CONFETTI: Amplifying Concolic Guidance for Fuzzers

James Kukucka, Luís Pina, Paul Ammann, Jonathan Bell

George Mason University
UIC
Northeastern University

Motivation - CVE-2021-45105 log4j DoS Vulnerability

CVE-2021-44832: New Vulnerability Found in Apache Log4j

Summary

A new vulnerability was discovered in the Apache Log4j library. Tracked as CVE-2021-44832, this bug may allow arbitrary code execution in compromised systems when the attacker has permissions to modify the logging configuration file.

CVE-2021-44832 has received a CVSS score of 6.6 out of 10, and it affects all versions of Log4j from 2.0-alpha7 to 2.17.0, excluding 2.3.2 and 2.12.4. This is the fourth Log4j vulnerability addressed by Apache in December 2021, followed by:

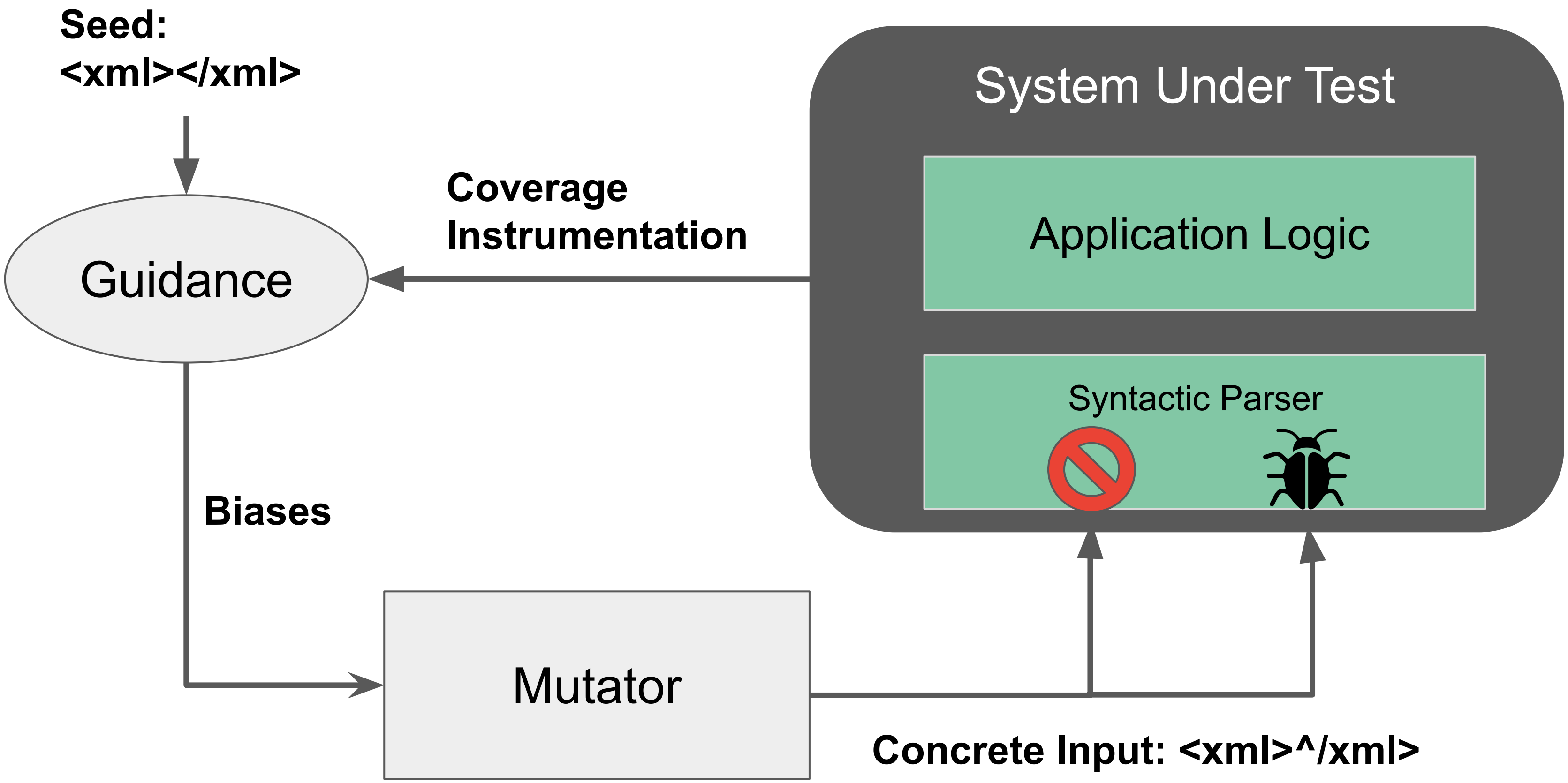
- CVE-2021-45105: Vulnerability that could allow DoS attacks (CVSS 5.9)
- CVE-2021-45046: Vulnerability that could allow Remote Code Execution (CVSS 9.0)
- CVE-2021-44228: Vulnerability that could allow Remote Code Execution (CVSS 10.0)

```
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)
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)
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)
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)
...
```

