Lecture 17: Fuzz Testing

17-355/17-655/17-819: Program Analysis
Rohan Padhye and Jonathan Aldrich
April 1, 2021

* Course materials developed with Claire Le Goues
Puzzle: Find $x$ such that $p1(x)$ returns True

def p1(x):
    if $x^2 - 10 == 15$:
        return True
    return False
Puzzle: Find $x$ such that $p2(x)$ returns True

def p2(x):
    if x > 0 and x < 1000:
        if ((x - 32) * 5/9 == 100):
            return True
    return False
Puzzle: Find x such \( p3(x) \) returns True

def p3(x):
    if x > 3 and x < 100:
        z = x - 2
        c = 0
        while z >= 2:
            if z ** (x - 1) % x == 1:
                c = c + 1
                z = z - 1
        if c == x - 3:
            return True
    return False
MONKEY TACOS!
I'M SO RANDOM.

YEAH, ME TOO.

Original: https://xkcd.com/1210 CC-BY-NC 2.5
Fuzz Testing

*Goal:* To find **program inputs** that reveal a **bug**

*Approach:* Generate inputs **randomly** until program **crashes**
Fuzz Testing

“On a dark and stormy night one of the authors was logged on to his workstation on a dial-up line from home and the rain had affected the phone lines; there were frequent spurious characters on the line. The author had to race to see if he could type a sensible sequence of characters before the noise scrambled the command. This line noise was not surprising; but we were surprised that these spurious characters were causing programs to crash.”

Communications of the ACM (1990)
1990s

Fuzz Testing 101

1990 study found crashes in: adb, as, bc, cb, col, diction, emacs, eqn, ftp, indent, lex, look, m4, make, nroff, plot, prolog, ptx, refer!, spell, style, tsort, uniq, vgrind, vi

/dev/random → w0019[a%#] → Execute → Program
Why do programs crash?
Common Fuzzer-Found Bugs

Causes: incorrect arg validation, incorrect type casting, executing untrusted code, etc.

Effects: buffer-overflows, memory leak, division-by-zero, use-after-free, assertion violation, etc. (“crash”)

Impact: security, reliability, performance, correctness
What are the **benefits**, **challenges**, & **limitations** of this approach?
Generate inputs randomly

$ ant -f build.xml

$ ant -f /dev/random

Purely random data is not a very interesting input!!
Generate inputs randomly via mutation

$ ant -f build.xml

$ ant -f build.xml.mut
What are some good mutations?
Mutation Heuristics

- **Binary input**
  - Bit flips, byte flips
  - Change random bytes
  - Insert random byte chunks
  - Delete random byte chunks
  - Set randomly chosen byte chunks to *interesting* values e.g. INT_MAX, INT_MIN, 0, 1, -1, ...
  - Other suggestions?

- **Text input**
  - Insert random symbols or keywords from a dictionary
  - Other suggestions?
Mutation-Based Fuzzing (e.g. Radamsa, zzuf)

2000s

Input

Seeds

Initial

Pick

<foo></foo>

Random Mutation

<woo>?</oo>

Execute

Program

2000s
2000s

**Mutation-Based Fuzzing (e.g. Radamsa, zzuf)**

![Diagram showing the process of mutation-based fuzzing. Initial seeds lead to random mutations, which are then executed to find bugs.]

- **Valid Seed Input (build.xml)**
  ```xml
  <project default="dist">
    <target name="init">
      <mkdir dir="${build}"/>
    </target>
    ...
  </project>
  ```

- **New Input (Mutated from Seed)**
  ```xml
  <project default="dist">
    <wget name="init">
      <mkdir dir="2{build}"/>
    </wget>
    ...
  </project>
  ```
What are the benefits, challenges, & limitations of this approach?
How do you know if you are making progress?
## Code Coverage

