CONFETTI: Amplifying Concolic Guidance For Fuzzers

James Kukucka, Luís Pina, Paul Amman, Jonathan Bell

Problem: Conventional unit testing is not enough to find latent bugs in program application logic

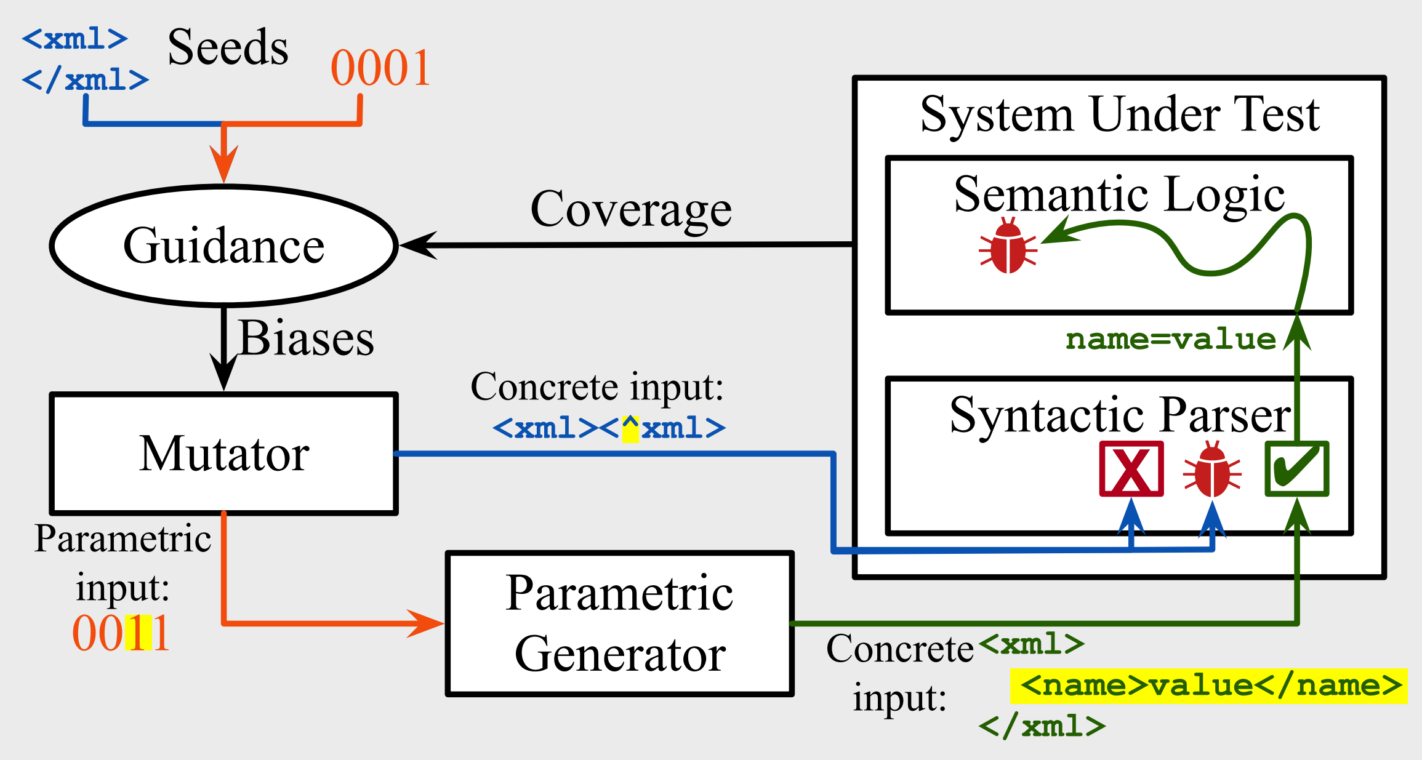
Developers often introduce latent bugs within application logic

```
protected boolean substitute(final LogEvent event, final StringBuilder buf,
final int offset, final int length) {
    return substitute(event, buf, offset, length, null) > 0;
}


Exception in thread "Thread-2" java.lang.StackOverflowError
at java.lang.StringBuilder.getChars(StringBuilder.java:76)
at org.apache.logging.log4j.core.lookup.StrSubstitutor.getChars(StrSubstitutor.java:1401)
at org.apache.logging.log4j.core.lookup.StrSubstitutor.substitute(StrSubstitutor.java:939)
at org.apache.logging.log4j.core.lookup.StrSubstitutor.substitute(StrSubstitutor.java:912)
at org.apache.logging.log4j.core.lookup.StrSubstitutor.substitute(StrSubstitutor.java:978)
at org.apache.logging.log4j.core.lookup.StrSubstitutor.substitute(StrSubstitutor.java:1042)
...
```

