Fuzzing

Ian Fox

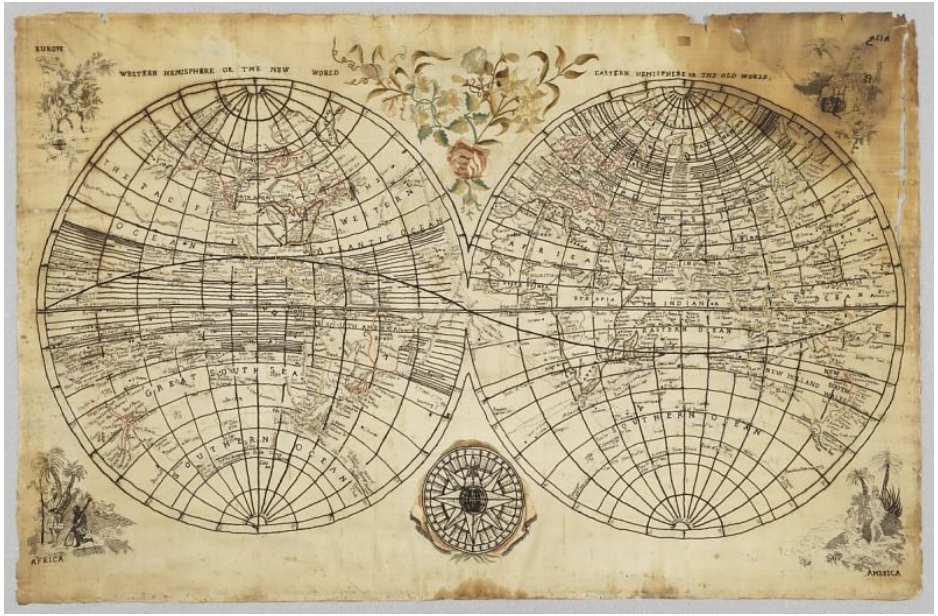
Who am I

- Security Researcher @ Omny
- Interested in security, reverse engineering, program analysis, operating systems

A different kind of fuzz

Outline

- Theory of fuzzing
  - Motivation
  - Basic fuzzing
  - More advanced techniques
- Examples



Testing Programs

- Why test programs?
- How to test programs?

Example Program

The following code from The Fuzzing Book has a bug:

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

Example Program: Manual Investigation

```
>>> my_sqrt(4)
2.0
>>> my_sqrt(2)
1.414213562373095
>>> my_sqrt(0)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "test.py", line 7, in my_sqrt
    guess = (approx + x / approx) / 2
ZeroDivisionError: float division by zero
>>> my_sqrt(-1)
_
```

Example Program: Static Analysis

```
1 def my_sqrt(x: float) -> float:
2     """Computes the square root of x, using the Newton-Raphson method"""
3     approx = None
4     guess = x / 2
5     while approx != guess:
6         approx = guess
7         guess = (approx + x / approx) / 2
8     return approx
```

```
$ mypy --strict test.py
test.py:8: error: Incompatible return value type (got "Optional[float]", expected "float"
Found 1 error in 1 file (checked 1 source file)
$ bandit -q ./test.py
$
```

Example Program: Automated Tests

```
def test_mysqrt():
    assert my_sqrt(4) == 2

def test_mysqrt_irrational():
    assert my_sqrt(2)**2 == pytest.approx(2)

def test_mysqrt_zero():
    assert my_sqrt(0) == 0
```