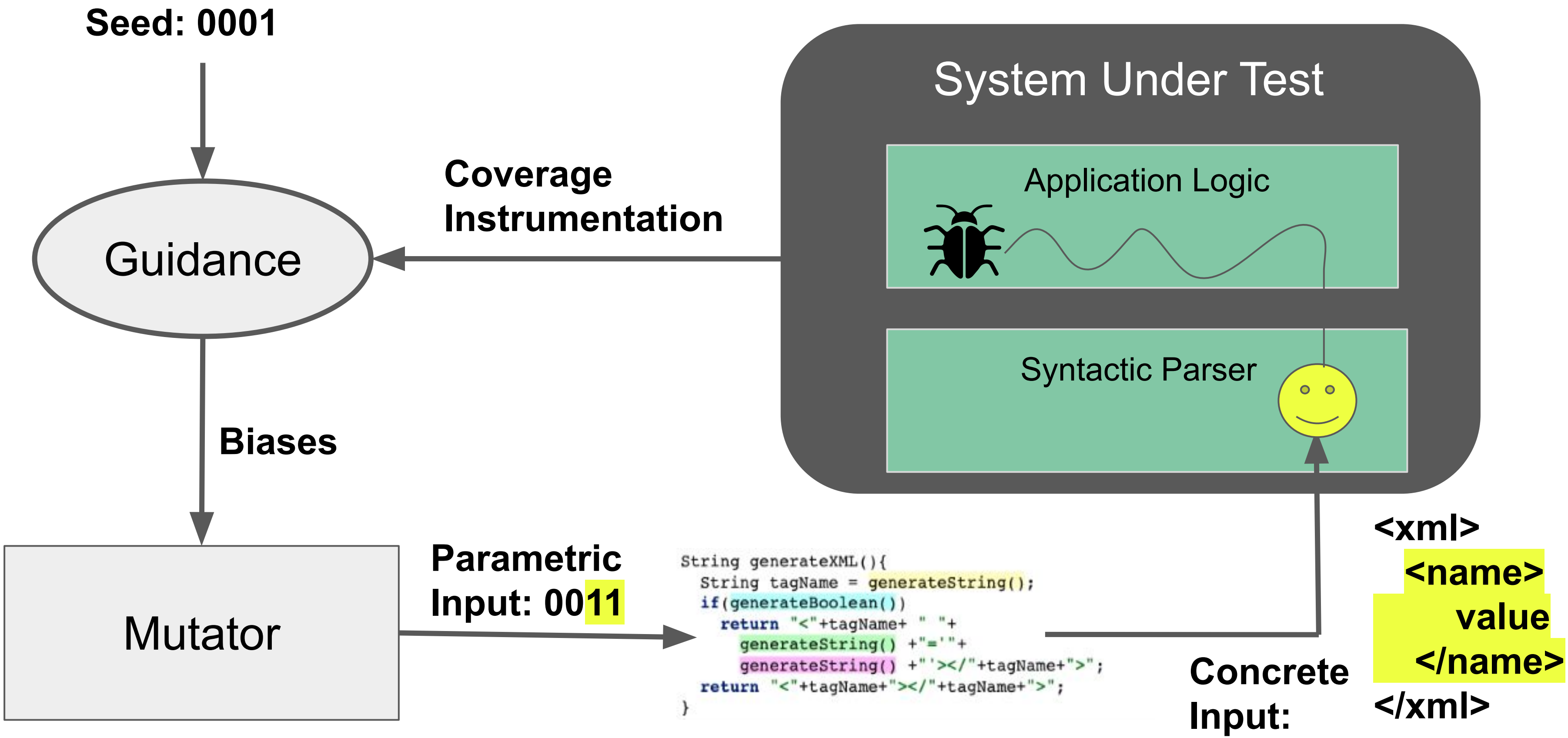
```
protected boolean substitute(final LogEvent event, final StringBuilder buf, final int offset, final int length) {
    return substitute(event, buf, offset, length, null) > 0;
}
```