This code snippet comes from CVE-2021-45105, one of the high-profile log4j vulnerabilities discovered in 2021. The function shown is subject to a stack overflow from a well-crafted input. It behooves developers to utilize means of detecting these syntactically valid, yet unexpected inputs to programs, to catch these latent bugs.

Parametric fuzzers can help developers to catch latent bugs



Parametric fuzzers are the current state of the art when it comes to greybox fuzzing for bugs in application logic. By mutating parametric inputs, these fuzzers' mutations are abstracted at the input generation level, allowing inputs to pass syntactic parsing and reveal bugs within the application logic. Parametric fuzzing is illustrated in the orange path, while traditional greybox fuzzing such as AFL is illustrated in the blue path.

Modern taint tracking and concolic execution techniques can augment fuzzer coverage, but don't capture relationships through control flow

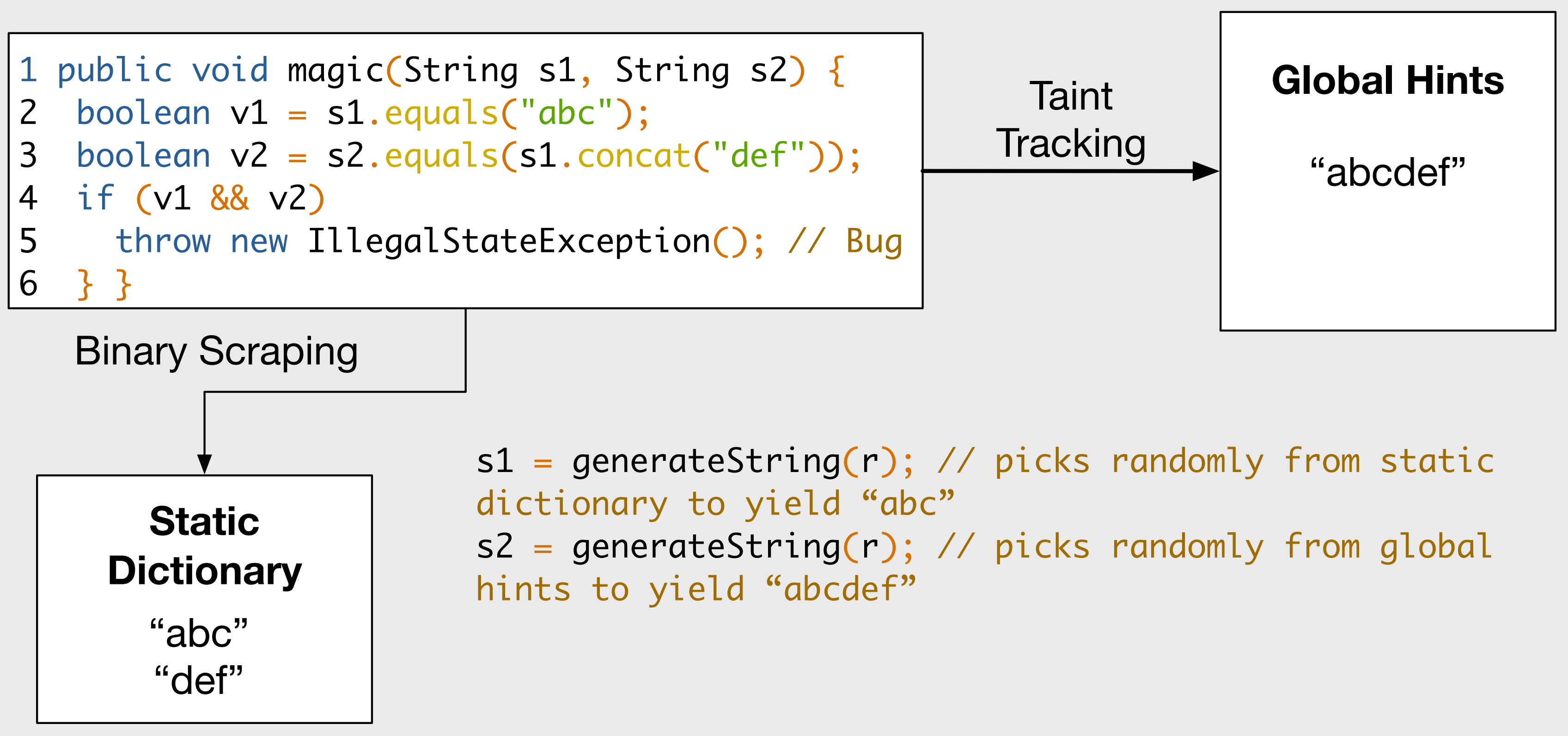
```
1 public void magic(String s1, String s2) {
2  if (s1.equals("abc") && s2.equals("def"))
3    throw new IllegalStateException(); // Bug
4  } }
```

