FairFuzz: A Targeted Mutation Strategy for Increasing Greybox Fuzz Testing Coverage

Caroline Lemieux, Koushik Sen
University of California, Berkeley

source: https://github.com/carolemieux/afl-rb
The Rise of Fuzz Testing

- Programs still have bugs.
- *Fuzz testing* has become very popular in practice and theory
Fuzzing in One Slide

Fuzzer

Input

Program

Feedback
Fuzzing in One Slide

Fuzzer  →  Input  →  Program  →  Feedback

Input:
AVERY_BAD_INPUT
Fuzzing in One Slide

Fuzzer

Program

Input

Feedback

A VERY BAD_INPUT
Fuzzing in One Slide

The input: AVERY_BAD_INPUT causes a crash.
What Bugs Can Fuzzing Find?

• Most popular: basic correctness assertions (C/C++)
  • Segmentation faults
  • Anything address sanitizer can catch:
    • Buffer overflows
    • Use-after-frees
    • Etc…
Coverage-Guided (Greybox) Mutational Fuzzing
Coverage-Guided (Greybox) Mutational Fuzzing

- Fuzzer
- Saved inputs
- Input
- Program
- Feedback (branches covered)

Mutate saved input
Coverage-Guided (Greybox) Mutational Fuzzing

- Fuzzer
  - Saved inputs
  - Mutate saved input
  - New coverage? Save.

- Program
  - Feedback
    - (branches covered)
Observation: some parts of the program easier to cover

```c
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
    if (fuzzed_data_len >= 10) {
        // more code
    }
    // ...
    if (starts_with(fuzzed_data, "<!ATTLIST") { // ...
        // ...
    }
    // ...
    return process_result;
}
```
Observation: some parts of the program easier to cover

```
int process_xml(char * fuzzed_data,
               int fuzzed_data_len) {
    if (fuzzed_data_len >= 10) {
        // more code
    }
    // ...
    if (starts_with(fuzzed_data, "<!ATTLIST")){
        // ...
    }
    // ...
    return process_result;
}
```
**Observation:** some parts of the program easier to cover

```
int process_xml(char * fuzzed_data,  
                int fuzzed_data_len) {
    if (fuzzed_data_len >= 10) {
        // more code
    }
    // ...
    if (starts_with(fuzzed_data, "<!ATTLIST") ){
        // ...
    }
    // ...
    return process_result;
}
```
Observation: some parts of the program easier to cover

```c
int process_xml(char * fuzzed_data,
                 int fuzzed_data_len) {
    if (fuzzed_data_len >= 10) {
        // more code
    }
    // ...
    if (starts_with(fuzzed_data, "<!ATTLIST") { 
        // ...
    }
    // ...
    return process_result;
}
```
What’s Missing? Uneven Fuzzing Coverage

**Observation:** some parts of the program easier to cover

```c
int process_xml(char * fuzzed_data,  
                 int fuzzed_data_len) {  
    if (fuzzed_data_len >= 10) {  
        // more code  
    }  
    // ...  
    if (starts_with(fuzzed_data, "<!ATTLIST"))){  
        // ...  
    }  
    // ...  
    return process_result;  
}
```

- **Hit by 100k+ inputs**  
  → Code under if well-covered

- **Hit by 1 input**  
  → Code under if *barely* covered
Uneven Fuzzing Coverage ➔ Uncovered Code

**Observation:** some parts of the program easier to cover

- Hit by 100k+ inputs ➔ Code under if well-covered
- Hit by 1 input ➔ Code under if barely covered

**Result:** some functionality wholly uncovered by fuzzing

```c
int process_xml(char * fuzzed_data, int fuzzed_data_len) {
    if (fuzzed_data_len >= 10) {
        // more code
        // ...
        if (starts_with(fuzzed_data, "<!ATTLIST")) {
            if (starts_with(&fuzzed_data[10], "ID")) {
                // lots more processing code
            }
        }
        // ...
        return process_result;
    }
}
```
Some branches hard to hit by naively mutated inputs

```c
int process_xml(char * fuzzed_data, int fuzzed_data_len) {
    if (fuzzed_data_len >= 10) {
        // more code
    }
    // ...
    if (starts_with(fuzzed_data, "<!ATTLIST")){
        if (starts_with(&fuzzed_data[10], "ID")) {
            // lots more processing code
        }
    }
    // ...
    return process_result;
}
```