https://www.netskope.com/blog/cve-2021-44832-new-vulnerability-found-in-apache-log4j#:~:text=Summary,modify%20the%20logging%20configuration%20file.

Introduction: Parametric Fuzzers vs Greybox Fuzzers



State-of-the-art

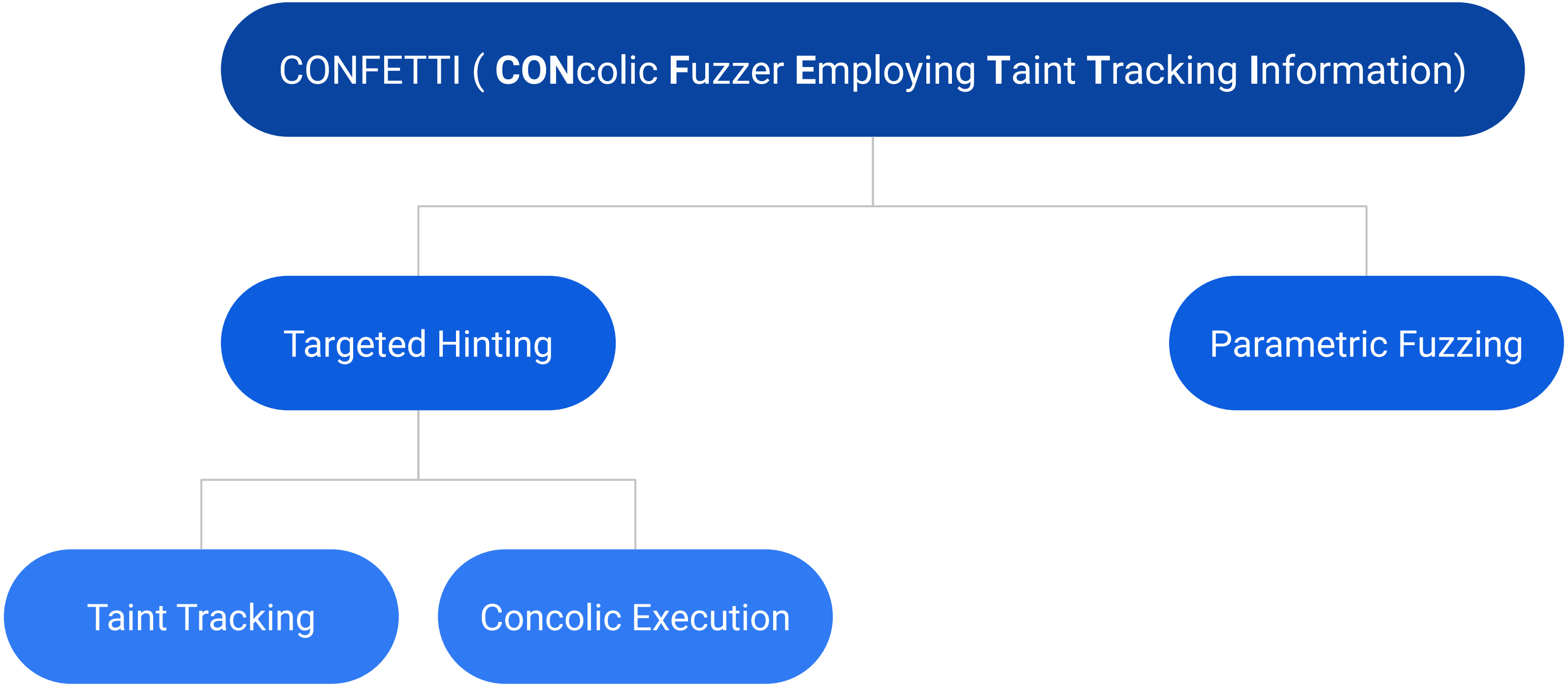


Our Solution, CONFETTI, leverages state-of-the-art parametric fuzzing and novel hinting

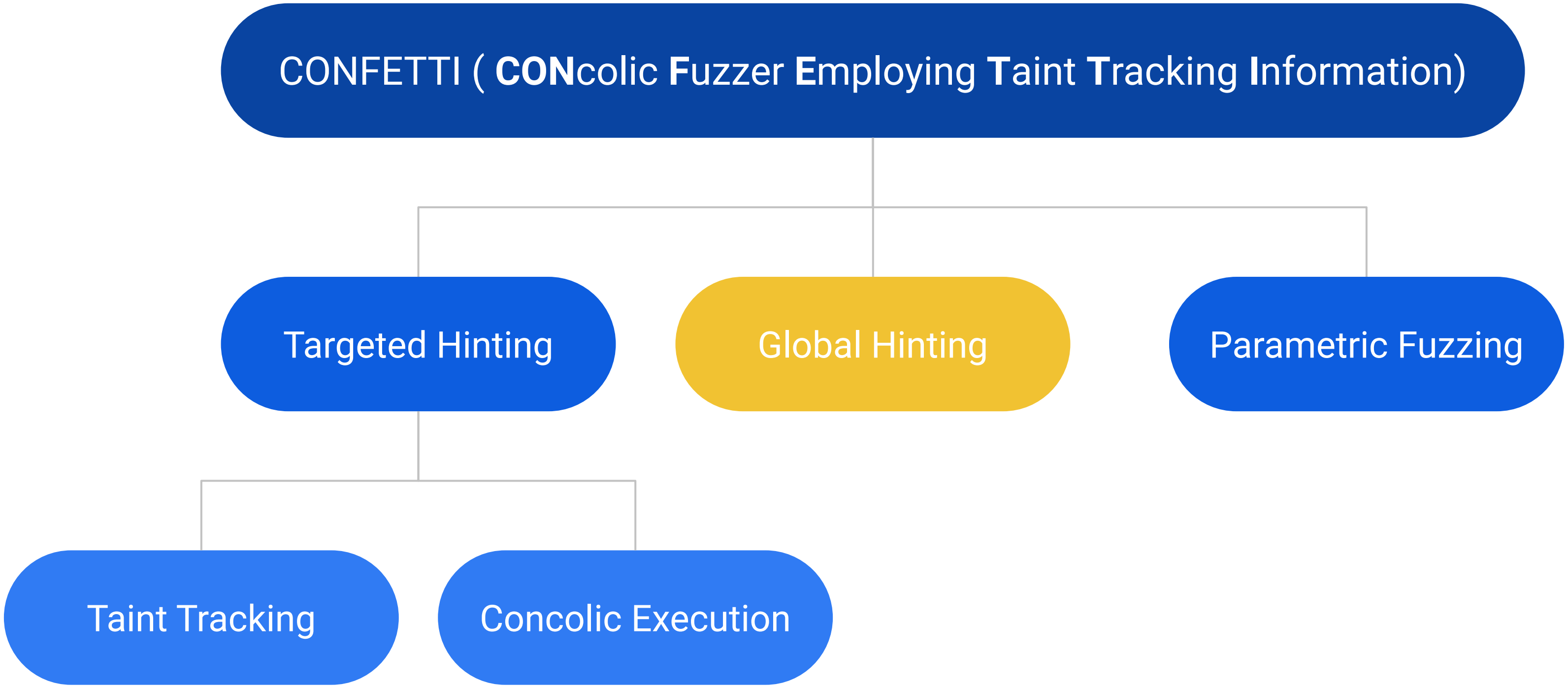
CONFETTI (**CON**colic **F**uzzer **E**mploying **T**aint **T**racking **I**nformation)