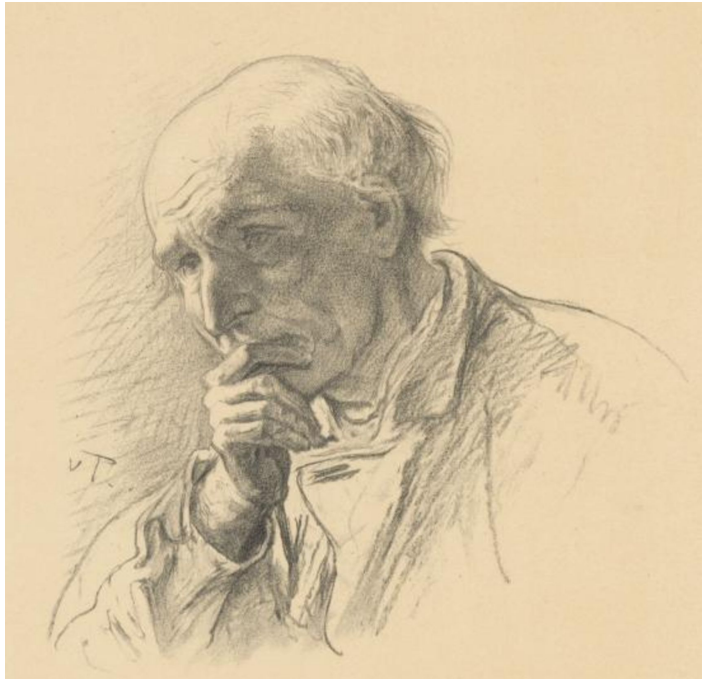
```
def should_be_true_for_all(num: float):
    assert my_sqrt(num) ** 2 == pytest.approx(num)

should_be_true_for_all(0)
should_be_true_for_all(2)
should_be_true_for_all(4)
```

Example Program: Runtime Checks

```
 1 def my_sqrt(x):
 2     """Computes the square root of x, using the Newton-Raphson method"""
 3     try:
 4       approx = None
 5       guess = x / 2
 6       while approx != guess:
 7           approx = guess
 8           guess = (approx + x / approx) / 2
 9       assert approx ** 2 == pytest.approx(x)
10       return approx
11     except Exception as e:
12       logger.critical(e)
13       raise
```

Automating the Automated Tests

- What were we doing when we tried to come up with test cases?
- Can we get a computer to think that way?
- Do we have to?



An empirical study of the reliability of UNIX utilities (1990)

It started 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.

…

[W]e were surprised that these spurious characters were causing programs to crash.

...

on receiving unusual input, [basic utilities] might exit with minimal error messages, but they should not crash. This experience led us believe that there might be serious bugs lurking in the systems that we regularly used.

Barton P. Miller, Lars Fredriksen, Bryan So

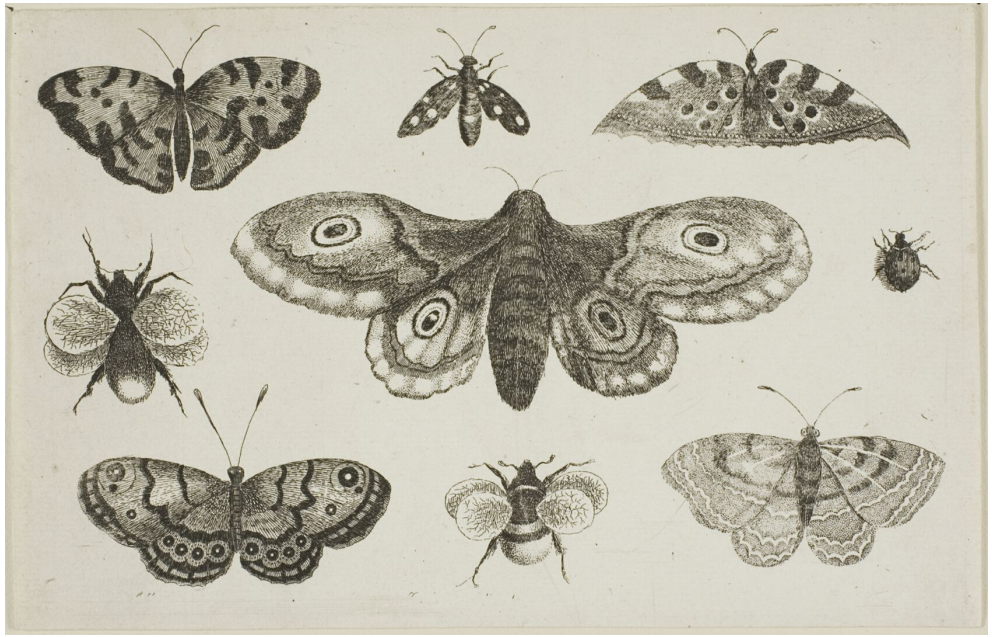
Basic Fuzzing

- Send random data (“fuzz”) to the program
- Wait for something interesting to happen
- Save the interesting input for a human to look at

```
while True:
    input_data = get_random_value()
    try:
        my_func_to_test(input_data)
    except:
        print(f"Found an exception with input {input_data}")
```

