Ankou: Guiding Grey-box Fuzzing towards Combinatorial Difference

Valentin Manès¹, Soomin Kim², Sang Kil Cha²

¹CSRC, KAIST ²KAIST
The Success of Grey-box Fuzzing

“OSS-Fuzz has found over 20,000 bugs in 300 open source projects.”

<table>
<thead>
<tr>
<th>Library</th>
<th>CVE-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>PoDoFo</td>
<td>CVE-2017-5886</td>
</tr>
<tr>
<td>GStreamer</td>
<td>CVE-2016-10198</td>
</tr>
<tr>
<td>GStreamer</td>
<td>CVE-2016-10199</td>
</tr>
<tr>
<td>GStreamer</td>
<td>CVE-2016-10200</td>
</tr>
<tr>
<td>GStreamer</td>
<td>CVE-2016-10201</td>
</tr>
<tr>
<td>ZZIPLib</td>
<td>CVE-2017-5970</td>
</tr>
<tr>
<td>ZZIPLib</td>
<td>CVE-2017-5980</td>
</tr>
<tr>
<td>ZZIPLib</td>
<td>CVE-2017-5981</td>
</tr>
<tr>
<td>ZZIPLib</td>
<td>CVE-2015-8985</td>
</tr>
</tbody>
</table>

Why one more?
Grey-box, How?

**Fitness Function:**

```python
if("interesting"):
    Add to seed pool
```

Diagram:
- **Fuzzer**
  - Seed Pool
    - Test Case A
    - Test Case B
    - Test Case C
- **Program**
- **Output**
- **Test Case**
- **Test Case**
Which Feedback?

Coverage has proved a good tradeoff between cost and benefits.

Ankou: Opportunity to improve?

Cost

- czuf
- BFF
...  - AFL
- LibFuzzer
...  - Vuzzer
- Angora
...
Coverage-Based Fuzzing

```c
int combinedBranches(char *data) {
    int bits = 0;
    if (data[0] == 'A') bits |= 1;
    if (data[1] == 'B') bits |= 2;
    if (data[2] == 'C') bits |= 4;
    if (bits == 7)
        printf("BINGO\n");
    return 0;
}
```

<table>
<thead>
<tr>
<th>Test Case</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>“A”</td>
<td>“BB”</td>
<td>“AB”</td>
<td>“ABC”</td>
</tr>
<tr>
<td>Branch 1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Branch 2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Branch 3</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fuzzer

Seed Pool

Test Case A

Test Case B

Test Case D

Program

Test Case Outputs

Fitness Function:

if (new branch):
    Add to seed pool

A more informative Fitness Function is needed!
Informative Fitness with Combination

Ankou goal: developing a fitness function taking into account combinations.

1. **Quantify** the difference between program executions.
2. Make fitness computation **fast**.
3. Make the fitness **adaptive** to the program.
Point Representation

Branch 1

Branch 2

0 1 2 3 4 5 6

0 1 2 3
Distance between Executions

Branch 2

Branch 1

Euclidean Distance
Distance between Executions

Detects Combinatorial Difference!
Distance-based Fitness Function

Seed Pool

? Point-to-Pool ?
Distance-based Fitness Function

Seed Pool

Point-to-Pool = Minimum Point-to-Point
Cost Sensitivity

The fitness function is computed for every test case.
Problem: Slow Computation

Euclidean Distance = $\mathcal{O}(\#\text{branch})$
Cost Reduction

Euclidean Distance = $\mathcal{O}(\#\text{branch})$

Dimensionality Reduction

Euclidean Distance = $\mathcal{O}(\#\text{"representative branch"})$

See paper for details on the Dynamic PCA.
Ankou Adaptive Fitness Function

Ankou fitness function:

if(new branch):
    if(Point-to-Pool distance ??):
        Add test to seed pool
Ankou Adaptive Fitness Function

Ankou fitness function:

```python
if(new branch):
    if(Point-to-Pool distance > \( \theta_{fit} \)):
        Add test to seed pool
        \( \theta_{fit} \leftarrow \text{Minimum inter-seed distance} \)
```
Benchmark

• Use 24 packages used by CollAFL\textsuperscript{1}.

• All experiments are 6x24 hours runs.

• In total: our experiments constitute 2,682 CPU days.

Q: Is the New Fitness Function Effective?
Ankou with and without Distance-based

Distance-based finds 44% more crashes.

<table>
<thead>
<tr>
<th>Distance</th>
<th># Crashes</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Crash ratio (in log) | Throughput ratio (in log)
Q: How does Ankou compare to other grey-box fuzzers?
Ankou vs. AFL

Ankou finds 41% more unique crashes.
Ankou vs. AFL: Speed

Ankou is 35% slower than AFL.
Conclusion

1. Coverage-based fuzzers ignore **combinations** of branches.

2. Ankou **distance-based** fitness function **quantify** combinatorial difference while being **fast** and **adaptive** to programs.

3. While being 35% slower than AFL, Ankou finds 41% more crashes.
Question?