In this example, taint tracking and concolic execution would succeed in covering line 3 because there is a data flow relationship between s1, s2 and the predicate on line 2.

```
1 public void magic(String s1, String s2) {
2  boolean v1 = s1.equals("abc");
3  boolean v2 = s2.equals(s1.concat("def"));
4  if (v1 && v2)
5    throw new IllegalStateException(); // Bug
6  } }
```

In this example, a taint tracking tool will not report any relationship between the input and the branch on line 4 because v1 is control-dependent on s1, but not data-dependent.

Global hinting is a novel strategy that helps further augment fuzzing coverage despite the loss of taint tags



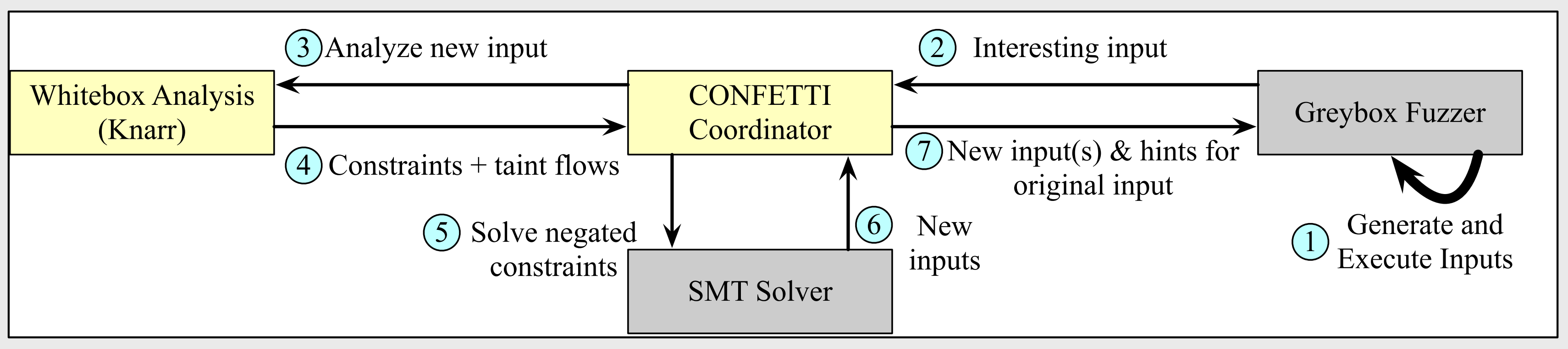
```
1 public void magic(String s1, String s2) {
2  boolean v1 = s1.equals("abc");
3  boolean v2 = s2.equals(s1.concat("def"));
4  if (v1 && v2)
5    throw new IllegalStateException(); // Bug
6  } }
```

```
s1 = generateString(r); // picks randomly from static
dictionary to yield "abc"
s2 = generateString(r); // picks randomly from global
hints to yield "abcdef"
```

By applying targeted hints pseudorandomly anywhere in the input, global hinting enables coverage of line 5.

Our Solution: CONFETTI combines parametric fuzzing, targeted hinting, and novel global hinting to achieve higher coverage than a baseline parametric fuzzer and targeted hinting alone.