Input satisfying if: AT_LEAST_10_BYTES
Some branches hard to hit by naively mutated inputs

Input satisfying if:

- AT_LEAST_10_BYTES
- BT_LEAST_10_BYTES
- AT_LEAST_10_BYS
- AT???_LEAST_10_BYTES

```c
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
    if (fuzzed_data_len >= 10) {
        // more code
    }
    // ...
    if (starts_with(fuzzed_data, "<!ATTLIST"){
        if (starts_with(&fuzzed_data[10], "ID")) {
            // lots more processing code
        }
    }
    // ...
    return process_result;
}
```
Some branches hard to hit by naively mutated inputs

```c
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
    if (fuzzed_data_len >= 10) {
        // more code
    }
    // ...
    if (starts_with(fuzzed_data, "<!ATTLIST"){
        if (starts_with(&fuzzed_data[10], "ID")}) {
            // lots more processing code
        }
    }
    // ...
    return process_result;
}
```

Input satisfying if:

```xml
<!ATTLIST BD
```
Why So Uneven?

Some branches hard to hit by naively mutated inputs

Input satisfying if:

`<!ATTLIST BD`

`<?ATTLIST BD`

`<!ATTLIST BD`

`<!CATLIST BDD`

```c
int process_xml(char * fuzzed_data, int fuzzed_data_len) {
    if (fuzzed_data_len >= 10) {
        // more code
    }
    // ...
    if (starts_with(fuzzed_data, "<!ATTLIST")) {
        if (starts_with(&fuzzed_data[10], "ID")) {
            // lots more processing code
        }
    }
    // ...
    return process_result;
}
```
Our Method: FairFuzz

Utilize existing greybox info
To target rarely-exercised code → increase coverage
Our Method: FairFuzz

Utilize existing greybox info
To target rarely-exercised code \rightarrow increase coverage

**Identify**: branches hit by few inputs (rare branches)
Our Method: FairFuzz

Utilize existing greybox info
To target rarely-exercised code → increase coverage

**Identify**: branches hit by few inputs (rare branches)

**Identify**: where input can be mutated and hit branch
Method
Recap: AFL

• AFL [1]: Popular coverage-guided greybox fuzzer
• Fuzzes programs taking in file or stdin
• Easy to use (just compile program with afl-gcc or afl-clang)
• Has found many bugs in practice

AFL Method

Seeds
AFL Method

Seeds

Parent Input Set
AFL Method

Seeds

Parent Input Set

Select parent to mutate

Parent Input
AFL Method

Seeds

Parent Input Set

Select parent to mutate

Parent Input

Create mutants

Mutant

Parent Input Set

Select parent to mutate

Parent Input

Create mutants

Mutant
AFL Method

1. Seeds
2. Parent Input Set
3. Select parent to mutate
4. Parent Input
5. Create mutants
6. Mutant
7. Run on instrumented program
8. Feedback
AFL Method

Seeds

Parent Input Set → Select parent to mutate → Parent Input → Create mutants → Mutant

Interesting Mutant → Get mutants adding new coverage → Feedback → Run on instrumented program
AFL Method

Seeds

Parent Input Set

Select parent to mutate

Parent Input

Create mutants

Mutant

Add to parent set

Interesting Mutant

Get mutants adding new coverage

Feedback

Run on instrumented program
AFL Method

Seeds → Parent Input Set

Parent Input Set → Select parent to mutate

Select parent to mutate → Parent Input

Parent Input → Create mutants

Create mutants → Mutants

Mutants → Seeds

Seeds → Add to parent set

Add to parent set → Interesting Mutant

Interesting Mutant → Get mutants adding new coverage

Get mutants adding new coverage → Feedback

Feedback → Run on instrumented program

Run on instrumented program → Interesting Mutant

Interesting Mutant → Add to parent set
AFL Mutation Types

- Fixed-location mutations
  - Choose mutation type, apply at all locations in input
  - Mutation types: byte flips, arithmetic inc/dec, replacing with “interesting” values, etc.
AFL Mutation Types

- Fixed-location mutations
  - Choose mutation type, apply at all locations in input
  - Mutation types: byte flips, arithmetic inc/dec, replacing with “interesting” values, etc.

\[
\text{< !ATTLIST BD} \\
\text{0 0 ATTLIST BD} \\
\text{< 00 TTLIST BD} \\
\text{< !0 0 TLIST BD} \\
\text{...}
\]
AFL Mutation Types

• Fixed-location mutations
  • Choose mutation type, apply at all locations in input
  • Mutation types: byte flips, arithmetic inc/dec, replacing with “interesting” values, etc.

