AI 用于软件安全和漏洞挖掘

陈浩

University of California, Davis, USA

2020-12-04

Success of AI

- AI has seen great success in
  - vision
  - speech
  - NLP
- How useful is AI for security?

Software security

Traditional methods

- Static analysis
- Dynamic analysis
- Symbolic execution
- Model checking

Can we apply AI to software security?

Fuzzing

- A popular dynamic analysis technique
- Mutates inputs to trigger bugs in the program.
- Benefits
  - No false positive.
  - Provides witness inputs to trigger bugs.
  - Scalable
- Challenge: How to explore different states of the program?

Background

American Fuzzy Lop (AFL)

- Instruments the program to track branch coverage.
- Keeps inputs that explore new branches as seeds.
- Mutates inputs by heuristics.

Limitations

- Uses only heuristics, with no principled guidance.
- Cannot target specific branches.
- Random mutation of input is unproductive.

Fuzzing: from art to science

- Model fuzzing as an optimization problem.
- Apply principled mathematical tools.

Motivating example

```
void foo(int i, int j) {

  if (i * i - j * 2 > 0) {
    // some code
  } else {
    // some code
  }
}
int main() {
  char buf[1024];
  int i = 0, j = 0;
  if (fread(buf, sizeof(char), 1024, fp) < 1024) {
    return(1);
  }
  if (fread(\&i, sizeof(int), 1, fp) < 1) {
    return(1);
  }
  if (fread(\&j, sizeof(int), 1, fp) < 1) {
    return(1);
  }
  foo(i, j);
}
```

Motivating example

```
void foo(int i, int j) {
  // Byte-level taint tracking: which input bytes flow into this predicate?
  if (i * i - j * 2 > 0) {
    // some code
  } else {
    // some code
  }
}
int main() {
  char buf[1024];
  int i = 0, j = 0;
  if (fread(buf, sizeof(char), 1024, fp) < 1024) {
    return(1);
  }
  if (fread(\&i, sizeof(int), 1, fp) < 1) {
    return(1);
  }
  if (fread(\&j, sizeof(int), 1, fp) < 1) {
    return(1);
  }
  foo(i, j);
}
```

Motivating example

```
void foo(int i, int j) {
  // Search based on gradient descent: how to solve this path constraint efficiently?
  if (i * i - j * 2 > 0) {
    // some code
  } else {
    // some code
  }
}
int main() {
  char buf[1024];
  int i = 0, j = 0;
  if (fread(buf, sizeof(char), 1024, fp) < 1024) {
    return(1);
  }
  if (fread(\&i, sizeof(int), 1, fp) < 1) {
    return(1);
  }
  if (fread(\&j, sizeof(int), 1, fp) < 1) {
    return(1);
  }
  foo(i, j);
}
```

Byte-level taint tracking

Question Which input bytes flow into each path constraint?

Challenges Taint tracking is expensive, more so at byte-level.

Observation Taint tracking is unnecessary while mutating the input.

Solution

- Run taint tracking once on each seed input.
- Mutate the seed many times and run without taint tracking.
- Benefit: amortize the cost of taint tracking over many mutations.

Search based on gradient descent

Question How to mutate the input to solve path constraints?

Solution
- View each path constraint as a function over the relevant input bytes.
- Solve the constraint using optimization techniques, such as gradient descent.

View path constraint as function over input

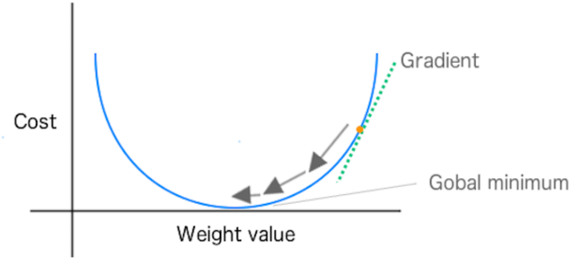
View each path constraint as $f(x)$ where

- x: a vector representing the relevant input bytes
- $f(x)$
  - Represents the computation from the program start.
  - Needs no analytic form.

Comparison	$f(\cdot)$	Constraint
$a < b$	$a - b$	$f(\cdot) < 0$
$a \leq b$	$a - b$	$f(\cdot) \leq 0$
$a > b$	$b - a$	$f(\cdot) < 0$
$a \geq b$	$b - a$	$f(\cdot) \leq 0$
$a = b$	$\mathrm{abs}(a - b)$	$f(\cdot) = 0$
$a \neq b$	$-\mathrm{abs}(a - b)$	$f(\cdot) < 0$

Gradient descent

- Goal: find a minimum of $f(x)$.
- Iterative algorithm
  1. Start: $x \leftarrow x_0$
  2. Repeat
     2.1 Compute $\nabla_x f(x)$
     2.2 $x \leftarrow x - \epsilon \nabla_x f(x)$ where ϵ: learning rate



Apply gradient descent to solving path constraints

Advantages

- In machine learning, the goal is to find the global minimum.
- In fuzzing, need only find a good enough solution $f(x) < 0$.