### LCOV - code coverage report

<table>
<thead>
<tr>
<th>Function</th>
<th>Line Coverage</th>
<th>Functions Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>main.c</td>
<td>74.0%</td>
<td>74.0%</td>
</tr>
<tr>
<td>test.c</td>
<td>97.6%</td>
<td>97.6%</td>
</tr>
<tr>
<td>libtest.c</td>
<td>78.0%</td>
<td>78.0%</td>
</tr>
<tr>
<td>testlib.c</td>
<td>97.7%</td>
<td>97.7%</td>
</tr>
<tr>
<td>crypto.c</td>
<td>83.3%</td>
<td>83.3%</td>
</tr>
<tr>
<td>cmmdabin.c</td>
<td>60.0%</td>
<td>60.0%</td>
</tr>
<tr>
<td>plugin.c</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>smtp.c</td>
<td>95.5%</td>
<td>95.5%</td>
</tr>
<tr>
<td>smtpd.c</td>
<td>72.9%</td>
<td>72.9%</td>
</tr>
<tr>
<td>smtpd.c</td>
<td>75.5%</td>
<td>75.5%</td>
</tr>
<tr>
<td>smtpd.c</td>
<td>84.6%</td>
<td>84.6%</td>
</tr>
<tr>
<td>smtpd.c</td>
<td>84.8%</td>
<td>84.8%</td>
</tr>
<tr>
<td>smtpd.c</td>
<td>97.1%</td>
<td>97.1%</td>
</tr>
<tr>
<td>smtpd.c</td>
<td>94.6%</td>
<td>94.6%</td>
</tr>
<tr>
<td>smtpd.c</td>
<td>94.0%</td>
<td>94.0%</td>
</tr>
<tr>
<td>smtpd.c</td>
<td>92.8%</td>
<td>92.8%</td>
</tr>
<tr>
<td>smtpd.c</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>smtpd.c</td>
<td>99.3%</td>
<td>99.3%</td>
</tr>
<tr>
<td>smtpd.c</td>
<td>84.7%</td>
<td>84.7%</td>
</tr>
<tr>
<td>smtpd.c</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>smtpd.c</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>smtpd.c</td>
<td>76.8%</td>
<td>76.8%</td>
</tr>
<tr>
<td>smtpd.c</td>
<td>92.8%</td>
<td>92.8%</td>
</tr>
<tr>
<td>smtpd.c</td>
<td>95.5%</td>
<td>95.5%</td>
</tr>
<tr>
<td>smtpd.c</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

### Code Snippet

```c
0 / */
1 / */
2 / */
3 / */
4 / */
5 / */
6 / */
7 / */
8 / */
9 / */
10 / */
11 / */
12 / */
13 / */
14 / */
15 / */
16 / */
17 / */
18 / */
19 / */
20 / */
21 / */
22 / */
23 / */
24 / */
25 / */
```
Exercise: How do you collect coverage?

```java
if (x && y) {
    s1;
    s2;
} else {
    while(b) {
        s3;
    }
}
```
Coverage-Guided Fuzzing with AFL

Initial → Seeds → Input → Pick → Input → Random Mutation → Input’ → Execute → Program

Save?:
- Yes → Execution feedback → coverage
- No → New branch coverage?
  - Yes → Add Input’ → Input
  - No → Exit

2010s

Coverage - Guided Fuzzing with AFL
Coverage-Guided Fuzzing with AFL

November 07, 2014

Pulling JPEGs out of thin air

This is an interesting demonstration of the capabilities of afl; I was actually pretty surprised that it worked!

```
$ mkdir in_dir
$ echo 'hello' > in_dir/hello
$ ./afl-fuzz -i in_dir -o out_dir ./jpeg-9a/djpeg
```
# Coverage-Guided Fuzzing with AFL

## The bug-o-rama trophy case

<table>
<thead>
<tr>
<th>LIG jpeg</th>
<th>libjpeg-turbo</th>
<th>libpng</th>
</tr>
</thead>
<tbody>
<tr>
<td>libtiff</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>mozjpeg</td>
<td></td>
</tr>
<tr>
<td>Mozilla Firefox</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adobe Flash / PCRE</td>
<td>sqlite</td>
<td>OpenSSL</td>
</tr>
<tr>
<td>LibreOffice</td>
<td>poppler</td>
<td>freetype</td>
</tr>
<tr>
<td>GnuTLS</td>
<td>GnuPG</td>
<td>OpenSSH</td>
</tr>
<tr>
<td>PuTTY</td>
<td>ntpd</td>
<td>nginx</td>
</tr>
<tr>
<td>bash (post-Shellshock)</td>
<td>tcpdump</td>
<td>JavaScriptCore</td>
</tr>
<tr>
<td>pdfium</td>
<td>ffmpeg</td>
<td>libmatroska</td>
</tr>
<tr>
<td>libarchive</td>
<td>wireshark</td>
<td>ImageMagick</td>
</tr>
<tr>
<td>BIND</td>
<td>QEMU</td>
<td>icms</td>
</tr>
</tbody>
</table>
ClusterFuzz @ Chromium