• Random-location mutation
  • Repeat: choose random mutation, apply at random location

\[
\begin{align*}
0\ 0\ \texttt{ATTLIST}\ BD \\
<\texttt{ATTLIST}\ BD \\
<\texttt{00TLIST}\ BD \\
<\texttt{00TLIST}\ BD \\
\end{align*}
\]
AFL Mutation Types

• Fixed-location mutations
  • Choose mutation type, apply at all locations in input
  • Mutation types: byte flips, arithmetic inc/dec, replacing with “interesting” values, etc.

• Random-location mutation
  • Repeat: choose random mutation, apply at random location
FairFuzz Method – Key Differences

1. **Parent Input Set**
   - Parent Input Set
   - Select parent to mutate
   - Parent Input
   - Compute mask
   - Create mutants
   - Mutant
   - Run on instrumented program
   - Feedback
   - Get mutants adding new coverage
   - Interesting Mutant
   - Add to parent set
   - Interesting

2. **Seeds**
   - Seeds
   - Parent Input Set

3. **Key Differences**
   - University of California, Berkeley
   - FairFuzz, presented by Caroline Lemieux
FairFuzz Method – Selecting Parent Inputs

Seeds

Parent Input Set

Select parent to mutate

Parent Input

Compute mask
Create mutants

Mutant

Run on instrumented program

Get mutants adding new coverage

Interesting Mutant

Interesting Mutant

Interesting Mutant

Add to parent set
FairFuzz Method – Selecting Parent Inputs

- Keep track of # of inputs produced exercising each branch
- Pick inputs that exercise a branch hit by relatively few inputs
- Rarest branch hit: target branch
FairFuzz Method – Selecting Parent Inputs

- Keep track of # of inputs produced exercising each branch
- Pick inputs that exercise a branch hit by relatively few inputs
- Rarest branch hit: target branch

```c
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
    // more code
  }
  // ...
  if (starts_with(fuzzed_data, "<!ATTLIST")) {
    if (starts_with(&fuzzed_data[10], "ID")) {
      // lots more processing code
    }
    // ...
  return process_result;
}
```
FairFuzz Method – Selecting Parent Inputs

- Keep track of # of inputs produced exercising each branch
- Pick inputs that exercise a branch hit by relatively few inputs
- Rarest branch hit: target branch

```c
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
    if (fuzzed_data_len >= 10) {
        // more code
    }
    // ...
    if (starts_with(fuzzed_data, "<!ATTLIST")) { // AT_LEAST_10_BYTES
        if (starts_with(&fuzzed_data[10], "ID")) {
            // lots more processing code
        }
        // ...
    }
    return process_result;
}
```
**FairFuzz Method – Computing Branch Mask**

1. **Seeds**
2. **Parent Input Set** → **Select parent to mutate** → **Parent Input** → **Compute mask** → **Create mutants** → **Mutant**
3. **Add to parent set**
4. **Get mutants adding new coverage** → **Interesting Mutant** → **Run on instrumented program**
5. **Feedback**
FairFuzz Method – Computing Branch Mask

• Easily integrated with fixed-location mutation phases of fuzzers
• Flip each byte, check if mutated input still hits target branch
FairFuzz Method – Computing Branch Mask

- Easily integrated with fixed-location mutation phases of fuzzers
- Flip each byte, check if mutated input still hits target branch

```c
int process_xml(char * fuzzed_data, int fuzzed_data_len) {
    if (fuzzed_data_len >= 10) {
        // more code
    }
    // ...
    if (starts_with(fuzzed_data, "<!ATTLIST")) {
        if (starts_with(&fuzzed_data[10], "ID")) {
            // lots more processing code
        }
    }
    // ...
    return process_result;
}
```
FairFuzz Method – Computing Branch Mask

• Easily integrated with fixed-location mutation phases of fuzzers
• Flip each byte, check if mutated input still hits target branch

