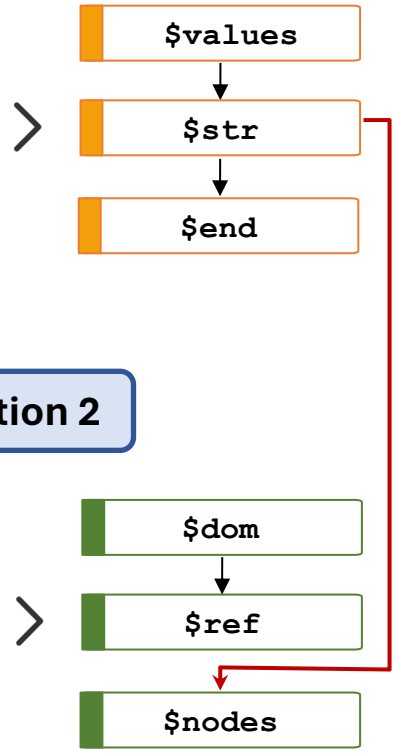
- Randomly interleaving data flows among two or more seed programs

```php
<?php
  /* Test Case A */
  $values = array("Hello World", "\n\t1\r\n");
  foreach($values as $str) {
    $enc = base64_encode($str);
  }
```

Test Case A: base64 encoding

```php
<?php
  /* Test Case B */
  $dom = new DOMDocument;
  $dom->loadXML('<tag>value</tag>');
  $ref = $dom->documentElement->firstChild;
  $nodes = $ref->childNodes;
```

Test Case B: DOM operations



Dataflow fusion option 2

```php
<?php
  /* Fused Test 02 */
  $values = array(..);
  foreach($values as $str) {
    $enc = base64_encode($str);
  }

  $fusion = $str;

  $dom = new DOMDocument;
  $dom ->loadXML ('<tag>value</tag>');
  $ref = $dom-> documentElement->..;
  $nodes = $fusion->childNodes;
```

Observation:
Random data flow combinations can create variant code semantics.

Data Flow Fusion

Data Flow Fusion

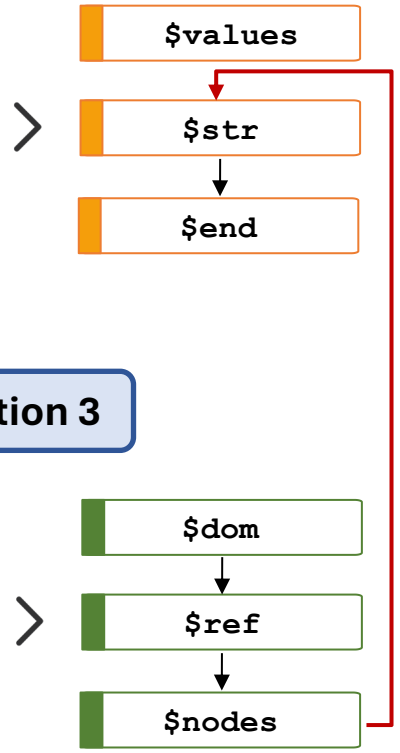
- Randomly interleaving data flows among two or more seed programs

```php
<?php
  /* Test Case A */
  $values = array("Hello World", "\n\t1\r\n");
  foreach($values as $str) {
    $enc = base64_encode($str);
  }
```

Test Case A: base64 encoding

```php
<?php
  /* Test Case B */
  $dom = new DOMDocument;
  $dom->loadXML('<tag>value</tag>');
  $ref = $dom->documentElement->firstChild;
  $nodes = $ref->childNodes;
```

Test Case B: DOM operations



Dataflow fusion option 3

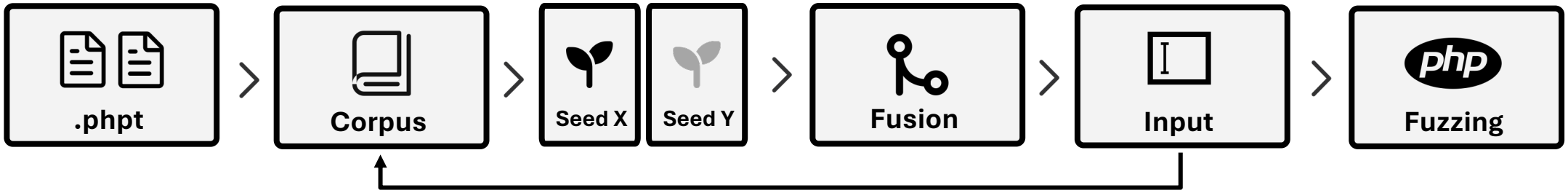
```php
<?php
  /* Fused Input 03 */
  $dom = new DOMDocument;
  $dom->loadXML('<tag>value</tag>');
  $ref = $dom->documentElement->firstChild;
  $nodes = $ref->childNodes;

  $fusion = $nodes;

  $values = array(..);
  foreach($fusion as $str) {
    $enc = base64_encode($str);
  }
// AddressSanitizer: heap-use-after-free
```