Our Solution, CONFETTI, leverages state-of-the-art parametric fuzzing and novel hinting

Our Solution, CONFETTI, leverages state-of-the-art parametric fuzzing and novel hinting



Our Solution, CONFETTI, leverages state-of-the-art parametric fuzzing and novel hinting



Example of applying targeted hints

```
String generateXML(){
  String tagName = generateString();
  if(generateBoolean())
    return "<"+tagName+ " "+
      generateString() +"='"+
      generateString() +"'></"+tagName+">";
  return "<"+tagName+"></"+tagName+">";
}
```

(a) Simple parametric XML tag generator that uses 4 random choices

#	Source	Parametric Input	Generator Choices	New Coverage
1	seed	010110100	"groupID" false	5

```
byte[] input = generateXML();
XMLDocument doc = parse(input);

if (!"expected".equals(doc.getElement(0).getName()))
  throw new Error();

String v = doc.getElement(0).getAttr("version");
if (v == null) throw new Error();

```

Example of applying targeted hints

```
String generateXML(){
  String tagName = generateString();
  if(generateBoolean())
    return "<"+tagName+ " "+
      generateString() +"='"+
      generateString() +"'></"+tagName+">";
  return "<"+tagName+"></"+tagName+">";
}
```

(a) Simple parametric XML tag generator that uses 4 random choices

#	Source	Parametric Input	Generator Choices	New Coverage
1	seed	01011010	"groupID" false	5
2	mutation	01011110	"package" false	-

```
byte[] input = generateXML();
XMLDocument doc = parse(input);

if (!"expected".equals(doc.getElement(0).getName()))
  throw new Error();

String v = doc.getElement(0).getAttr("version");
if (v == null) throw new Error();

```

Example of applying targeted hints

```
String generateXML(){
  String tagName = generateString();
  if(generateBoolean())
    return "<"+tagName+ " "+
      generateString() +"='"+
      generateString() +"'></"+tagName+">";
  return "<"+tagName+"></"+tagName+">";
}
```

(a) Simple parametric XML tag generator that uses 4 random choices

#	Source	Parametric Input	Generator Choices	New Coverage
1	seed	01011010	"groupID" false	5
2	mutation	01011110	"package" false	-
3	hint	10010110	"expected" false	6

```
byte[] input = generateXML();
XMLDocument doc = parse(input);

if (!"expected".equals(doc.getElement(0).getName()))
  throw new Error();

String v = doc.getElement(0).getAttr("version");
if (v == null) throw new Error();
```

Example of applying targeted hints

```
String generateXML(){
  String tagName = generateString();
  if(generateBoolean())
    return "<"+tagName+ " "+
      generateString() +"='"+
      generateString() +"'></"+tagName+">";
  return "<"+tagName+"></"+tagName+">";
}
```