Defining “Interesting”

- Exceptions
- Crashes
- Timeouts
- Runtime instrumentation

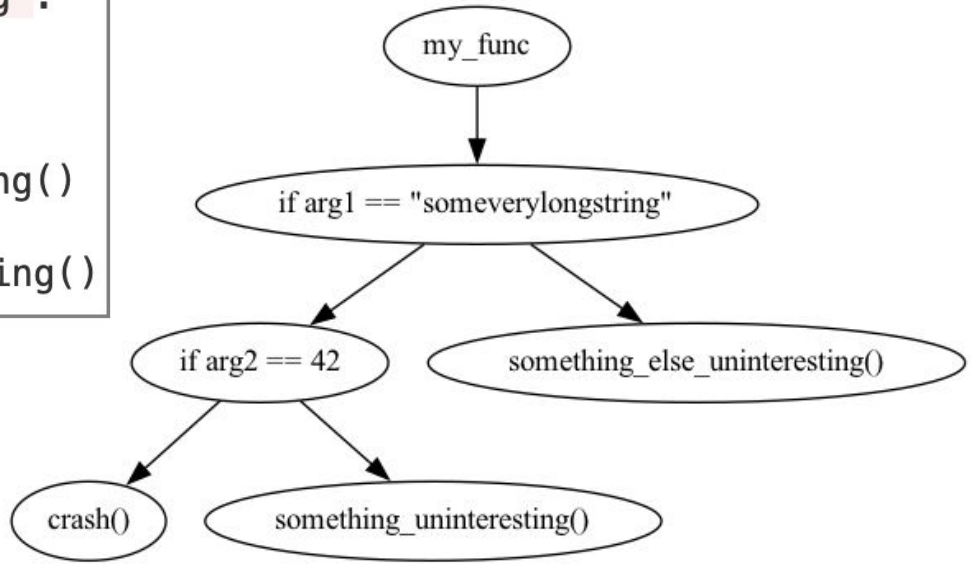


Generating Values

```
def get_string(max_len: int = 100, char_start: int = 32, char_range: int = 32) -> str:
    """A string of up to `max_len` characters
       in the range [`char_start`, `char_start` + `char_range`)"""
    string_length = random.randrange(0, max_length + 1)
    out = ""
    for i in range(0, string_length):
        out += chr(random.randrange(char_start, char_start + char_range))
    return out

def get_int():
    """A random integer, biased towards more 'interesting' values"""
    if random.randint(1, 10) > 3:
        return random.choice(-1, 0, 1, sys.maxsize, -sys.maxsize-1)
    return random.randint(-sys.maxsize-1, sys.maxsize)
```

Exploring a Program

```
1 def my_func(arg1: str, arg2: int):
2     if arg1 == "someverylongstring":
3         if arg2 == 42:
4             crash()
5         else:
6             something_uninteresting()
7     else:
8         something_else_uninteresting()
```