This data flow fusion triggers a **20-year-old** memory error existed in PHP interpreter!

FlowFusion Overview

Workflow

- FlowFusion collects phpt files as corpus, and execute inputs in PHP engines for fuzzing



Recursive Fusion

- Fused inputs can be added back to corpus if they have new coverage or find new bugs
- Recursive fusion enables more complex semantic fusion among >2 seed programs

Fuzzing Setup

- Continuously fuzzing the nightly build of PHP interpreters for six months
- Oracles: address sanitizer and undefined behavior sanitizers; crashes; assertion failures

Complementary Strategies

Test Mutation

- Before fusion, FlowFusion applies mutations to original test cases
- Introduces semantic variations while maintaining syntactic validity
- Examples: operators exchange (‘+’ to ‘%’), special values (‘1’ to ‘int_max’)

Interface Fuzzing

- After generating fused test cases, FlowFusion injects random calls to internal functions
- Uses variables from the fused context as arguments for these functions
- Effectively tests the robustness of PHP's API under novel and unexpected conditions

Environment Crossover

- Merges configurations (required modules, .ini settings) of seeds
- Randomly introduces other valid configurations collected from the entire test suite
- Examples of altered settings: memory limits, JIT settings, Opcache modes

Bug Results

Key Findings (from paper)

- FlowFusion detected **158** unique and unknown bugs
- Our bug reports led to security improvements across **80+** source files
- Developers introduce patches that modified **5,000+** lines of code

Bug Status Breakdown

- Among 158 bugs, 125 are fixed, 11 confirmed, 18 pending, 4 expected

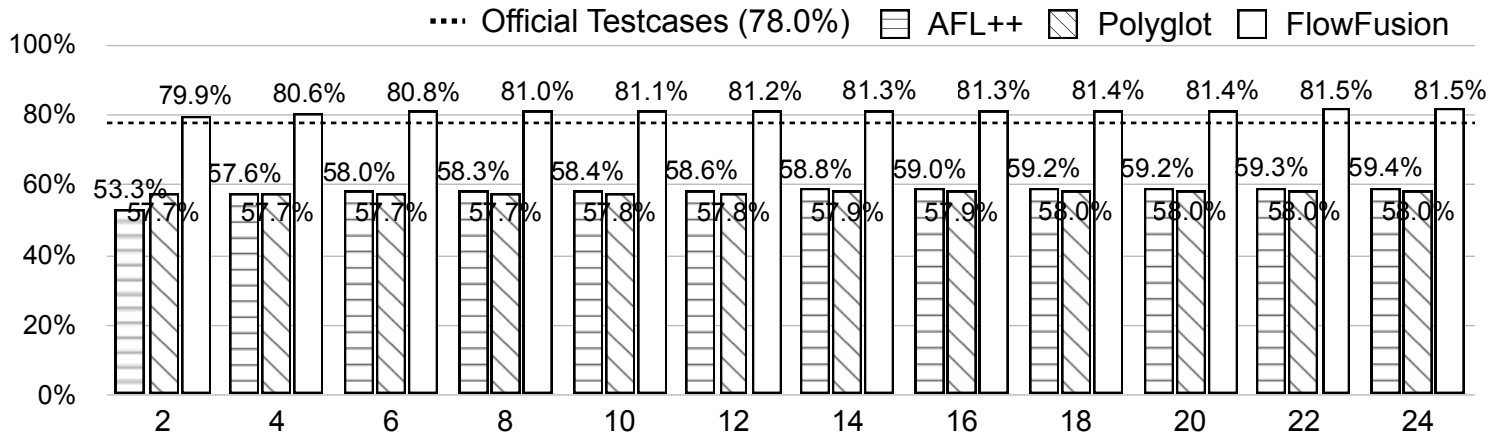
Discovered CWE Types

- FlowFusion covers bugs spanned 10 different CWE categories
- Examples: CWE-121 Stack-based Buffer Overflow, CWE-122 Heap-based Buffer Overflow, CWE124 Buffer Underwrite,, CWE-416 Use After Free, *etc.*