<table>
<thead>
<tr>
<th>ID</th>
<th>Pri</th>
<th>M</th>
<th>Stars</th>
<th>ReleaseBlock</th>
<th>Component</th>
<th>Status</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1133812</td>
<td>1</td>
<td>2</td>
<td>-----</td>
<td>-----</td>
<td>Blink&gt;GetUserMedia&gt;Webcam</td>
<td>Untriaged</td>
<td></td>
</tr>
<tr>
<td>1133763</td>
<td>1</td>
<td>1</td>
<td>-----</td>
<td>-----</td>
<td></td>
<td>Untriaged</td>
<td></td>
</tr>
<tr>
<td>1133701</td>
<td>1</td>
<td>1</td>
<td>-----</td>
<td>-----</td>
<td>Blink&gt;JavaScript</td>
<td>Untriaged</td>
<td></td>
</tr>
<tr>
<td>1133254</td>
<td>1</td>
<td>2</td>
<td>-----</td>
<td>-----</td>
<td></td>
<td>Untriaged</td>
<td></td>
</tr>
<tr>
<td>1133124</td>
<td>1</td>
<td>1</td>
<td>-----</td>
<td>-----</td>
<td></td>
<td>Untriaged</td>
<td></td>
</tr>
<tr>
<td>1133024</td>
<td>2</td>
<td>3</td>
<td>-----</td>
<td>-----</td>
<td>Internals&gt;Network</td>
<td>Started</td>
<td>dmcardle@ct</td>
</tr>
<tr>
<td>1132958</td>
<td>1</td>
<td>2</td>
<td>-----</td>
<td></td>
<td>Uts&gt;Accessibility,</td>
<td>Assigned</td>
<td>sin...@chrom</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Blink&gt;Accessibility</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1152907</td>
<td>2</td>
<td>2</td>
<td>-----</td>
<td></td>
<td>Blink&gt;JavaScript&gt;GC</td>
<td>Assigned</td>
<td>dirfuehr@chr</td>
</tr>
</tbody>
</table>
Libarchive#1165 (CVE-2019-11463)

Fix typo in preprocessor macro in archive_read_format_zip_cleanup()
Frees lzma_stream on cleanup()
Fixes #1165

Easy to fix, hard to find!!
Challenging Problems

• Fuzzing heuristics
  o Mutation: Which input to mutate? How many times? Which mutations?
  o Feedback: What to instrument? How to keep overhead low?

• Oracles
  o What is a bug? Crash? Silent overflow? Infinite loop? Race condition? Undefined behavior? How do we know when we have found a bug?

• Debugging
  o Reproducibility
  o Crash triaging
  o Input minimization

• Fuzzing roadblocks
  o Magic bytes, checksums (see PNG, SSL)
  o Dependencies in binary inputs (e.g. length of chunks, indexes into tables – see PNG)
  o Inputs with complex syntax and semantics (e.g. XML, JSON, C++)
  o Stateful applications
Oracles: Sanitizers

- Address Sanitizer (ASAN)
- LeakSanitizer (comes with ASAN)
- Thread Sanitizer (TSAN)
- Undefined-behavior Sanitizer (UBSAN)

https://github.com/google/sanitizers
int get_element(int* a, int i) {
    if (a == NULL) abort();
    return a[i];
}

int get_element(int* a, int i) {
    if (a == NULL) abort();
    region = get_allocation(a);
    if (in_heap(region)) {
        low, high = get_bounds(region);
        if ((a + i) < low || (a + i) > high) {
            abort();
        }
    }
    return a[i];
}

int get_element(int* a, int i) {
    if (a == NULL) abort();
    region = get_allocation(a);
    if (in_stack(region)) {
        if (popped(region)) abort();
        ...
    }
    if (in_heap(region)) { ...
    return a[i];
}
AddressSanitizer

https://github.com/google/sanitizers/wiki/AddressSanitizer

Asan is a memory error detector for C/C++. It finds:

- Use after free (dangling pointer dereference)
- Heap buffer overflow
- Stack buffer overflow
- Global buffer overflow
- Use after return
- Use after scope
- Initialization order bugs
- Memory leaks

Compile with `clang --fsanitize=address`

Slowdown on SPEC CPU 2006
## Crash Triaging

### Process Timing
- run time: 0 days, 0 hrs, 5 min, 20 sec
- last new path: 0 days, 0 hrs, 9 sec
- last uniq crash: 0 days, 0 hrs, 49 sec
- last uniq hang: 0 days, 0 hrs, 19 sec

### Cycle Progress
- now processing: 121 (50.21%)
- paths timed out: 0 (0.00%)

### Stage Progress
- now trying: interest 32/8
- stage execs: 3550/8883 (39.96%)
- total execs: 7777
- exec speed: 3560/sec

### Fuzzing Strategy Yields
- bit flips: 91/30.7k, 15/30.7k, 6/30.6k
- byte flips: 1/3838, 1/3542, 2/3510
- arithmetics: 42/198k, 3/71.9k, 0/32.0k
- known ints: 3/19.1k, 7/84.4k, 22/132k
- dictionary: 0/0, 0/0, 5/23.3k
- havoc: 55/106k, 0/0
- trim: 22.95%/1711, 7.22%

### Overall Results
- cycles done: 0
- total paths: 241
- uniq crashes: 14
- uniq hangs: 22

### Findings in Depth
- favored paths: 51 (21.16%)
- new edges on: 75 (31.12%)
- total crashes: 140 (14 unique)
- total hangs: 400 (22 unique)

### Path Geometry
- levels: 3
- pending: 217
- pend fav: 38
- own finds: 239
- imported: n/a
- stability: 100.00%
Crash Triaging

- Given two crashing inputs $x_1$ and $x_2$, do they trigger the same bug?
- Very difficult to answer in practice
- Herustics: $\text{bug}(x_1) = \text{bug}(x_2)$ only if...
  - $\text{exitcode}(x_1) = \text{exitcode}(x_2)$ // or exception or error msg
  - $\text{coverage}(x_1) = \text{coverage}(x_2)$
  - $\text{stacktrace}(x_1) = \text{stacktrace}(x_2)$
  - $\text{newcoverage}(x_1, \text{old}) = \text{newcoverage}(x_2, \text{old})$ // AFL
  - $\text{fix}(x_1) = \text{fix}(x_2)$
CAN WE GO BEYOND COVERAGE AND CRASHES?

(recent research results)
Domain-Specific Fuzzing

Zest [Padhye et al. 2018]
exercise semantic analysis and transformation pipelines
save if “increases coverage amongst valid inputs”

PerfFuzz [Lemieux et al. 2018]
discover worst-case performance
save if “maximizes branch exec counts”

MemFuzz [Coppik et al. 2019]
explore state machines
save if “accesses new input-dependent memory locations”

DifFuzz [Nilizadeh et al. 2019]
find side-channel attacks
save if “side channel info differs from reference execution”

Seed input \[\rightarrow\] Waypoints \[\leftarrow\] Target input

single-mutation search space
Given a new mutated input $i \in I$, should $i$ be saved to set of seeds $S$?

**Our goal:** Allow users to define *is_waypoint*
is_waypoint(i, S) = false

is_waypoint(i, S) = true
is_waypoint\((i, S) = \text{ "Closer to my goal"}\)
Q1: How to define $is\_waypoint(i, S)$ easily?