Mutation-Based Fuzzing

- Start with some “interesting” inputs
- Combine them to create new ones

Mutation-Based Fuzzing

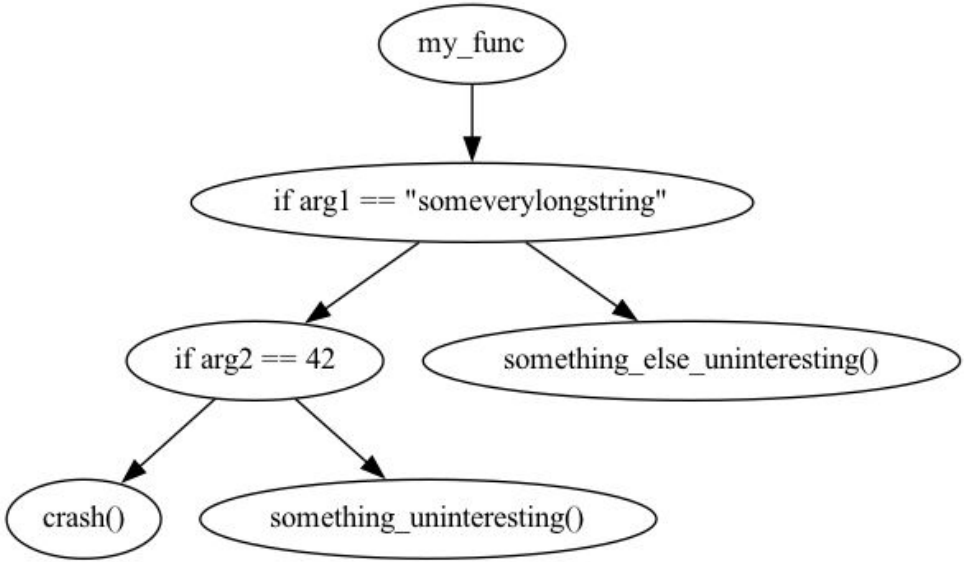
```
1 def mutate(input: str) -> str:
2   type_of_mutation = random.choice(["delete", "append", "bitflip", ...])
3   if type_of_mutation == "delete":
4     idx = random.randrange(0, len(input))
5     num_chars_to_delete = random.randrange(0, len(input) - idx)
6     return input[0:idx] + input[idx+num_chars_to_delete:]
7   elif type_of_mutation == "append":
8   ...
9
10 def get_string(corpus: Set[str]) -> str
11   new_string = mutate(random.choice(corpus))
12   corpus.add(new_string)
13   return new_string
```

Some Problems

- Difficulty exploring the entire program
- Corpus size

Code Coverage

- Block coverage vs branch coverage
  - A → B → C → D
  - A → C → B → D




```
# inserted at the start of every block
current_location_tag = INSTRUMENTATION_TIME_RANDOM
shared_coverage[(current_location_tag, previous_location_tag)] += 1
previous_location_tag = current_location_tag
```

Corpus Size

- Discard inputs which cause duplicate traces
- Try reducing the size of an input to see if the trace changes

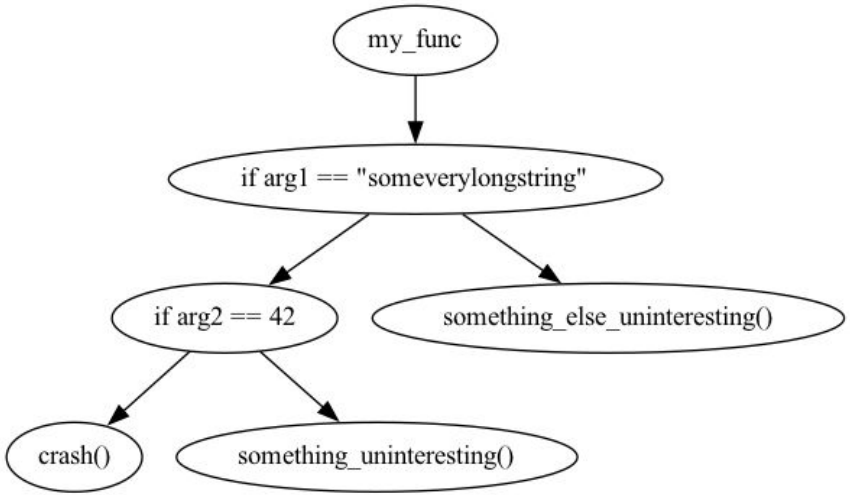
Grammars & Post-Processing

- Generate inputs according to a grammar
  - Choosing how to expand nodes
- Mine grammars from the program itself
- Calculate constrained portions of the input instead of relying on chance

