Lec 23: Fuzzing

IS561: Binary Code Analysis and Secure Software Systems

Sang Kil Cha

Fuzzing

What is Fuzzing?

Fuzzing is a software testing technique for finding ***software bugs***.

Why Fuzzing?

Simple[1] and fast.

[1] Simpler than symbolic exeuction.

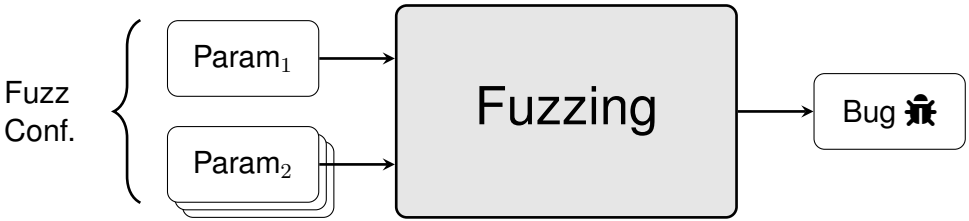
History of Fuzzing

The original work was inspired by being logged on to a modem during a storm with lots of line noise. And the line noise was generating junk characters that seemingly were causing programs to crash. The noise suggested the term ***fuzz***.

The term was coined by Barton Miller in 1990.

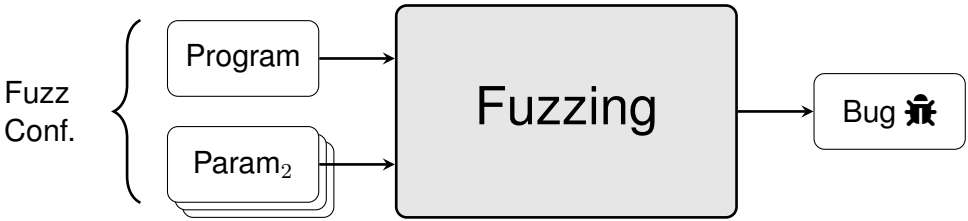
Fuzzing in 1990s

An Empirical Study of the Reliability of UNIX Utilities, ***CACM 1990***.



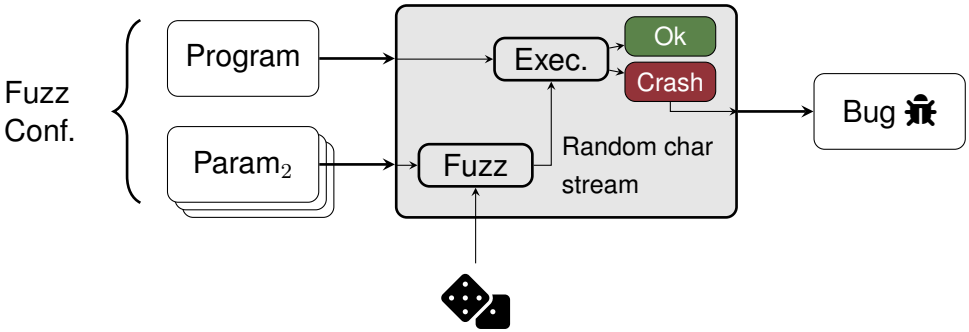
Fuzzing in 1990s

An Empirical Study of the Reliability of UNIX Utilities, ***CACM 1990***.



Fuzzing in 1990s

An Empirical Study of the Reliability of UNIX Utilities, ***CACM 1990***.



