A Tight Integration of Symbolic Execution and Fuzzing

(short paper)

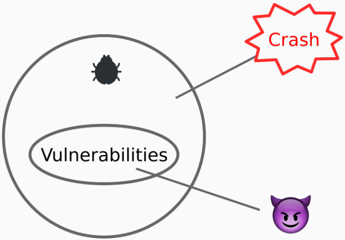
Yaëlle Vinçont[1,2], Sébastien Bardin[2], Michaël Marcozzi[2]

International Symposium on Foundations & Practice of Security 2021

[1]Laboratoire Méthodes Formelles, Université Paris-Saclay

[2]CEA, List, Université Paris-Saclay

Vulnerabilities



Heartbleed

2014

BigSig

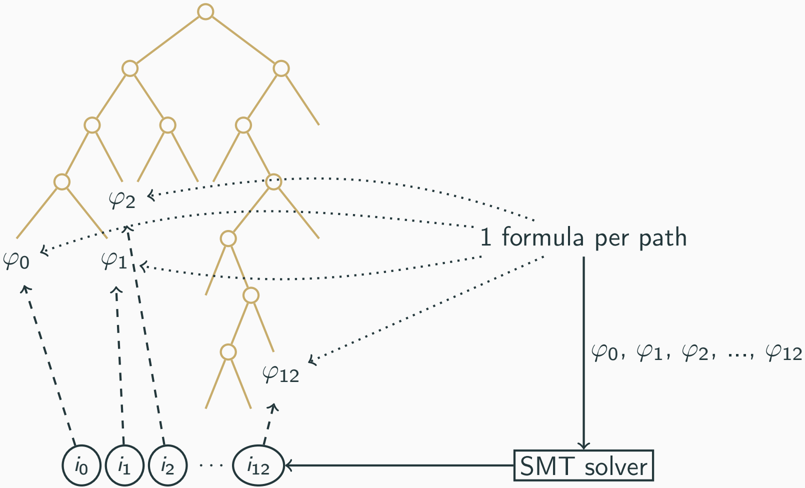
moz://a

2021

Goal

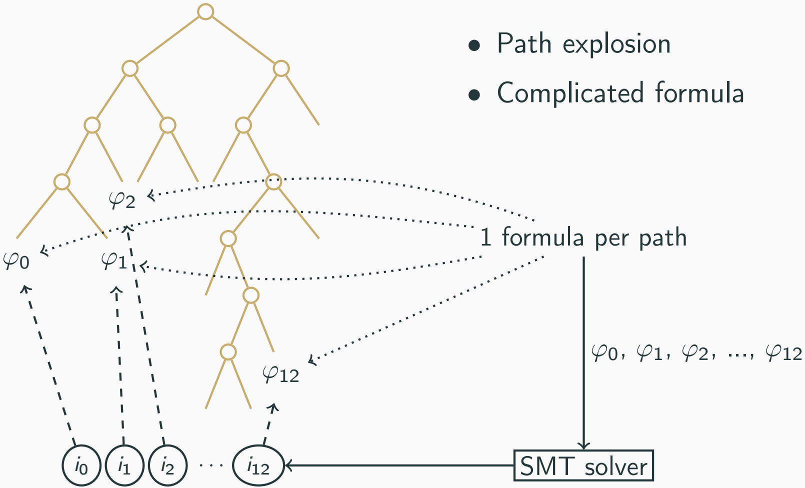
Automatically test programs to find bugs

Symbolic Execution



Examples: KLEE, BINSEC, Angr, Manticore...

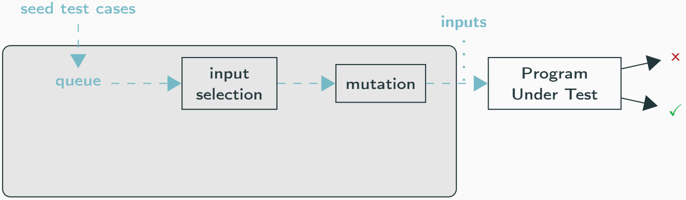
Symbolic Execution



- Path explosion
- Complicated formula

Examples: KLEE, BINSEC, Angr, Manticore...