(a) Simple parametric XML tag generator that uses 4 random choices

#	Source	Parametric Input	Generator Choices	New Coverage
1	seed	01011010	“groupID”false	5
2	mutation	01011110	“package”false	-
3	hint	10010110	“expected”false	6
4	mutation	10010111011010001101	“expected”true“groupID”“A”	7, 8

```
1 byte[] input = generateXML();
2 XMLDocument doc = parse(input);
3
4 if (!"expected".equals(doc.getElement(0).getName()))
5   throw new Error();
6
7 String v = doc.getElement(0).getAttr("version");
8 if (v == null) throw new Error();
9
```

Example of applying targeted hints

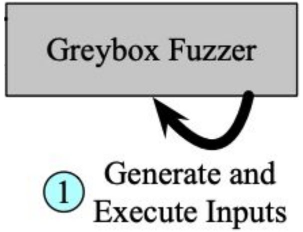
```
String generateXML(){
  String tagName = generateString();
  if(generateBoolean())
    return "<"+tagName+ " "+
      generateString() +"='"+
      generateString() +"'></"+tagName+">";
  return "<"+tagName+"></"+tagName+">";
}
```

(a) Simple parametric XML tag generator that uses 4 random choices

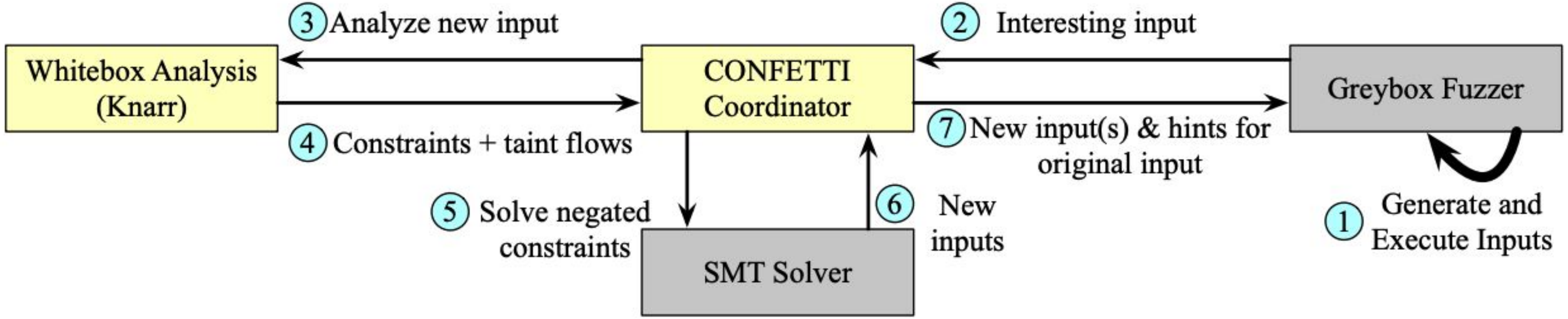
#	Source	Parametric Input	Generator Choices	New Coverage
1	seed	0101101 0	"groupID" false	5
2	mutation	0101111 0	"package" false	-
3	hint	1001011 0	"expected" false	6
4	mutation	1001011 1 011010 01101	"expected" true "groupID" "A"	7, 8
5	hint	1001011 1 011111 01101	"expected" true "version" "A"	9

```
1 byte[] input = generateXML();
2 XMLDocument doc = parse(input);
3
4 if (!"expected".equals(doc.getElement(0).getName()))
5   throw new Error();
6
7 String v = doc.getElement(0).getAttr("version");
8 if (v == null) throw new Error();
9
```

CONFETTI uses a non-blocking Architecture



CONFETTI uses a non-blocking architecture



KNARR builds path conditions by propagating taint tags from parametric bytes to concrete input

- KNARR is able to taint parametric input bytes and propagate taint tags with minimal changes to underlying generators.
- Strings are tainted at the character level, and operations such as `equals()` and `startsWith()` are instrumented.
- KNARR extends the taint engine to create an abstract expression as part of the taint tag, building it as taints are propagated to new variables.
- When a tainted input reaches a branch, the taint tag of the branch is the complete symbolic expression from the parametric input.
- KNARR facilitates concolic execution in this way, as opposed to pure symbolic execution.

