



University of
Sheffield



COM3529 Software Testing and Analysis

Fuzzing

Dr José Miguel Rojas

BB-YB-BD

Roadmap

Beizer's Maturity Model

Test Automation

Unit Testing

Control/Data-Flow Analysis

Code Coverage

Mutation Testing

Regression Testing

Fuzzing

Search-based Test Generation

Model-Based Testing

Roadmap

Beizer's Maturity Model

Test Automation

Unit Testing

Control/Data-Flow Analysis

Code Coverage

Mutation Testing

Regression Testing

Fuzzing

Search-based Test Generation

Model-Based Testing

Motivation

One dark, stormy night in 1990, Bart Miller (U Wisc.) was logged to work in over dial-up

There was a lot of line noise due to the storm

His shell and editors kept crashing

This gave him an idea...



Motivation

Idea: Feed “fuzz” – streams of pure randomness to OS & utility code, e.g., noise from `/dev/random`

Watch them break!

This would crash 25-33% of utilities

Reports every few years since then

Some of the bugs are the same ones in common security exploits (specially buffer overruns)



Fuzzing

Software developers make mistakes

Mistakes -> bugs -> security vulnerabilities

Fuzzing aims to send malformed/random input to a program and hope to crash it or find a security hole

Useful for finding bugs to protect programs – “white hat” work

But also for finding bugs to hack into systems – “black hat”!

Used to find (security) problems in many well-known companies / products, e.g., Apple, Firefox

User Testing vs Fuzzing

Run program on multiple **normal** inputs, looking for bad things to happen

Goal: Prevent **normal users** from encountering errors

Run program on multiple **abnormal** inputs, looking for bad things to happen

Goal: Prevent **attackers** from encountering **exploitable** errors

Excel File Edit View Insert Format Tools Data Window Help

Sort... ⌘R

Filter ⌘F

Save the changes you made to [book21]

Clear Filters

Advanced Filter...

Form...

Subtotals...

Validation...

Data Table...

Text to Columns...

Consolidate...

Group and Outline ►

PivotTable...

Table Tools ►

Get External Data ► (highlighted)

Refresh Data

Run Saved Query...

New Database Query...

Import Text File...

Import from FileMaker Pro...

Import from FileMaker Server...

Edit Query...

Data Range Properties

Parameters...

Insert Chart

Column Line Pie Bar Area Scatter Other Line

A12

1 lasdjflfajsd

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

Sheet1 +

Ready

Sum = 0

(47%) Sun 12:33 AM guitest Q

2012-08-26 00:33:47 366 BD YB-BD

Excel File Edit View Insert Format Tools Data Window Help

Sort... ⌘R

Filter ⌘F

Save the changes you made to [book21]

Clear Filters

Advanced Filter...

Form...

Subtotals...

Validation...

Data Table...

Text to Columns...

Consolidate...

Group and Outline ►

PivotTable...

Table Tools ►

Get External Data ► (highlighted)

Refresh Data

Run Saved Query...

New Database Query...

Import Text File...

Import from FileMaker Pro...

Import from FileMaker Server...

Edit Query...

Data Range Properties

Parameters...