CONFETTI uses a non-blocking architecture to maximize performance



Whitebox Analysis (Knarr)
CONFETTI Coordinator
Greybox Fuzzer
SMT Solver
1 Generate and Execute Inputs
2 Interesting input
3 Analyze new input
4 Constraints + taint flows
5 Solve negated constraints
6 New inputs
7 New input(s) & hints for original input

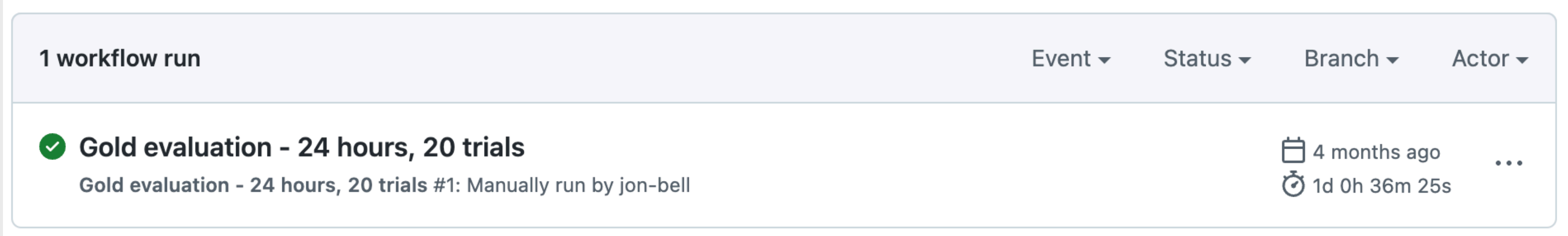
The CONFETTI coordinator queues inputs for whitebox analysis and delivers targeted and global hints to the greybox fuzzer, all without inhibiting the performance of the greybox fuzzer.

CONFETTI achieves higher branch coverage across most benchmarks

Program	Total Branches	JQF-Zest	CONFETTI (Targeted Hints Only)	CONFETTI
Apache Ant	23,361	859	871	872
Apache BCEL	6,220	1361	1423	1421
Google Closure	49,602	10,545	10,640	11,458
Apache Maven	5,858	821	853	857
Mozilla Rhino	25,035	3,757	3,534	3,744

CONFETTI's continuous integration workflow allows for easier evaluation and extensibility

Fork CONFETTI on Github



1 workflow run   Event ▾   Status ▾   Branch ▾   Actor ▾

✓ Gold evaluation - 24 hours, 20 trials
Gold evaluation - 24 hours, 20 trials #1: Manually run by jon-bell
4 months ago
1d 0h 36m 25s

https://github.com/neu-se/confetti

Our GitHub repository has a continuous integration workflow to run rapid performance evaluations of pull requests. We hope that our open-source release of CONFETTI and its continuous workflow will help to support the growing community of practitioners and researchers engaged in fuzzing JVM-based software. Please feel free to fork CONFETTI on GitHub and try it out in your fuzzing workflow. You can access our repository by scanning the QR code!

CONFETTI is able to find more bugs with a higher rate of repeatability

Issue	Status	JQF-Zest	CONFETTI (Targeted Hints Only)	CONFETTI
Ant 1		100	100	100
BCEL 1		100	0	0
BCEL 2		100	100	0
BCEL 3	Reported	0	0	40
BCEL 4		0	0	80
BCEL 5	Reported	0	5	100
BCEL 6	Reported	0	20	100
Closure 1		100	100	100
Closure 2		90	85	5
Closure 3		80	70	45
Closure 4	Acknowledged	0	45	95
Closure 5	Fixed	0	15	90
Closure 6		0	0	5
Closure 7	Acknowledged	0	20	100
Closure 8	Fixed	0	0	100
Closure 9	Acknowledged	15	15	20
Closure 10		0	5	100
Closure 11	Fixed	0	0	100
Closure 12	Fixed	0	0	35
Closure 13		0	0	20
Closure 14		0	0	5
Closure 15		0	0	5
Rhino 1		100	100	100
Rhino 2		100	100	100
Rhino 3		100	100	100
Rhino 4		100	100	100

This heat map shows the rate of repeatability for each bug that JQF-Zest, CONFETTI with only targeted hinting and CONFETTI with global hinting were able to find. Cells range in color from red to green, with green indicating a repeatability rate approaching 100%. Where applicable, we also indicated bugs that were reported, acknowledged, or fixed by developers.