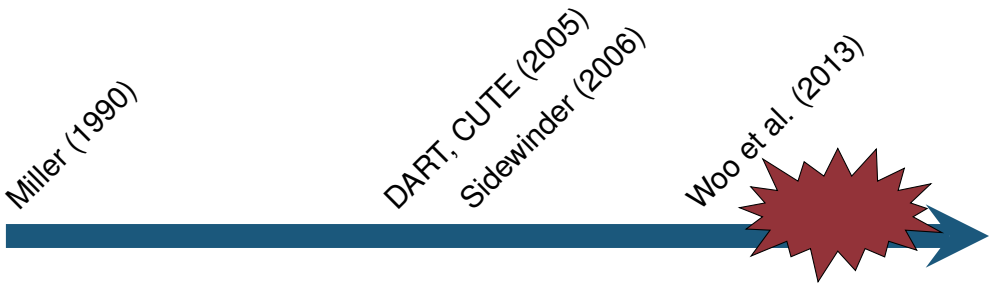
Fuzzing is ...

- Simple, and popular way to find security bugs.
- Used by security practitioners.
- But, ***not studied systematically until early 2010***.

Fuzzing is ...

- Simple, and popular way to find security bugs.
- Used by security practitioners.
- But, ***not studied systematically until early 2010***.
- Natural research questions:
  - Why fuzzing works so well in practice?
  - Are we maximizing the ability of fuzzing?

Rough History of Fuzzing[2]

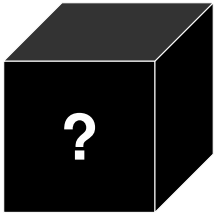


[2]Visit `https://fuzzing-survey.org` to learn more.

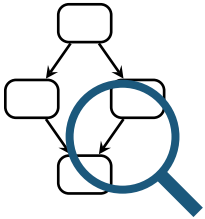
Fuzzing is an Overloaded Term

- White-box, black-box, and grey-box fuzzing.
- Directed fuzzing and undirected fuzzing.
- Feedback-driven fuzzing.
- Generational fuzzing and mutational fuzzing.
- Grammar-based fuzzing.
- Seed-based fuzzing.
- Model-based fuzzing and model-less fuzzing.
- Etc.

Black-box vs. White-box Fuzzing



VS.



Grey-Box Fuzzing

- White-box fuzzing (strictly speaking).
- Obtain “some” partial information about the program execution.

Mutation- vs. Generation-based Fuzzing

- ***Mutation-based***: mutate a given seed[3] to generate test cases.
- ***Generation-based***: generate test cases from a model.

[3] ***Seed*** is an input to a program.

Mutation- vs. Generation-based Fuzzing

- ***Mutation-based***: mutate a given seed[3] to generate test cases.
- ***Generation-based***: generate test cases from a model.

Which category is Miller's fuzzer?

[3] ***Seed*** is an input to a program.

Why Mutate Seeds?

Random inputs are likely to be rejected.

Mutation-based Fuzzing

Many research problems:

- Obtaining a good set of initial seeds.
- Mutation strategy.
- Seed selection strategy.

Generation-based Fuzzing

If the input ***model*** is precise, we can generate valid inputs.

- JavaScript interpreter fuzzing (with JavaScript grammar as a model).
- PNG parser fuzzing (with PNG file format as a model).
- Etc.

Is Concolic Execution Fuzzing?

Historically, Sage (which is a descendant of DART) is referred to as a white-box fuzzer[4]. So we often use the term 'white-box fuzzing' to refer to concolic execution.

[4] Automated Whitebox Fuzz Testing, ***NDSS 2008***.

Fuzzing Algorithm

Key Properties of Fuzzing

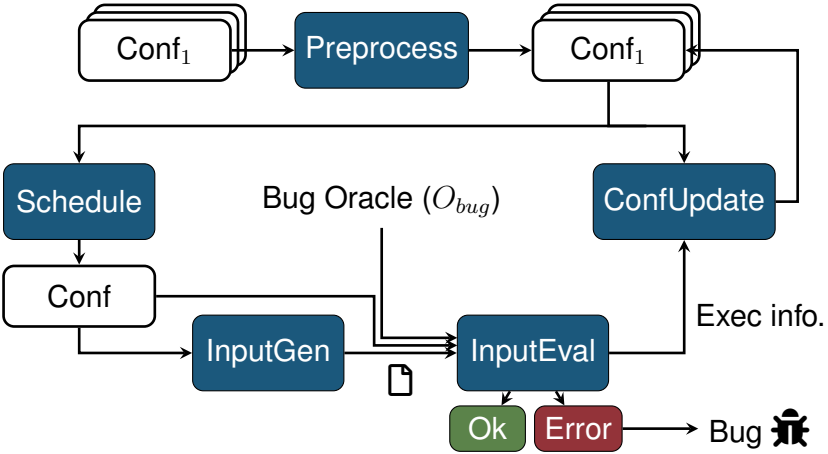
- Generate test cases.
- Run the program under test with the generated test cases.
- Check if the program misbehaves.

