Arvin: Greybox Fuzzing Using Approximate Dynamic CFG Analysis

Sirus Shahini, Mu Zhang, Mathias Payer, Robert Ricci
Introduction: Fuzz Testing

• Fuzz testing: The most popular method to discover bugs
• Running programs with random inputs, looking for crashes
• Thousands of security critical bugs in the past few years
• The core challenge:
  • We must find the right input(s) ...
  • There are infinite number of inputs...
Arvin's Contributions

• A new way of **instrumenting** programs under test
• Better **context** for analysis of dynamic program behavior
• Better program **coverage, faster**
• Better **bug finding**: 50 bugs in 17 programs
Background

• Time is GOLD
• High-level goal: good coverage and many crashes
  • Usually measured in basic blocks or edges
  • Grey-box fuzzing: limited visibility into program under test
• Better inputs : better coverage
• Prioritization makes a big difference
• Understanding program behavior requires context awareness
Arvin: Context-Aware Fuzzing

- Understanding the PUT’s context using control-flow graphs
- Dynamic control-flow graphs at basic block level
- Most inputs end very close to the PUT’s entry point
  - Getting farther from entry and closer to more interesting areas
- Prioritize inputs that are *high quality*
  - First, get high coverage
  - Then, heavily exercise those inputs
Example
Example
Building Dynamic Graphs

• Graphs are complex – most fuzzers use only bitmaps
• Building DCFGs at runtime is challenging for a greybox fuzzer
• Arvin’s core instrumentation: The DCFG runtime library
  • Instruments PUT in-memory
• Independent and nested basic blocks
  • Treat call instructions differently from most fuzzers
  • For Arvin, they start a new nested basic block
Building Dynamic Graphs: Nested Blocks

<table>
<thead>
<tr>
<th>Transitions</th>
<th>Without nested blocks</th>
<th>With nested blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A branches B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B branches A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A branches C</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ii</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A calls B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B returns A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A branches C</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iii</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A calls B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B branches C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C returns A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A branches D</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Calculating Priorities From DCFGs

• New coverage
  • Not just new blocks, but new edges between them

• Depth
  • Get far from PUT entry point

• Target specific basic blocks
  • Representing potentially vulnerable functions
  • ... and specific paths to reach them
Balancing the Priorities

A) Input queue without prioritization*

```
H L H L L L H L L L L H H
```
Offsetting the Analysis Load

• Millions of executions for a single bug
• We need to reduce the cost of graph analysis
• Some nodes are not needed in later iterations
• Arvin makes use of directed approximation
  • Strategy: Remove instrumentation for blocks that run the most
  • Reduces instrumentation and analysis cost
  • They were generally not giving much information anyway
  • Remember them between runs of the PUT
DCFG Approximation

• Arvin’s shrink-grow cycle: Decremental CFG Growth (DCG)
• Hit counts matter: DHT
• Shrink: Choose candidate nodes for exclusion
• Grow: Discover and add new nodes
• Save time to invest on more efficient coverage growth
Example: Shrink and Grow

- Total size: 5
- Covered nodes: 5

SHRINK

- Total size: 4
- Excluded nodes: 1
Varying Thresholds
Adaptive Mutation

- Deterministic and Non-deterministic mutations
- Deterministic (IT)
  - Pick a good input
  - Try all of our mutations on it
  - Tends to find crashes
- Random (NI)
  - Explore inputs with random muts.
  - Tends to expand coverage
- Adjust rates over time
Evaluation: Testing coverage growth
Evaluation: Depth
Evaluation: Approximation

![Graph showing speedup vs level with different tools and their speedup on a log-log scale.](image)
## Discovered bugs

<table>
<thead>
<tr>
<th>Program</th>
<th>Arvin</th>
<th>AFL</th>
<th>hf</th>
<th>hf-qemu</th>
<th>APP</th>
<th>APP-qemu</th>
<th>TortoiseFuzz</th>
<th>Angora*</th>
<th>VUzzer</th>
<th>ParmeSan</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNU assembler (v 2.35, 2.36, 2.37)</td>
<td>13</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>unzip (v 6.00)</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>gif2png (v 2.5.8-1)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>readelf (v 2.35)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bison (v 3.5, 3.7)</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>fig2dev (v 3.2.8a)</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>flvmeta (v 1.2.2)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>nasm (v 2.15.05)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ibmtpm (last commit May 15 2021)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>libraw (v 0.21.0)</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pnmtopng (v 1.17.dfs-g-4)</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pnmtopjpeg (v 1.17.dfs-g-4)</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pnmtoglasc (v 1.17.dfs-g-4)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>pstopnm (v 1.17.dfs-g-4)</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>psnup (v 1.17.dfs-g-4)</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>xpdf (v 4.03)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>xvid (last version from <a href="http://svn.xvid.org/trunk">http://svn.xvid.org/trunk</a>)</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>50</strong></td>
<td><strong>11</strong></td>
<td><strong>23</strong></td>
<td><strong>13</strong></td>
<td><strong>27</strong></td>
<td><strong>16</strong></td>
<td><strong>16</strong></td>
<td><strong>6</strong></td>
<td><strong>0</strong></td>
<td><strong>5</strong></td>
</tr>
</tbody>
</table>
Related Work

• angr framework [Shoshitaishvili et al, IEEE S&P '16]
  • We use it to find basic blocks, etc.

• Inspiration
  • AFL [Zalewski], AFL++ [Fioraldi et al, WOOT '20], and others

• Fuzzing with CFGs
  • ParmeSan [Osterlund et al, USENIX Security '22]

• Fuzzing with adaptive instrumentation
  • Full Speed Fuzzing [Nagy et al, IEEE S&P '19]
More In The Paper

• Parallel fuzzing
• Detailed examples
• More evaluation
Try Arvin

• Contact: sirus.shahini@gmail.com

• Open source: https://github.com/0xsirus/arvin
Questions?
Example
Exclusion in Detail

Identify and exclude

Missed transitions after exclusion

Approximated transitions after exclusion
Parallel Fuzzing

- Separate instrumentation libraries for different fuzzing modes
- Cooperative parallel fuzzing
Evaluation: Approximation
Evaluation: Parallel fuzzing