Challenges

In fuzzing, we cannot compute gradient directly because

- $f(x)$ has no closed form.
- $f(x)$ is not continuous because x is usually discrete.

Numerically approximate directional derivative

Directional derivative

$$\frac{\partial f(x)}{\partial x_i} = \lim_{\delta \to 0} \frac{f(x + \delta e_i) - f(x)}{\delta}$$

Approximation

Let δ be the smallest values of the type (usually 1 or -1).

Algorithm

1. Run program with the input x to get $f(x)$
2. For $i \in [1, n]$
   2.1 Run program with the input $x + \delta e_i$ to get $f(x + \delta e_i)$
   2.2
   $$\frac{\partial f(x)}{\partial x_i} \approx \frac{f(x + \delta e_i) - f(x)}{\delta}$$

Nested conditional statements

```
// pngrutil.c 2406
png_crc_read(png_ptr, buffer, length);
buffer[length] = 0;

if (png_crc_finish(png_ptr, 0) != 0)
  return;

if (buffer[0] != 1 && buffer[0] != 2)
{
  png_chunk_benign_error(png_ptr, "invalid unit");
  return;
}
```

Nested conditional statements

```
// pngrutil.c 2406
png_crc_read(png_ptr, buffer, length);
buffer[length] = 0;
// First conditional statement
if (png_crc_finish(png_ptr, 0) != 0)
  return;
// Second conditional statement
if (buffer[0] != 1 && buffer[0] != 2)
{
  png_chunk_benign_error(png_ptr, "invalid unit");
  return;
}
```

Nested constraints are difficult to solve

Program	Percentage of nested constraints in	
	all unsolved constraints	all constraints
djpeg	90.00 %	75.65 %
file	86.49 %	44.14 %
jhead	57.95 %	51.53 %
mutool	80.88 %	58.63 %
nm	84.32 %	68.16 %
objdump	90.54 %	73.95 %
readelf	84.12 %	70.50 %
readpng	94.02 %	89.50 %
size	87.86 %	71.46 %
tcpdump	96.15 %	78.98 %
tiff2ps	75.56 %	62.18 %
xmlint	78.18 %	72.37 %
xmlwf	96.18 %	68.16 %

Control and data flow dependencies of nested conditional statements

```
void foo(unsigned x, unsigned y, unsigned z) {

  if (w + x < 2) {

    if (x + y < 3) {

      if (a == 1111) {
        if (y + z == 2222) {
          ...
        }
        if (y > 1) {
          ...
        } else {
          ...
        }
      }
    }
  }
}
```

Control and data flow dependencies of nested conditional statements

```
void foo(unsigned x, unsigned y, unsigned z) {
  // control flow dependency
  if (w + x < 2) {
    // control flow dependency
    if (x + y < 3) {
      // control flow dependency
      if (a == 1111) {
        if (y + z == 2222) {
          ...
        }
        if (y > 1) {
          ...
        } else {
          ...
        }
      }
    }
  }
}
```

Control and data flow dependencies of nested conditional statements

```
void foo(unsigned x, unsigned y, unsigned z) {
  // data flow dependency
  if (w + x < 2) {
    // data flow dependency
    if (x + y < 3) {

      if (a == 1111) {
        if (y + z == 2222) {
          ...
        }
        if (y > 1) {
          ...
        } else {
          ...
        }
      }
    }
  }
}
```

Solve nested constraints

1. Determine control flow dependency
   - based on post-dominator
2. Determine data flow dependency
   - based on dynamic taint analysis
3. Solve constraints
   - Prioritize reachability
   - Prioritize satisfiability
   - Joint optimization

Joint optimization

- Problem: find x such that $f_i(x) = 0, \forall i \in [1, n]$
- Define

$$g(x) = \sum_{i=1}^{n} R(f_i(x))$$

  where the rectifier $R(x) \equiv 0 \vee x$
- $g(x) = 0 \Rightarrow f_i(x) \leq 0, \forall i \in [1, n]$
- Use gradient descent to solve the constraint.