Insert Chart

Column Line Pie Bar Area Scatter Other Line

A12

1 lasdjflfajsd

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

Sheet1

Ready

Sum = 0

(47%) Sun 12:33 AM guitest Q

2012-08-26 00:33:47 366 BD YB-BD

Fuzzing

Automatically generate **random** (malformed) test cases

Monitor application for errors

Nature of inputs can be diverse

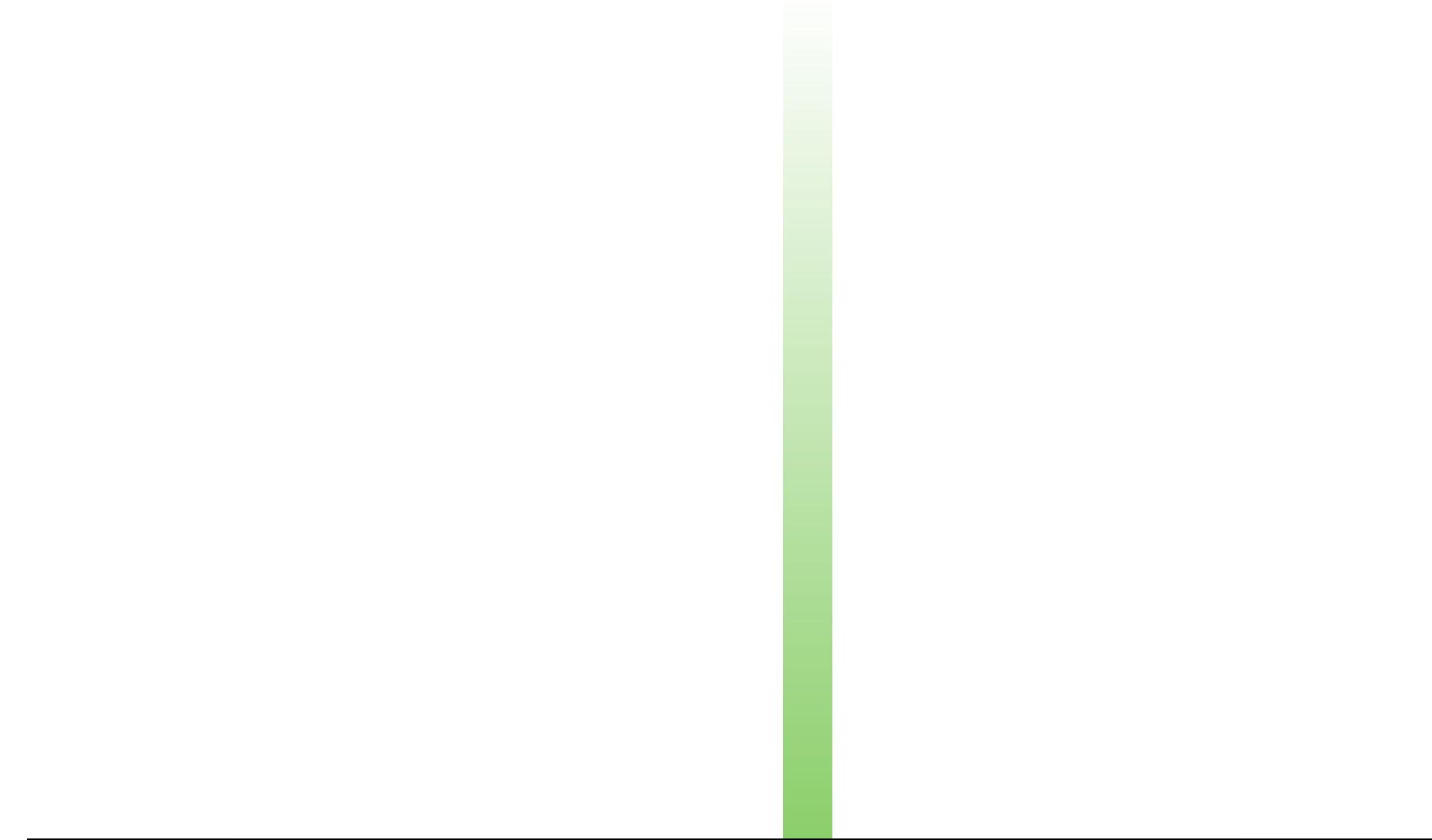
Files: .pdf, .png, .wav, .mpg, .json, etc

Network based: http, SOAP, SNMP

Likelihood of Finding Bugs?