Parent input `< !ATTLIST BD hits`

```c
int process_xml(char * fuzzed_data, int fuzzed_data_len) {
    if (fuzzed_data_len >= 10) {
        // more code
    }
    // ...
    if (starts_with(fuzzed_data, "<!ATTLIST")){
        if (starts_with(&fuzzed_data[10], "ID")) {
            // lots more processing code
        }
        // ...
        return process_result;
    }
}```
FairFuzz Method – Computing Branch Mask

- Easily integrated with fixed-location mutation phases of fuzzers
- Flip each byte, check if mutated input still hits target branch

Parent input `<!ATTLIST BD` hits

X`<!ATTLIST BD`
FairFuzz Method – Computing Branch Mask

- Easily integrated with fixed-location mutation phases of fuzzers
- Flip each byte, check if mutated input still hits target branch

```c
int process_xml(char * fuzzed_data, int fuzzed_data_len) {
    if (fuzzed_data_len >= 10) {
        // more code
    }
    // ...
    if (starts_with(fuzzed_data, "<!ATTLIST")){
        if (starts_with(&fuzzed_data[10], "ID")) {
            // lots more processing code
        }
    }
    // ...
    return process_result;
}
```
FairFuzz Method – Computing Branch Mask

- Easily integrated with fixed-location mutation phases of fuzzers
- Flip each byte, check if mutated input still hits target branch

```c
int process_xml(char * fuzzed_data, int fuzzed_data_len) {
    if (fuzzed_data_len >= 10) {
        // more code
    }
    // ...
    if (starts_with(fuzzed_data, "<!ATTLIST")) {
        if (starts_with(&fuzzed_data[10], "ID")) {
            // lots more processing code
        }
    }
    // ...
    return process_result;
}
```
FairFuzz Method – Computing Branch Mask

- Easily integrated with fixed-location mutation phases of fuzzers
- Flip each byte, check if mutated input still hits target branch

```c
int process_xml(char * fuzzed_data, int fuzzed_data_len) {
    if (fuzzed_data_len >= 10) {
        // more code
    }
    // ...
    if (starts_with(fuzzed_data, "<!ATTLIST")) {
        if (starts_with(&fuzzed_data[10], "ID")) {
            // lots more processing code
        }
    }
    // ...
    return process_result;
}
```
FairFuzz Method – Computing Branch Mask

- Easily integrated with fixed-location mutation phases of fuzzers
- Flip each byte, check if mutated input still hits target branch

```
int process_xml(char * fuzzed_data, int fuzzed_data_len) {
    if (fuzzed_data_len >= 10) {
        // more code
    }
    // ...
    if (starts_with(fuzzed_data, "<!ATTLIST") {\n        if (starts_with(&fuzzed_data[10], "ID")) {\n            // lots more processing code
        }\n        // ...
        return process_result;
    }
```
FairFuzz Method – Computing Branch Mask

- Easily integrated with fixed-location mutation phases of fuzzers
- Flip each byte, check if mutated input still hits target branch

Parent input: `<!ATTLIST BD` hits

Mask: `<!ATTLIST BD`