Joint optimization

```
void foo(unsigned x, unsigned y, unsigned z) {

  if (w + x < 2) {

    if (x + y < 3) {
      if (a == 1111) {

        if (y + z == 2222) {
          ...
        }
      }
    }
  }
}
```

Joint optimization

```
void foo(unsigned x, unsigned y, unsigned z) {
  // f_1(·) = w + x − 2 + ϵ
  if (w + x < 2) {
    // f_2(·) = x + y − 3 + ϵ
    if (x + y < 3) {
      if (a == 1111) {
        // f_3(·) = y + z − 2222
        if (y + z == 2222) {
          ...
        }
      }
    }
  }
}
```

Joint optimization

```
void foo(unsigned x, unsigned y, unsigned z) {
  // f_1(·) = w + x − 2 + ε
  if (w + x < 2) {
    // f_2(·) = x + y − 3 + ε
    if (x + y < 3) {
      if (a == 1111) {
        // f_3(·) = y + z − 2222
        if (y + z == 2222) {
          ...
        }
      }
    }
  }
}
```

$$g(\cdot) \equiv \sum_{i=1}^{3} R(f_i(\cdot))$$

$g(\cdot) = 0 \Rightarrow$ all constraints are satisfied

Implementation

Instrumentation

- Instrument LLVM Pass.
- Extend DFSAN with byte-level taint tracking.
- Size: 1262 line C++

Fuzzer

Size: 8672 line Rust

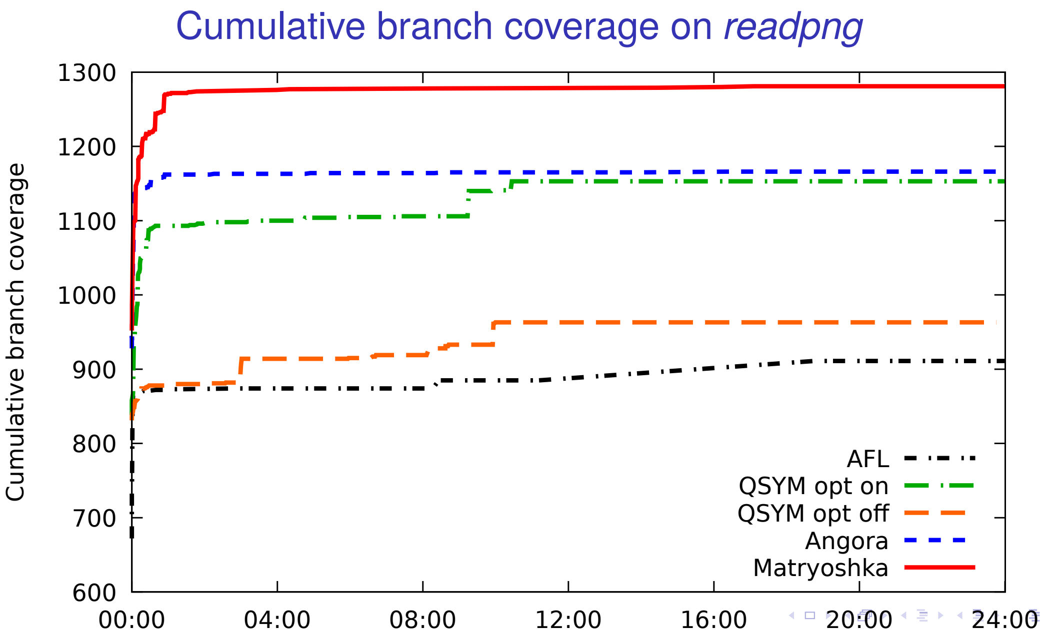
Bugs found on the LAVA-M data set

Program	Listed bugs	Bugs found by					
		AFL	QSYM	NEUZZ	REDQUEEN	Angora	**Matryoshka**
uniq	28	9	28	29	29	29	29
base64	44	0	44	48	48	48	48
md5sum	57	0	57	60	57	57	57
who	2136	1	1238	1582	2462	1541	2432

Verified bugs

Program	Types of bugs				
	SBO	HBO	OOM	OBR	Total
file	4				4
jhead	2	15		6	23
nm	1	1			2
objdump			3	1	4
size		1	1		2
readelf		4			4
tiff2ps		1	1		2

- SBO: stack buffer overflow
- HBO: heap buffer overflow
- OOM: out of memory
- OBR: out of bound read


Cumulative branch coverage on *readpng*
Cumulative branch coverage
1300
1200
1100
1000
900
800
700
600
00:00
04:00
08:00
12:00
16:00
20:00
24:00
AFL
QSYM opt on
QSYM opt off
Angora
Matryoshka

Conclusion

- AI is a powerful tool for software security.
- Formulate security problem as AI/optimization problem.
- AI and traditional program analyses are mutually beneficial.

Angora

Matryoshka