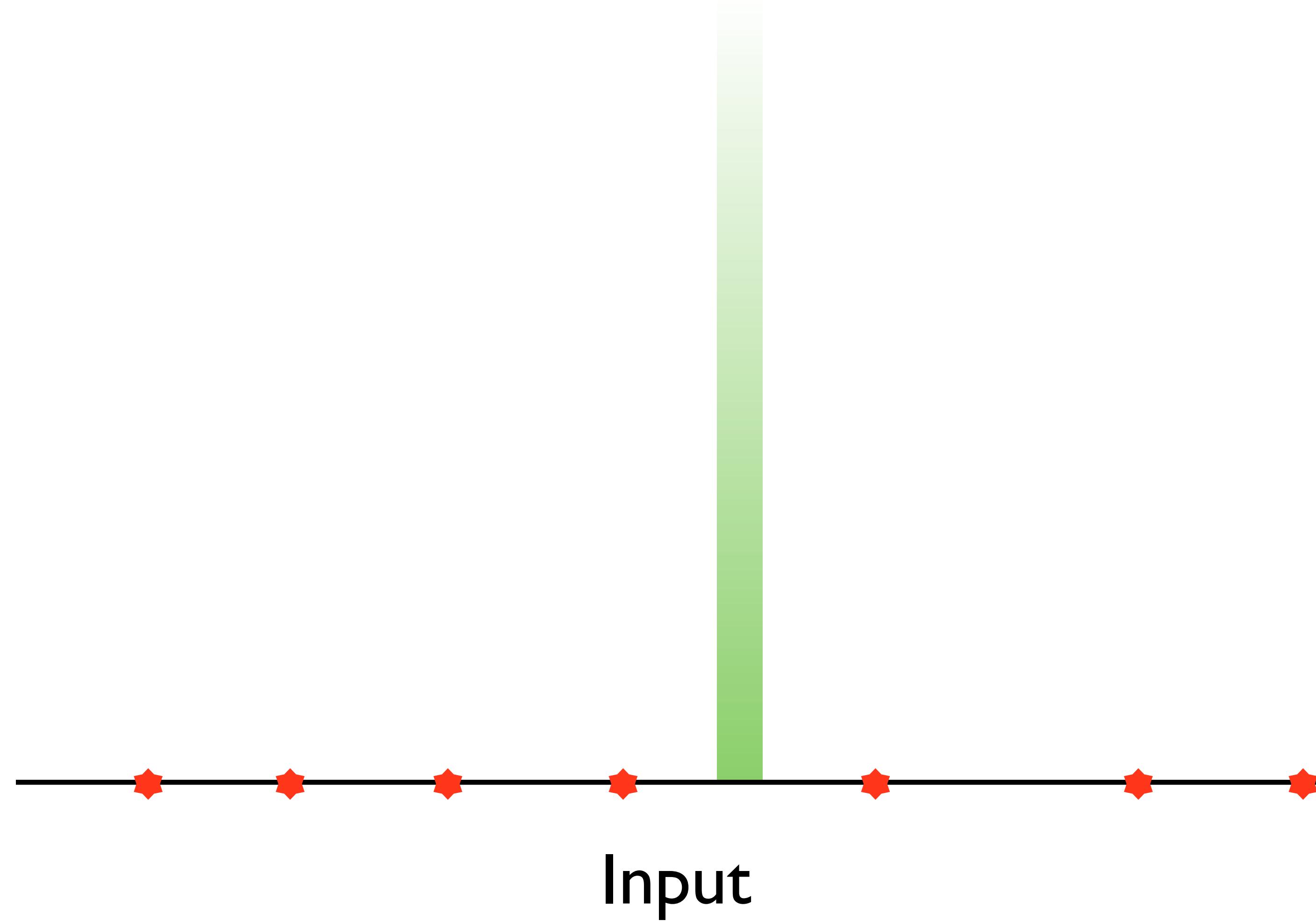
Input

Likelihood of Finding Bugs?



Input

Likelihood of Finding Bugs?



How to Fuzz?

Very long inputs, very short inputs, blank input, ...

Min/max values of integers, zero and negative values

Special values, characters or keywords likely to trigger bugs, e.g., nulls, newlines, or end-of-file characters

Format string characters %s %x %n

Semi-colons, slashes and backslashes, quotes, non-printable characters, ...

Application-specific keywords, e.g., halt, return, DROP TABLES, ...

Template-based Fuzzing

Also known as **Mutation-based**

Programs requiring **complex input** cannot properly be tested by random input

Example: JavaScript interpreter needs valid JavaScript programs as test inputs

Instead of random inputs, **modify (mutate)** existing, valid inputs

Mutations may be random or guided by heuristics

Minimal set up time, but technique is dependant on target programs

Coverage-based Fuzzing

If the fuzzer happens to create a valid, useful input, e.g., covering a new branch, use that as a starting point for further mutations

Also known as: **white-box fuzzing**

Grammar-based Testing

How to generate complex **textual** inputs?

Classical application: Testing compilers

Nowadays: Web applications, interpreters, parsers, ...anything that uses XML, JSON, etc.

Microsoft's **SAGE** tool (Godefroid et al, 2008a) is reported to have saved the company millions of dollars' worth of software bugs.

Context-free Grammars

Finite set of *terminals*

Finite set of *non-terminals*

Finite set of *rules*

Each *rule*: *Non-terminal* \rightarrow List of *terminals* and *non-terminals*

Starting rule

Example Grammar

```
expr → expr op expr  
expr → ( expr )  
expr → - expr  
expr → id  
op → +  
op → -  
op → *  
op → /  
op → ↑
```

Derivation

Interpret production as rewriting rule

Non-terminal on the left hand side is replaced by string on the right hand side

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(id)$$

Sequence of replacements = derivation

Two choices at each derivation step:

- Which non-terminal to replace
- Which alternative to use for the replaced non-terminal

Grammar-based Testing

Test generation → rewriting based on grammar

Begin with start symbol

Replace one non-terminal on the right with a rule with the non-terminal on the left

Repeat until only terminals are left

Result is a test input

expr → expr op expr
expr → (expr)
expr → - expr
expr → id
op → +
op → -
op → *
op → /
op → ↑

E ⇒ - E ⇒ - (E) ⇒ - (E + E) ⇒ - (E + id) ⇒ - (id + id)

Problems

Context-free grammars → recursion

Recursion → **infinite** number of possible inputs

Which rule to replace with next?