```c
int process_xml(char * fuzzed_data, int fuzzed_data_len) {
    if (fuzzed_data_len >= 10) {
        // more code
    }
    // ...
    if (starts_with(fuzzed_data, "<!ATTLIST")){
        if (starts_with(&fuzzed_data[10], "ID")) {
            // lots more processing code
        }
    }
    // ...
    return process_result;
}
```
FairFuzz Method – Targeting Mutations

**Seeds**

- **Parent Input Set**
  - Select parent to mutate
  - Parent Input
  - Compute mask
  - Create mutants
  - Mutant

- **Add to parent set**
  - Interesting Mutant
  - Get mutants adding new coverage
  - Feedback
  - Run on instrumented program

- **Interesting Mutant**
  - Add to parent set
FairFuzz Method – Targeting Mutations

• Fixed-location mutation
  • Don’t produce mutants at locations in mask

• Random-location mutation
  • Choose random locations outside mask
FairFuzz Method – Targeting Mutations

• Fixed-location mutation
  • Don’t produce mutants at locations in mask

• Random-location mutation
  • Choose random locations outside mask
FairFuzz Method – Targeting Mutations

• Fixed-location mutation
  • Don’t produce mutants at locations in mask

• Random-location mutation
  • Choose random locations outside mask
FairFuzz Method – Targeting Mutations

• Fixed-location mutation
  • Don’t produce mutants at locations in mask

• Random-location mutation
  • Choose random locations outside mask
Evaluation
Evaluation – Tools Compared

- **FairFuzz**: our tool, with highest-performing settings
- **AFL**: vanilla AFL, default settings
- **FidgetyAFL**: AFL with highest-performing settings
- **AFLFast.new**: AFLFast with highest-performing settings

Evaluation - Benchmarks

djpeg
readpng
mutool draw
xmlint
tcpdump
c++filt
objdump
readelf
nm

FidgetyAFL benchmarks
More complex input structures
AFLFast benchmarks
Evaluation Setup

For each benchmark:
• Run each technique 24hrs
• Start with 1 valid seed file
• No dictionaries
• Repeat runs 20x
  • Calculated confidence intervals
Summary Results – Coverage Leaders

![Graph showing coverage leaders over time](image)

- **AFL**
- **FidgetyAFL**
- **AFLFast.new**
- **FairFuzz**

Higher is better

Time (hrs): 0, 5, 10, 15, 20, 25

Number of benchmarks: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
FairFuzz achieves the highest coverage fast, for nearly all benchmarks
Branch Coverage Over Time

(a) tcpcap
(b) readelf
(c) nm
(d) objdump
(e) c++filt
(f) xmllint
(g) muttool draw
(h) djpepg
(i) readpng
Where Does FairFuzz Perform Much Better?

Both are programs with nested conditional structure:
- tcpdump: if this packet type, then if has this field...
- xmllint: byte-by-byte comparisons
Where Doesn’t FairFuzz Perform As Well?

C++ name demangler: highly recursive-structured program

• Covering different branches may not be best exploration strategy
Conclusion

Why So Uneven?

Some branches hard to hit by naively mutated inputs

```
int process_xml(char * fuzzed_data,
    int fuzzed_data_len) {
    if (fuzzed_data_len > 10) { // more code }
    if (starts_with(fuzzed_data, "<ATTLIST")
    if (starts_with(fuzzed_data[10], "ID") { // lots more processing code }
    // ...
    return process_result;
}
```

Summary Results – Coverage Leaders

- FairFuzz achieves the highest coverage fast, for nearly all benchmarks

FairFuzz Method – Computing Branch Mask

- Flip each byte, check if mutated input still hits target branch

```
int process_xml(char * fuzzed_data,
    int fuzzed_data_len) {
    if (fuzzed_data_len > 10) { // more code }
    if (starts_with(fuzzed_data, "<ATTLIST")
    if (starts_with(fuzzed_data[10], "ID") { // lots more processing code }
    // ...
    return process_result;
}
```

- Easily integrated with fixed-location mutation phases of fuzzers

Where Does FairFuzz Perform Much Better?

- Both are programs with nested conditional structure
  - tcpdump: if this packet type, then if has this field...
  - xmlint: byte-by-byte comparisons

Branch Mask Performance

For a subset of benchmarks, run a cycle with “shadow run”:

- For each selected input, create mutants
  - (1) without branch mask
  - (2) without branch mask
- Compare % of inputs hitting target branch:
  - Average over all inputs selected for mutation in cycle
Branch Mask Performance

- Mask substantially increases % of inputs hitting target branch

<table>
<thead>
<tr>
<th></th>
<th>Fixed-Location Mutants</th>
<th>Random-Location Mutants</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>With Mask</td>
<td>Without Mask</td>
</tr>
<tr>
<td>xmlint</td>
<td>90.3%</td>
<td>22.9%</td>
</tr>
<tr>
<td>tcpdump</td>
<td>98.7%</td>
<td>72.8%</td>
</tr>
<tr>
<td>c++filt</td>
<td>96.6%</td>
<td>14.8%</td>
</tr>
<tr>
<td>readelf</td>
<td>99.7%</td>
<td>78.2%</td>
</tr>
<tr>
<td>readpng</td>
<td>97.8%</td>
<td>39.0%</td>
</tr>
<tr>
<td>objdump</td>
<td>99.2%</td>
<td>66.7%</td>
</tr>
</tbody>
</table>