1. $S \rightarrow a$
2. $S \rightarrow SS$
3. $aSa \rightarrow b$

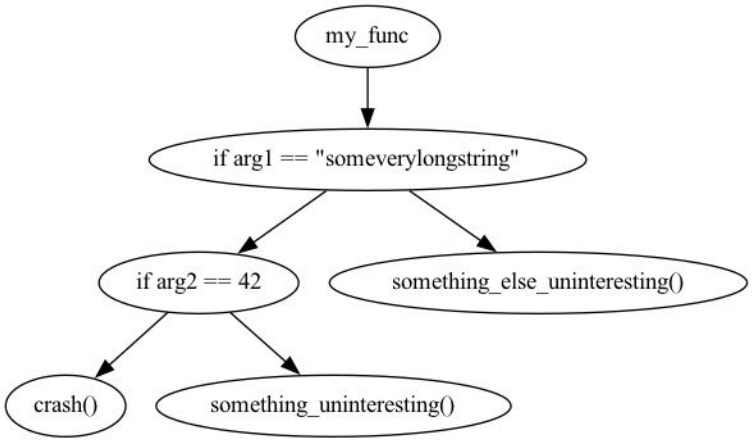
Power Schedules

- With information from coverage, bias towards more “promising” inputs



Symbolic & Concolic Execution

- SAT Solvers
  - (A ∧ ¬ B ∨ C)
- Other data types → SMT Solvers
  - (A = “someverylongstring” ∧ B = 42)
- Symbolic execution: explosion of paths
- Concolic: combine symbolic execution with fuzzing



Search-Based Techniques

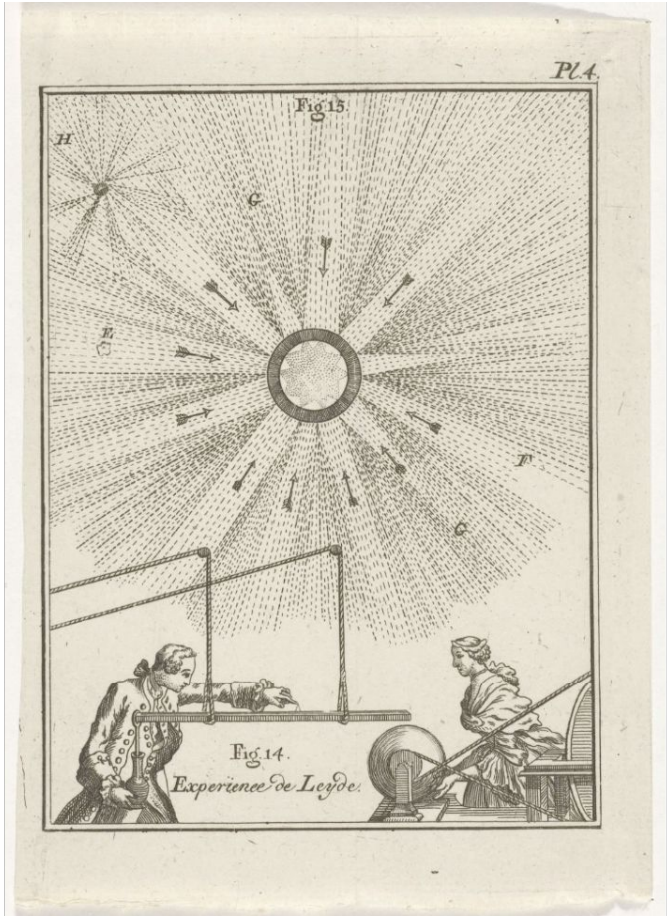
- Ideas from optimization
- Define
  - Search space
  - Fitness function
- Strategies
  - Hill climbing
  - Evolutionary algorithms
  - Genetic algorithms

Fuzzing Effectively

- Can’t provide hardcoded arguments to e.g. `expectEquals` → assert things that should always be true
- Targets should be as deterministic as possible
  - Not rely on randomness
  - Not rely on shared state which isn’t reset between runs
- Isolate small, fast components to test

Demos!

- Libfuzzer
  - Heartbleed/OpenSSL
- Golang
- Quickcheck variants



Further Reading

- Original fuzzing paper
- AFL technical details
- Fuzzing book
- Libfuzzer tutorial
- Golang tutorial
- Quickcheck on Wikipedia (links to many implementations)
- OSS Fuzz

Questions & Summary

- ian@whatthefox.dev, iansfox on LinkedIn
- Slides at whatthefox.dev/slides/fuzzing.pdf
- Questions!