Grammar Annotation

Limit recursion depth

Limit number of occurrences

Limit parse tree depth

```
{count 3} zeros;  
zeros ::= '0';  
zeros ::= '0' zeros;
```

Covering Grammars

Production coverage

Each production rule must be used to generate at least one (section of) test case

Boundary condition

Annotate each recursive production rule with minimum and maximum number of application, then generate:

Minimum

Minimum + 1

Maximum - 1

Maximum

A Combinatorial Problem

E.g., testing VoIP software

Caller, VoIP server, client

CallerOS: Windows, Mac

ServerOS: Linux, Sun, Windows

CalleeOS: Windows, Mac

Same Problem as Grammar

It is possible to use covering arrays to test with grammars

```
Call ::= CallerOS ServerOS CalleeOS;  
CallerOS ::= 'Mac';  
CallerOS ::= 'Win';  
ServerOS ::= 'Lin';  
ServerOS ::= 'Sun';  
ServerOS ::= 'Win';  
CalleeOS ::= 'Mac';  
CalleeOS ::= 'Win';
```

Tools

LibFuzzer (<https://llvm.org/docs/LibFuzzer.html>),

AFL (<https://github.com/google/AFL>)

AFL++ (<https://github.com/AFLplusplus>)

Spike (<https://www.kali.org/tools/spike/>)

Peach (<https://wiki.mozilla.org/Security/Fuzzing/Peach>)

OneFuzz (<https://github.com/microsoft/onefuzz>)

Honggfuzz (<https://github.com/google/honggfuzz>)

AFL (American Fuzzy Lop)++

Feedback-based fuzzer

Mutate file – Does it add coverage? Keep it; else delete

```
american fuzzy lop 0.47b (readpng)
process timing
  run time : 0 days, 0 hrs, 4 min, 43 sec
  last new path : 0 days, 0 hrs, 0 min, 26 sec
  last uniq crash : none seen yet
  last uniq hang : 0 days, 0 hrs, 1 min, 51 sec
cycle progress
  now processing : 38 (19.49%)
  paths timed out : 0 (0.00%)
stage progress
  now trying : interest 32/8
  stage execs : 0/9990 (0.00%)
  total execs : 654k
  exec speed : 2306/sec
fuzzing strategy yields
  bit flips : 88/14.4k, 6/14.4k, 6/14.4k
  byte flips : 0/1804, 0/1786, 1/1750
  arithmetics : 31/126k, 3/45.6k, 1/17.8k
  known ints : 1/15.8k, 4/65.8k, 6/78.2k
  havoc : 34/254k, 0/0
  trim : 2876 B/931 (61.45% gain)
overall results
  cycles done : 0
  total paths : 195
  uniq crashes : 0
  uniq hangs : 1
map coverage
  map density : 1217 (7.43%)
  count coverage : 2.55 bits/tuple
findings in depth
  favored paths : 128 (65.64%)
  new edges on : 85 (43.59%)
  total crashes : 0 (0 unique)
  total hangs : 1 (1 unique)
path geometry
  levels : 3
  pending : 178
  pend fav : 114
  imported : 0
  variable : 0
  latent : 0
```

```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
process timing
  run time : 0 days, 0 hrs, 0 min, 43 sec
  last new path : 0 days, 0 hrs, 0 min, 1 sec
  last uniq crash : none seen yet
  last uniq hang : none seen yet
cycle progress
  now processing : 261*1 (37.1%)
  paths timed out : 0 (0.00%)
stage progress
  now trying : splice 14
  stage execs : 31/32 (96.88%)
  total execs : 2.55M
  exec speed : 61.2k/sec
fuzzing strategy yields
  bit flips : n/a, n/a, n/a
  byte flips : n/a, n/a, n/a
  arithmetics : n/a, n/a, n/a
  known ints : n/a, n/a, n/a
  dictionary : n/a, n/a, n/a
  havoc/splice : 506/1.05M, 193/1.44M
  py/custom : 0/0, 0/0
  trim : 19.25%/53.2k, n/a
overall results
  cycles done : 15
  total paths : 703
  uniq crashes : 0
  uniq hangs : 0
map coverage
  map density : 5.78% / 13.98%
  count coverage : 3.30 bits/tuple
findings in depth
  favored paths : 114 (16.22%)
  new edges on : 167 (23.76%)
  total crashes : 0 (0 unique)
  total timeouts : 0 (0 unique)
path geometry
  levels : 11
  pending : 121
  pend fav : 0
  own finds : 699
  imported : n/a
  stability : 99.88%
[cpu000: 12%]
```

Summary

Simple testing techniques such as **fuzzing** can be very effective

Relatively easy to implement

Can be costly – when to stop? – and hard to assess w.r.t. coverage

Several tools available, e.g.,

Excellent resource: The Fuzzing Book (<https://www.fuzzingbook.org/>)