Definitions[5]

- ***Fuzzing*** is the execution of the program using input(s) sampled from an input space that protrudes the expected input space of the PUT.
- ***Fuzz testing*** is the use of fuzzing to test if a program violates a correctness policy (e.g., security policy).
- A ***fuzz configuration*** of a fuzz algorithm comprises the parameter value(s) that control(s) the fuzz algorithm.
- A ***bug oracle*** (O_{bug}) is a program, perhaps as part of a fuzzer that determines whether a given execution of the program violates a specific security policy.

[5]The Art, Science, and Engineering of Fuzzing: A Survey, ***TSE 2021***.

Algorithm


Conf$_1$
Preprocess
Conf$_1$
Schedule
Bug Oracle (O_{bug})
ConfUpdate
Conf
InputGen
InputEval
Exec info.
Ok
Error
Bug

Algorithm

Algorithm 1: Fuzz Testing

Input: $\mathbb{C}$, t_{limit}

Output: $\mathbb{B}$ // a finite set of bugs

1 $\mathbb{B} \leftarrow \varnothing$

2 $\mathbb{C} \leftarrow$ PREPROCESS($\mathbb{C}$)

3 while $t_{\text{elapsed}} < t_{\text{limit}} \wedge$ CONTINUE($\mathbb{C}$) do

4 $\quad$ conf $\leftarrow$ SCHEDULE($\mathbb{C}$, $t_{\textbf{\textit{elapsed}}}$, $t_{\textbf{\textit{limit}}}$)

5 $\quad$ tcs $\leftarrow$ INPUTGEN(***conf***)

$\quad$ // O_{bug} is embedded in a fuzzer

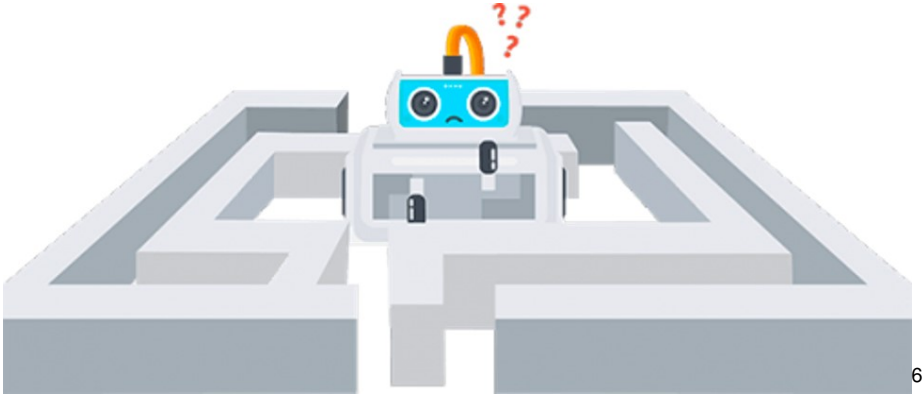
6 $\quad$ $\mathbb{B}'$, execinfos $\leftarrow$ INPUTEVAL(***conf***, ***tcs***, $O_{\textbf{\textit{bug}}}$)

7 $\quad$ $\mathbb{C} \leftarrow$ CONFUPDATE($\mathbb{C}$, ***conf***, ***execinfos***)

8 $\quad$ $\mathbb{B} \leftarrow \mathbb{B} \cup \mathbb{B}'$

9 return $\mathbb{B}$

Traditional AI



[6]

[6]Image from `http://arcbotics.com/lessons/maze-solving-home-lessons/`.