Q2: How to define $is\_waypoint(i, S)$ to ensure progress?
**FuzzFactory**: Domain-Specific Fuzzing

- **Seeds** → **Initial** → **Input** → **Pick** → **Input** → **Mutate** → **Input’** → **Execute** → **Program**

- **Save?**
  - Yes → **Execution feedback**
  - No → **is_waypoint?**

- **Domain-Specific Feedback**

- **Domain-Specific Instrumentation**

- **Teacher:**
  - **dsf** :: **K** → **V**
  - **is_waypoint?**

- **New aggregate value in A for some k?**

---

Q1: How to define $\text{is\_waypoint}(i, S)$ easily?

\[ dsf_i \:: \ K \rightarrow V \]

E.g. $K = \text{Program Locations}$, $V = \text{Exec Counts}$

E.g. PerfFuzz

<table>
<thead>
<tr>
<th>$k$</th>
<th>$dsf_{i1}$</th>
<th>$dsf_{i2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loc$_1$</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Loc$_2$</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Q1: How to define `is_waypoint(i, S)` easily?

\[ dsf_i :: K \rightarrow V \]
\[ \text{e.g. } K = \text{Program Locations}, \ V = \text{Exec Counts} \]

\[ \triangleright :: A \times V \rightarrow A \text{ with } a_0 \in A \]
\[ \text{e.g. } \triangleright = \text{max} \]
\[ A = \mathbb{N}, \ a_0 = 0 \]

<table>
<thead>
<tr>
<th>k</th>
<th>dsf_{i1}</th>
<th>dsf_{i2}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loc_1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Loc_2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Q1: How to define `is_waypoint(i, S)` easily?

\[ dsf_i : K \rightarrow V \]

\[ \text{e.g. } K = \text{Program Locations, } V = \text{Exec Counts} \]

\[ \triangleright : A \times V \rightarrow A \text{ with } a_o \in A \]

\[ \text{e.g. } \triangleright = \text{max} \]

\[ A = \mathbb{N}, a_o = 0 \]

---

Q1: How to define \textit{is\_waypoint}(i, S) easily?

\[ dsf_i :: K \rightarrow V \]
\[ \text{e.g. } K = \text{Program Locations}, \ V = \text{Exec Counts} \]

\[ \triangleright :: A \times V \rightarrow A \text{ with } a_o \in A \]
\[ \text{e.g. } \triangleright = \text{max} \]
\[ A = \mathbb{N}, \ a_o = 0 \]

\begin{tabular}{|c|c|c|c|c|}
\hline
k & $\triangleright$ & dsf_{i1} & $\triangleright$ & dsf_{i2} \\
\hline
Loc_1 & 0 & 4 & 4 & 5 \\
\hline
Loc_2 & 0 & 2 & 2 & 1 \\
\hline
\end{tabular}

Q1: How to define \( \text{is\_waypoint}(i, S) \) easily?

\[ dsf_i :: K \rightarrow V \]
\[ \text{e.g. } K = \text{Program Locations}, \ V = \text{Exec Counts} \]

\[ \triangleright :: A \times V \rightarrow A \text{ with } a_0 \in A \]
\[ \text{e.g. } \triangleright = \text{max} \]
\[ A = \mathbb{N}, \ a_o = 0 \]

---

Q1: How to define \texttt{is}_\texttt{waypoint}(i, S) easily?

\[
\text{dsf}_i :: K \rightarrow V
\]
e.g. \( K = \text{Program Locations}, \ V = \text{Exec Counts} \)

\[
\triangleright :: A \times V \rightarrow A \text{ with } a_o \in A
\]
e.g. \( \triangleright = \max \)
\( A = \mathbb{N}, \ a_o = 0 \)

\[
is\_\text{waypoint}(i, S) \overset{\text{def}}{=} \exists k : s \in S \ 	ext{dsf}_s(k) \neq s \in (S \cup \text{dsf}_s(k))
\]

\begin{table}[h]
\begin{tabular}{|c|c|c|c|c|c|}
\hline
k & $\triangleright$ & dsf_{i1} & $\triangleright$ & dsf_{i2} & $\triangleright$ \\
\hline
Loc_1 & 0 & 4 & 4 & 5 & 5 \\
\hline
Loc_2 & 0 & 2 & 2 & 1 & 2 \\
\hline
\end{tabular}
\end{table}

Q1: How to define \texttt{is\_waypoint}(i, S) easily?

