HotFuzz

Discovering Algorithmic Denial-of-Service Vulnerabilities through Guided Micro-Fuzzing

William Blair
Boston University

Andrea Mambretti
Northeastern University

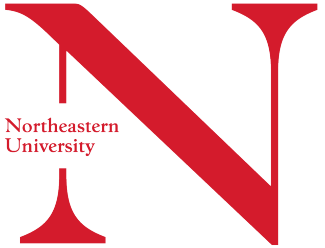
Sajjad Arshad
Northeastern University

Michael Weissbacher
Northeastern University

William Robertson
Northeastern University

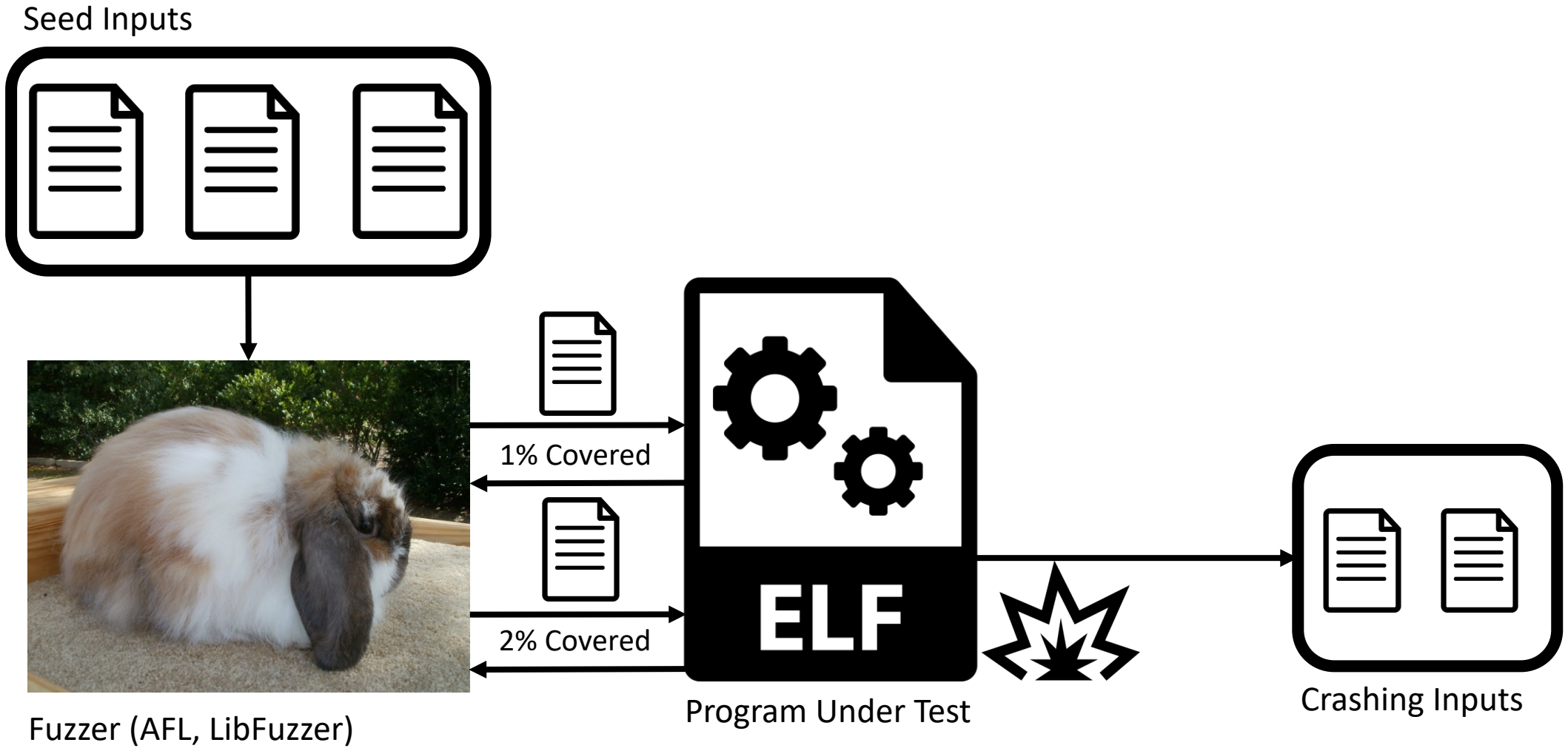
Engin Kirda
Northeastern University

Manuel Egele
Boston University



1988

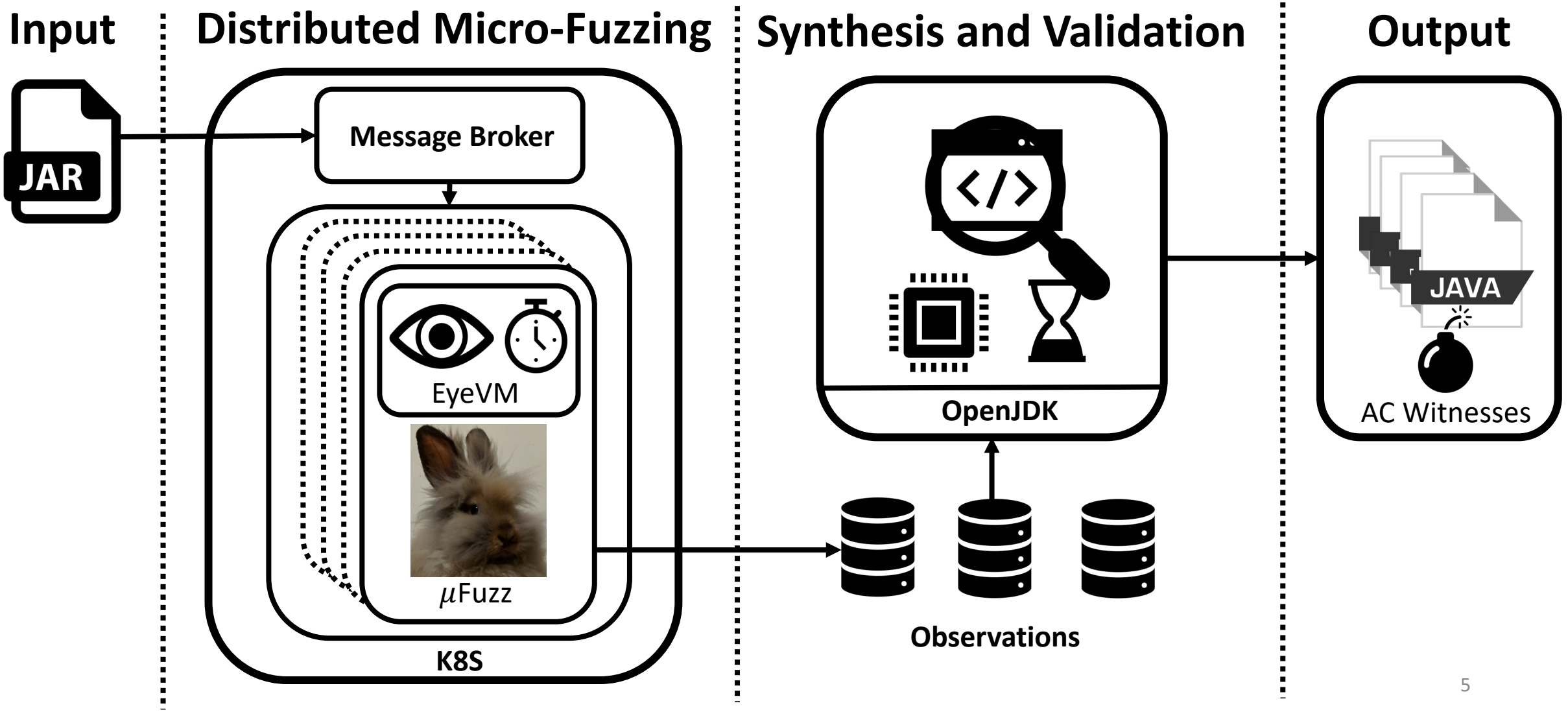
2020 Fuzz Testing



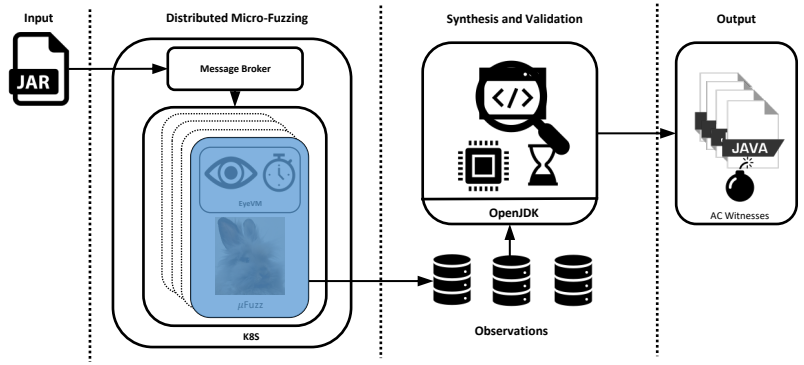
Algorithmic Complexity (AC) Bugs

We observed computing the total price of your cart can take 4 ½ months!