AI-based Maze Solving Robot

(Reinforcement Learning)

1. Move the agent.
2. Make observations.
3. Decide the next move based on the information gathered.
4. Go to 1 and iterate.

Fuzzing = AI-based Maze Solving

1. Run the program under test.
2. Observe the execution and gather execution information.
3. Generate new test cases based on the information gathered.
4. Go to 1 and iterate.

The Analogy (Fuzzing and Maze Solving)

	Fuzzing	Maze Solving
Target	Program	Maze
Goal	Finding a buggy path	Finding a solution path

Fuzzing is akin to exploring a maze!

Easy Questions

- **Q**: Which robot (AI) is better in maze solving?
- **Q**: Which fuzzer is better in fuzzing?

Easy Questions

- **Q**: Which robot (AI) is better in maze solving?
  - **A**: One that quickly solves mazes.
- **Q**: Which fuzzer is better in fuzzing?
  - **A**: One that quickly finds bugs in programs.

Hard Questions

- **Q**: On which mazes should we evaluate robots?
- **Q**: On which programs should we evaluate fuzzers?

Using Existing Benchmarks?

Robots (and fuzzers) may ***overfit*** to the benchmarks.

- One may develop a fuzzer that specifically targets a certain benchmark.
- Hence, one need to constantly develop new benchmarks, which is costly.

Hard Questions Revisited

- **Q**: On which mazes should we evaluate robots?
  - Use a random maze.
- **Q**: On which programs should we evaluate fuzzers?
  - Use a ***random program*** (?)

Bug Synthesis Problem

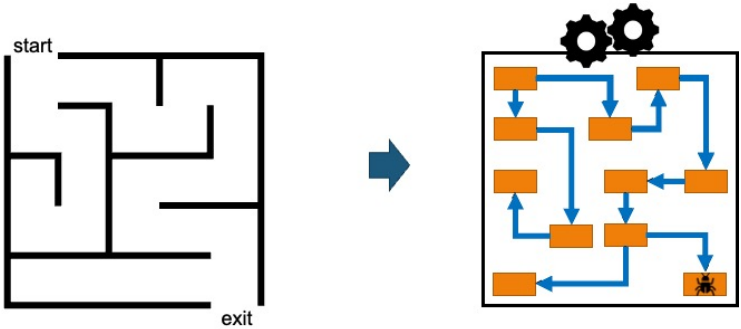
Can we synthesize a random buggy program for evaluating fuzzers?

Four Requirements for Bug Synthesis

- ***Reproducible***: Bug-trigger input should exist.
- ***Ground Truth***: Number and location of bugs should be known.
- ***Realistic***: Bugs should be triggered by exercising a realistic path.
- ***Uncorrelated***: Finding one bug should not change the detection probabilities of others.

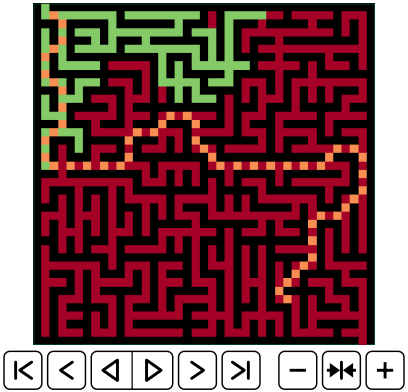
Fuzzle: Maze-based Synthesis[7]

Key insight: create a random program as if we are creating a random maze!



[7] Fuzzle: Making a Puzzle for Fuzzers, ***ASE 2022***.

Additional Benefit: Coverage Visualization



What's Next?

Fuzzing as an AI technique:

- ...

Question?