\[ dsf_i :: K \rightarrow V \]

\[ \text{e.g. } K = \text{Program Locations}, \ V = \text{Exec Counts} \]

\[ \triangleright :: A \times V \rightarrow A \text{ with } a_o \in A \]

\[ \text{e.g. } \triangleright = \text{max} \]

\[ A = \mathbb{N}, \ a_o = 0 \]

\[ \text{is\_waypoint}(i, S) \overset{\text{def}}{=} \exists k : \bigtriangleup_s \in S \ dsf_s(k) \neq \bigtriangleright_s = (S \cup \ dsf_s(k)) \]

Q1: How to define \texttt{is\_waypoint}(i, S) easily?

\[ \text{dsf}_i :: K \rightarrow V \]

e.g. \( K = \text{Program Locations}, \ V = \text{Exec Counts} \)

\[ \triangleright :: A \times V \rightarrow A \text{ with } a_0 \in A \]

\[ a \triangleright v = a \cup \{ \text{floor} (\log_2 v) \} \]

\[ A = 2^\mathbb{N}, \ a_0 = \emptyset \]

\[ \text{is\_waypoint}(i, S) \overset{\text{def}}{=} \exists k : \bigtriangleup_{s \in S} \text{dsf}_S (k) \neq s \triangleright (s \cup \text{dsf}_S (k)) \]

\begin{tabular}{|c|c|c|c|c|}
\hline
\( k \) & \( \triangleright \) & \text{dsf}_{i1} & \( \triangleright \) & \text{dsf}_{i2} & \( \triangleright \) \\
\hline
Loc_1 & \( \emptyset \) & 4 & \{4\} & 5 & \{4\} \\
\hline
Loc_2 & \( \emptyset \) & 2 & \{2\} & 1 & \{1,2\} \\
\hline
\end{tabular}

Q1: How to define `is_waypoint(i, S)` easily?

\[
\text{dsf}_i :: K \rightarrow V
\]
e.g. \( K = \text{Program Locations}, \ V = \text{Exec Counts} \)

\[
\triangleright :: A \times V \rightarrow A \ \text{with} \ a_0 \in A
\]
a \triangleright v = a \cup \{\ \text{floor}(\log_2 v)\ \}
\( A = 2^\mathbb{N}, \ a_0 = \emptyset \)

\[
is\_\text{waypoint}(i, S) \ \overset{\text{def}}{=} \exists k . \ \text{dsf}_s(k) \neq \bigtriangledown_{s \in (S \cup \{i\})} \ \text{dsf}_s(k)
\]

Q2: How to ensure progress?

\[
a \triangleright v \triangleright v = a \triangleright v \quad \text{(Idempotence)}
a \triangleright u \triangleright v = a \triangleright v \triangleright u \quad \text{(Order Insensitivity)}
\]

Sufficient to show that: \( \triangleright \) is monotonic

\[
\therefore \ is\_\text{waypoint} \iff \text{progress}
\]

Q2: How to ensure progress? Defining the $\triangleright$ function

$\forall a \in A \forall v \in V : a \triangleright v \triangleright v = a \triangleright v$ (Idempotence)

$\forall a \in A \forall v \in V : a \triangleright u \triangleright v = a \triangleright v \triangleright u$ (Order Insensitivity)

**Good:**

- $a \triangleright v = \max(a, v)$
- $a \triangleright v = \min(a, v)$
- $a \triangleright v = a \lor b$
- $a \triangleright v = a \cup \{ \lfloor \log_2 v \rfloor \}$

**Bad:**

- $a \triangleright v = a + v$
- $a \triangleright v = a \oplus v$
- $a \triangleright v = v$

Open Problems – Research Opportunities!

• What if fuzzing doesn’t find any bugs after X hours?
  o Is the program bug free?
    ▪ **RQ:** What is the probability that there are more bugs lurking around?
  o Should we keep fuzzing?
    ▪ **RQ:** When should we stop to balance cost vs. results?
  o Can we change the feedback function? Mutation?
    ▪ **RQ:** What changes can we make? How can we bring a human in the loop?

• How to balance instrumentation overhead with feedback quality?
  o **RQ:** What parts of the code should be instrumented?

• How to generate *meaningful* test cases?
  o **RQ:** What is “meaningful”?
  o **RQ:** How to generate good inputs by construction?