HotFuzz


Input
Distributed Micro-Fuzzing
Synthesis and Validation
Output
JAR
Message Broker
EyeVM
μFuzz
K8S
OpenJDK
Observations
JAVA
AC Witnesses

HotFuzz Micro-Fuzzing




```
class A {
  public method(B b, C c);
}
```

Micro-Fuzzing

```
a, b, c = TestHarness(method)
```

$$\texttt{a} \xleftarrow{R} \texttt{A}$$

$$\texttt{b} \xleftarrow{R} \texttt{B}$$

$$\texttt{c} \xleftarrow{R} \texttt{c}$$

```
a.method(b, c)
```

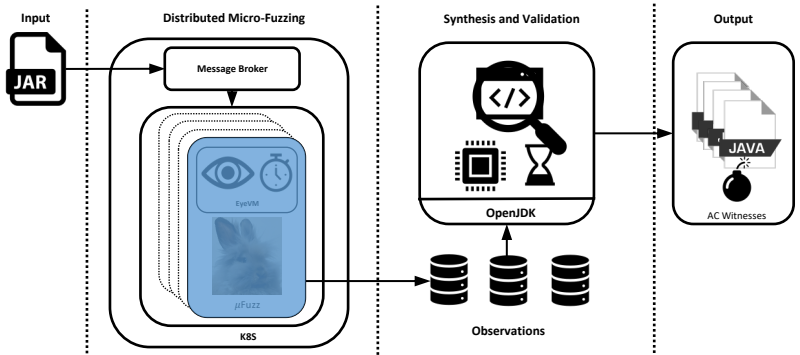
AC Sanitization

Threshold T

$$\texttt{Runtime(a.method(b, c))} \leq T$$

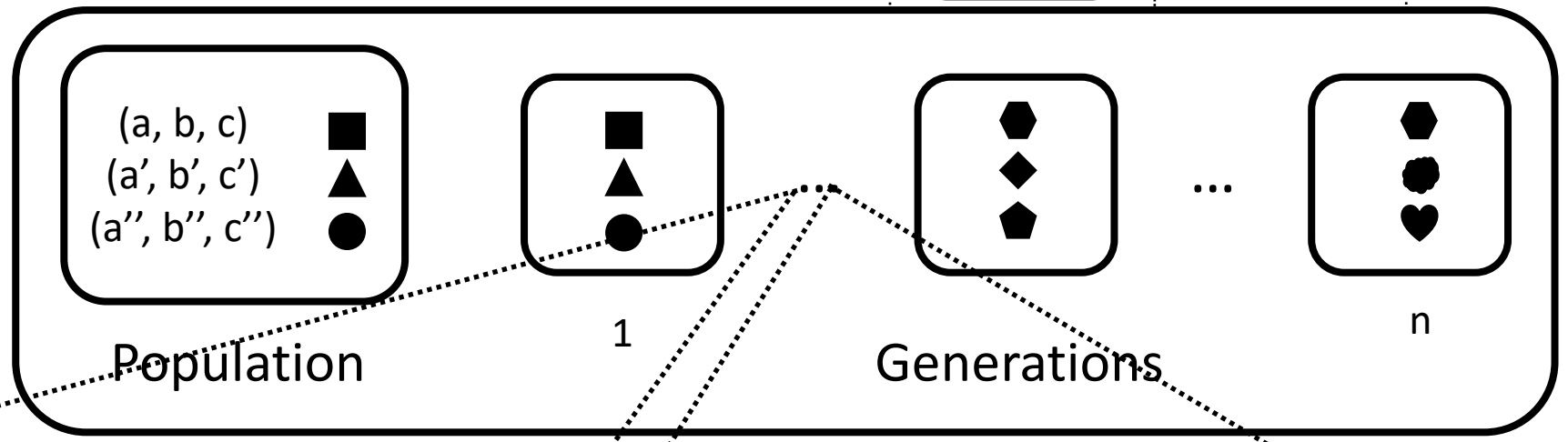
$$\texttt{Runtime(x.method(y, z))} > T$$