The CONFETTI coordinator ingests constraints from KNARR to attempt to discover new branches

- In the style of concolic execution, the CONFETTI Coordinator targets branches based on whether they are uncovered and whether their branch predicate contains some part of the input
- Branch is negated and all other constraints are dropped, then it is passed to Z3.
- Helpful to cover branches the fuzzer got stuck on.
- User-configurable parameters to cut down on wasted solving time.

Taint Tracking Doesn’t Capture Relationships Through Control Flow

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

Global hinting allows CONFETTI to explore branches it could not otherwise.

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

Static Dictionary

“abc”
“def”

Global hinting allows CONFETTI to explore branches it could not otherwise.

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

Taint Tracking →

Global Hints

"abcdef"

Static Dictionary

"abc"
"def"

```
public String generateString(ParametricInputArray r) {
        if( r.nextBoolean() )
        {
                return static_dict[r.nextInt()];
        }
        return global_hints[r.nextInt()];
}

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

CONFETTI leverages both targeted and global hints in guiding the fuzzer

- CONFETTI does not seek to purely perform whitebox analysis, but to *guide* the fuzzing process so that it maximizes the efficiency of greybox fuzzing.
- It does this by leveraging several choices when mutating an input:
  - Apply a single targeted hint
  - Apply multiple targeted hints
  - Mutate, which may or may not apply global hints
- Hints are *inheritable* meaning they are preserved in future generations (if an input reveals new coverage and is fuzzed again).
- Stacking hints allows for more complex inputs that may reveal new coverage.

On most benchmark programs, the use of CONFETTI's global hinting with targeted hinting resulted in higher branch coverage and more bugs found.

		Total Branch Coverage			Bugs Found		
Program	Total Branches	Zest	CONFETTI_tgt	CONFETTI	Zest	CONFETTI_tgt	CONFETTI
ant	23,361	859	871	872	1	1	1
bcel	6,220	1361	1423	1421	2	3	5
closure	49,602	10,545	10,640	11,458	4	8	15
maven	5,858	821	853	857	0	0	0
rhino	25,035	3,757	3,534	3,744	4	4	4

CONFETTI finds more bugs, including bugs that the baseline fuzzer cannot

Issue #	JQF-Zest	CONFETTI_tgt	CONFETTI
A1	100	100	100

Issue #	JQF-Zest	CONFETTI_tgt	CONFETTI
B1	100	0	0
B2	100	100	0
B3	0	0	40
B4	0	0	80
B5	0	5	100
B6	0	20	100

Issue #	JQF-Zest	CONFETTI_tgt	CONFETTI
R1	100	100	100
R2	100	100	100
R3	100	100	100
R4	100	100	100

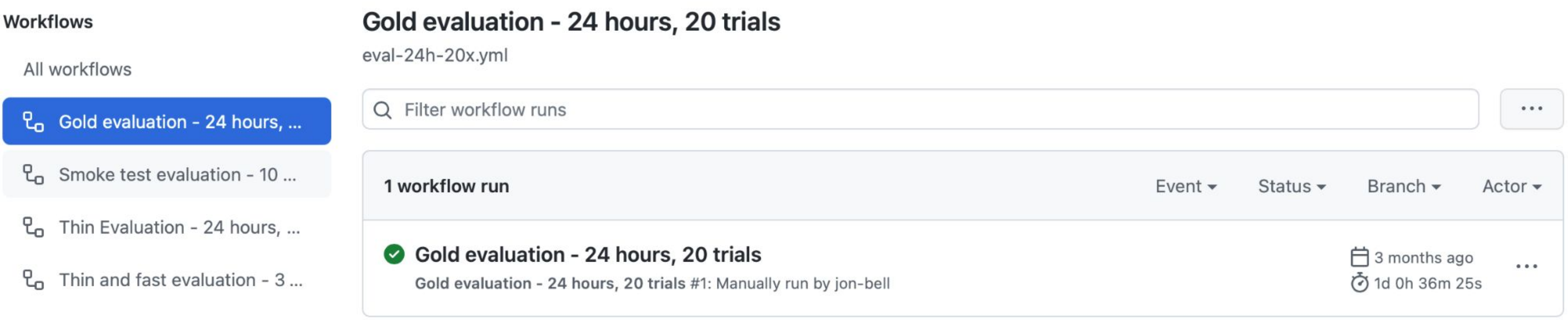
Issue #	JQF-Zest	CONFETTI_tgt	CONFETTI
C1	100	100	100
C2	90	85	5
C3	80	70	45
C4	0	45	95
C5	0	15	90
C6	0	0	5
C7	0	20	100
C8	0	0	100
C9	15	15	20
C10	0	5	100
C11	0	0	100
C12	0	0	35
C13	0	0	20
C14	0	0	5
C15	0	0	5

Our evaluation, all data and CONFETTI are archived and open-source

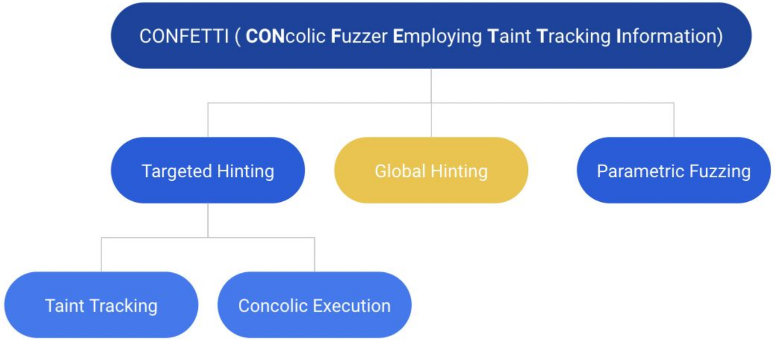
https://doi.org/10.6084/m9.figshare.16563776

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

Continuous Integration workflow allows for easy evaluation



Our Solution, CONFETTI, leverages state-of-the-art parametric fuzzing and novel hinting



Global hinting allows CONFETTI to explore branches it could not otherwise.

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

Taint Tracking →

Global Hints

"abcdef"

Static Dictionary

"abc"
"def"