More Effectiveness

Comparison to existing works

- Baselines:
  - PHP test suite: maintained by PHP developers for CI/CD testing
  - AFL++: a well-known fuzzer that adds advanced instrumentation, persistent fuzzing, and customizable feedback for faster vulnerability discovery
  - PolyGlot (S&P'21): a multi-language fuzzer that translates inputs into a uniform IR, applies constrained, semantics-preserving mutations with semantic validation
- Settings: code coverage (collected via *gcovr*) after 24 hours fuzzing



Continued Impacts

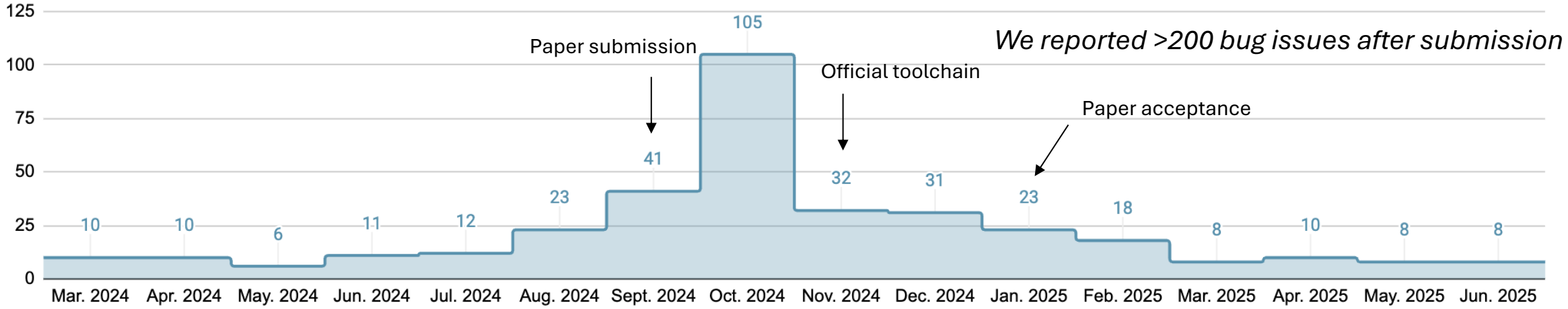
Living Project

FlowFusion seamlessly incorporates nightly builds in PHP source code to testing new features and patches. FlowFusion gets auto-updates when new test cases are added to official test suite.

Community Integration

FlowFusion becomes an official toolchain (i.e., available at https://github.com/php/flowfusion) for its outstanding bug discover effectiveness to help developers improve PHP interpreter security.

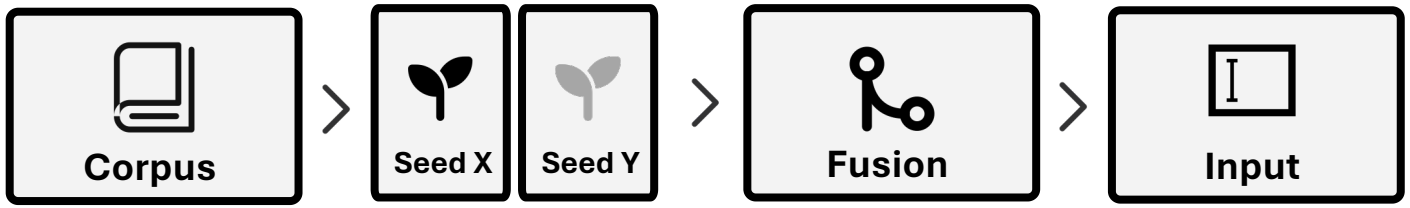
FlowFusion's Monthly Bug Report Count



FlowFusion's impact is not bound by paper submission or acceptance; it focuses on sustained community support. FlowFusion continuously detects new PHP memory errors nearly every week.

Summary

Fusion-based Fuzzing



(Data Flow As a Representation)

Availability

- Source code: https://github.com/php/flowfusion
- Bug reports: https://github.com/php/php-src/issues?q=author%3AYuanchengJiang

Future work

- More oracles to detect other bug categories (*e.g.*, logic bugs) that FlowFusion missed
- Adaptation to other programming language implementations (*e.g.*, cPython, v8)
- Combination with large language models