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

Seed Inputs

Fuzzer (AFL, LibFuzzer)

Program Under Test

1% Covered
2% Covered

Crashing Inputs
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

Message Broker

EyeVM

μFuzz

K8S

OpenJDK

Observations

AC Witnesses

JAVA
HotFuzz Micro-Fuzzing

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

Micro-Fuzzing

a, b, c = TestHarness(method)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a ↩</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>b ↩</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>c ↩</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

a.method(b, c)

AC Sanitization

Threshold $T$

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

Runtime(x.method(y, z)) $> T$
Micro-Fuzzing

A.\texttt{method}(B, C)
Method Under Test

\begin{itemize}
\item \texttt{(a, b, c)}
\item \texttt{(a', b', c')}
\item \texttt{(a'', b'', c'')}
\end{itemize}

Input

\begin{itemize}
\item Distributed Micro-Fuzzing
\item Synthesis and Validation
\end{itemize}

Output

\begin{itemize}
\item AC Witnesses
\item OpenJDK
\item EyeVM
\end{itemize}

\(\mu\)Fuzz

Genetic Algorithm

Population

1

Generations

Cross-Over

Mutation
Instantiating Seed Inputs

Identity Value Instantiation (IVI)

\[ X = 0 \]

new D(int)

new A(D, E)

\[ \text{a.method}(b, c) \]

Small Recursive Instantiation (SRI)

\[ X \xleftarrow{R} \mathcal{N}(0, \frac{\alpha}{3}) \]
Synthesizing Test Cases

```java
public static void main(String argv[]){
    new D(10)  new E("a")
    new A(D, E)
    new B(179) new C(-1)
    a.method(b, c)
}
```
### Micro-Fuzzing Evaluation

<table>
<thead>
<tr>
<th>Library</th>
<th>No. Methods</th>
<th>AC Bugs Detected</th>
<th>AC Bugs Confirmed</th>
<th>Methods Covered</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Both</td>
<td>IVI</td>
<td>SRI</td>
<td>Both</td>
</tr>
<tr>
<td>JRE</td>
<td>91,632</td>
<td>6</td>
<td>8</td>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>STAC</td>
<td>67,494</td>
<td>34</td>
<td>6</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>Maven</td>
<td>239,777</td>
<td>46</td>
<td>38</td>
<td>56</td>
<td>46</td>
</tr>
</tbody>
</table>
import java.math.BigDecimal;

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

x.add(y);

Computing
new BigDecimal("1E2147483647").add("1E0");

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

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