```
public String generateString(ParametricInputArray r) {
        if( r.nextBoolean() )
        {
                return static_dict[r.nextInt()];
        }
        return global_hints[r.nextInt()];
}

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

CONFETTI finds more bugs, including bugs that the baseline fuzzer cannot

UPDATE

Issue #	JQF-Zest	CONFETTI_tgt	CONFETTI
A1	100	100	100

Issue #	JQF-Zest	CONFETTI_tgt	CONFETTI
B1	100	0	0
B2	100	100	0
B3	0	0	40
B4	0	0	80
B5	0	5	100
B6	0	20	100

Issue #	JQF-Zest	CONFETTI_tgt	CONFETTI
R1	100	100	100
R2	100	100	100
R3	100	100	100
R4	100	100	100

Issue #	JQF-Zest	CONFETTI_tgt	CONFETTI
C1	100	100	100
C2	90	85	5
C3	80	70	45
C4	0	45	95
C5	0	15	90
C6	0	0	5
C7	0	20	100
C8	0	0	100
C9	15	15	20
C10	0	5	100
C11	0	0	100
C12	0	0	35
C13	0	0	20
C14	0	0	5
C15	0	0	5

Our evaluation, all data and CONFETTI are archived and open-source

https://doi.org/10.6084/m9.figshare.16563776

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