Micro-Fuzzing

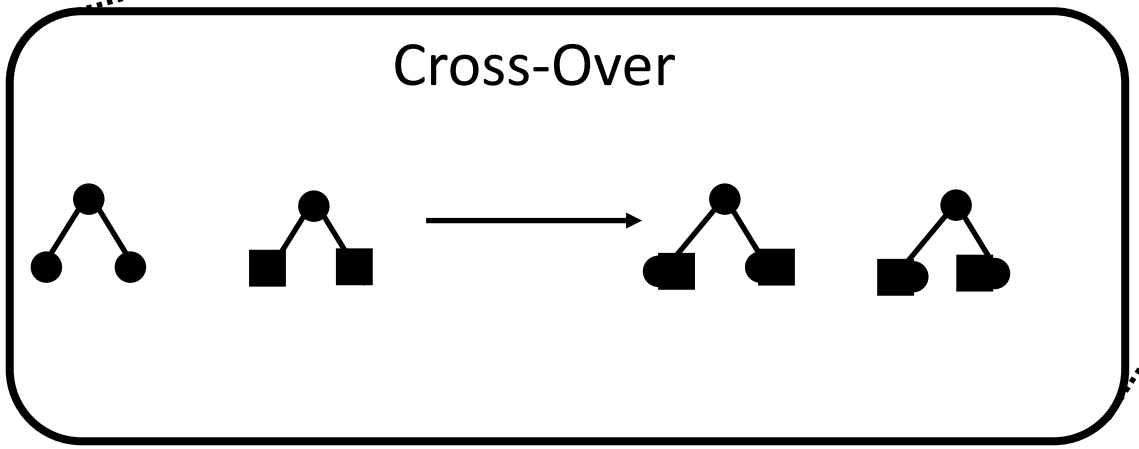


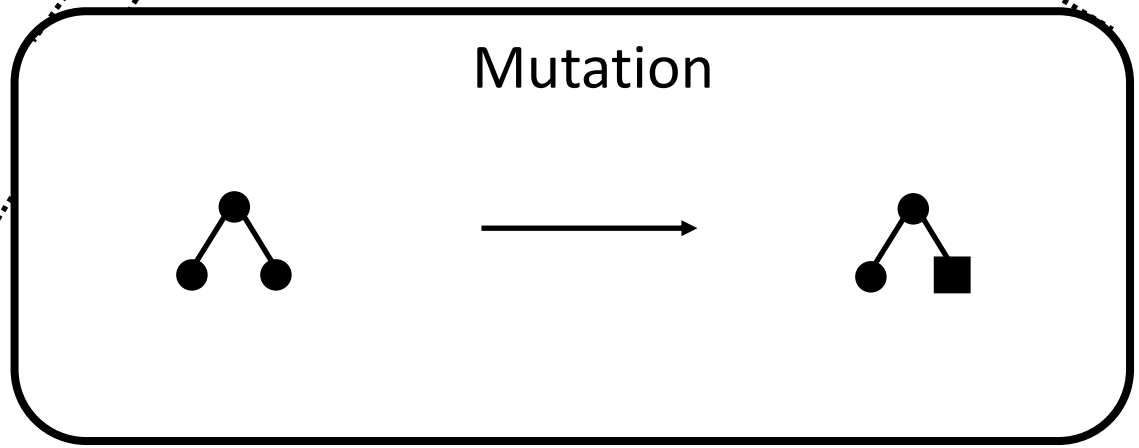
`A.method(B, C)`

Method Under Test

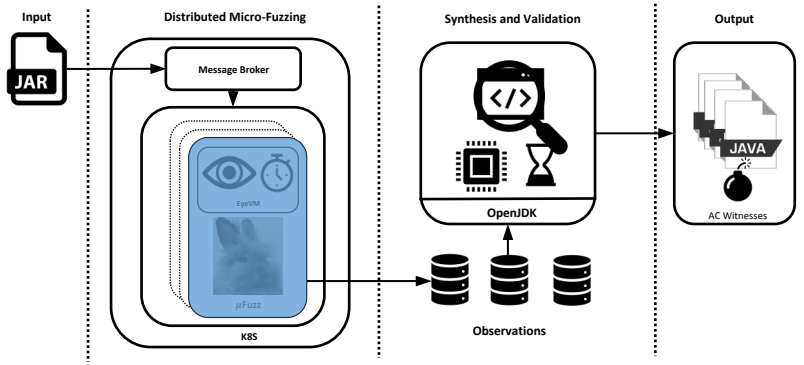


Genetic Algorithm





Instantiating Seed Inputs



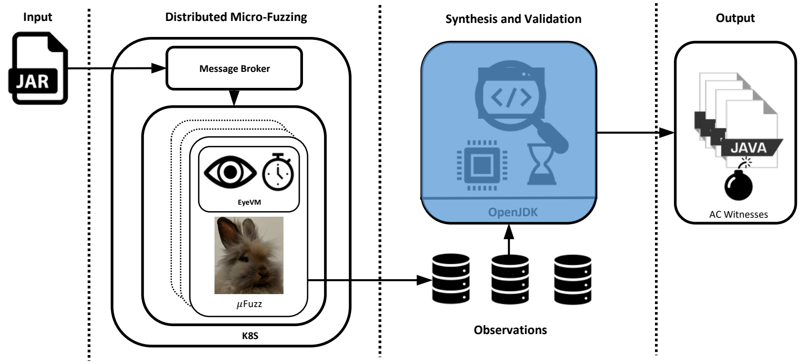
Identity Value Instantiation (IVI)

$$X = 0$$

Small Recursive Instantiation (SRI)

$$X \xleftarrow{R} \mathcal{N}(0, \frac{\alpha}{3})$$

```
new D(int)
new A(D, E)
a.method(b, c)
```



Synthesizing Test Cases

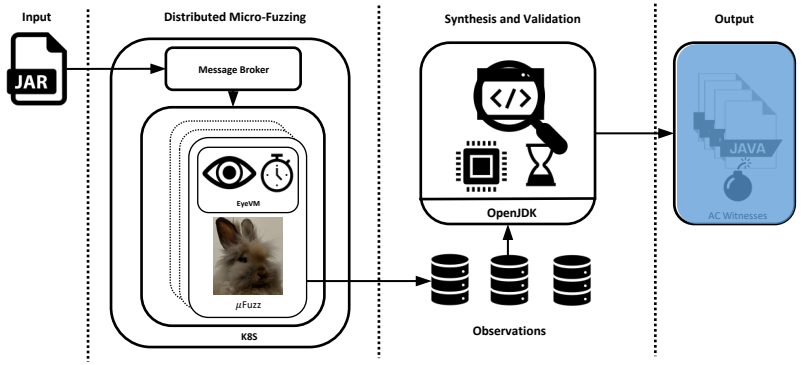
```
new D(10)    new E("a")

new A(D, E)
              new B(179)  new C(-1)

a.method(b, c)
public static void main(String argv[]){

}
```

Micro-Fuzzing Evaluation



Library	No. Methods	AC Bugs Detected			AC Bugs Confirmed			Methods Covered			Throughput	
		Both	IVI	SRI	Both	IVI	SRI	Both	IVI	SRI	IVI	SRI
JRE	91,632	6	8	13	5	8	13	23,818	2,780	1,439	4,389,675	3,092,866
STAC	67,494	34	6	15	5	0	0	8,064	847	1,162	3,608,741	3,172,502
Maven	239,777	46	38	56	46	38	56	66,987	2,622	1,770	5,906,687	5,591,106

AC Vulnerability in the JRE

```
import java.math.BigDecimal;

BigDecimal x = new BigDecimal(s);
BigDecimal y = new BigDecimal(t);

x.add(y);
```

If an adversary can influence the value of s or t, they can trigger DoS.

Computing

new BigDecimal(“1E2147483647”)).add(“1E0”);

Takes at least an hour to complete on every major implementation of the JVM!

Impact of BigDecimal Findings

- Affects all widely used JVM implementations
- Disclosed our findings to 3 vendors
- IBM J9
  - Proof of Concept (PoC) terminates after running for 4 ½ months
  - Issued us a CVE for our findings
- Oracle OpenJDK
  - PoC runs in an hour
  - Credited us in a Security-in-Depth Issue
- Google Android
  - PoC takes over 24 hours to run
  - Stated the issue falls outside their definition of DoS vulnerabilities

Summary

- Introduced Micro-Fuzzing
- Presented HotFuzz
  - Prototype implementation of micro-fuzzing for Java libraries
  - Automatically detects AC bugs
- Introduced strategies for generating seed inputs for micro-fuzzing
  - IVI … Identity Value Instantiation
  - SRI … Small Recursive Instantiation
- Micro-fuzzing detected 158 AC bugs in our evaluation artifacts
- Showed how an AC bug in production code can trigger DoS

Thank you!