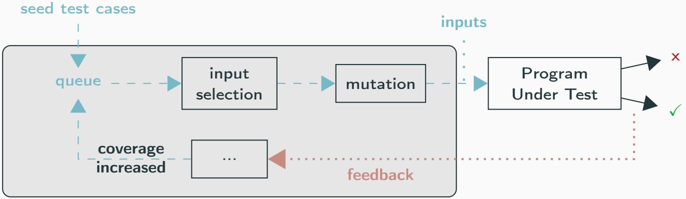
Fuzzing



Fuzzing

Examples: AFL, Radamsa, FairFuzz, Steelix...

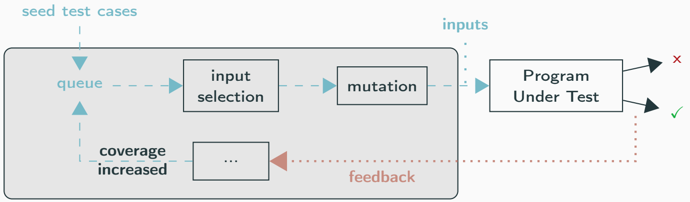
Fuzzing



Coverage-based Greybox **Fuzzing**

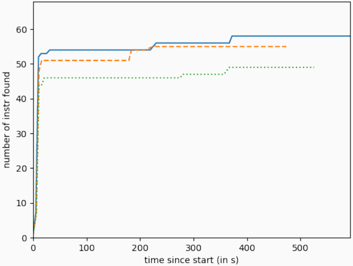
Examples: AFL, Radamsa, FairFuzz, Steelix...

Fuzzing



Coverage-based Greybox **Fuzzing**

Examples: AFL, Radamsa, FairFuzz, Steelix...



Our goal

Mixing test generation techniques, to get *power of SE* and *lightness of fuzzing*:

- efficient approach
- reason about complex code
- quick and easy input generation

Pitfall: getting the worst of both worlds

Challenges

w.r.t. symbolic reasoning

- cheap [solver-less]
- targets interesting paths
- correct
- integrated with fuzzer

w.r.t. fuzzing

- efficient
- solves constraints

Positioning

	Analysis				Fuzzing		
	Symbolic	Cheap	Targeted	Correct	Efficient	Constraints	Well-integrated components
Fuzzing	-	-	-	-	✓	×	-
SE	✓	×	×	✓	-	-	-
Driller	✓	×	✓	✓	✓	×	×
Qsym	✓	✓	✓	×	✓	×	×
Pangolin	✓	✓	✓	×	✓	✓	✓
Angora	×	✓	✓	×	∼	✓	ok
Matryoshka	×	✓	✓	×	∼	✓	ok
Eclipser	×	✓	×	×	✓	×	×
ConFuzz	**✓**	**✓**	**✓**	**✓**	**✓**	**✓**	**✓**

Our proposal

Lightweight Symbolic Execution

- variant of Dynamic SE [Targeted & correct]
- target easily-enumerable constraints [Cheap & integrated]

leads exploration past specific conditions

Constrained Fuzzer

- based on AFL [Efficient]
- takes seed & easily-enumerable constraint [Cheap & solves constraints]

efficiently creates seeds, including solutions to constraints

Contents

Motivating example

```
int main(int argc, char** argv) {

  char buf[BUF_LENGTH];
  int x, y;

  int res = read(0, buf, BUF_LENGTH);

  if (res < BUF_LENGTH) {
    printf("entry too small\n");
    return 0;
  }

  int cpt;
  for (cpt = 16; cpt < 36; cpt++) {
    if (buf[cpt] == cpt % 20)
      y += 1;
  }

  if (buf[0] == 'a')
    if (buf[4] == 'F')
      if (buf[7] == '6')
        if (buf[12] == 'g')
          if (buf[15] == 'L')
            x = 1;
          else
            x = 2;
        else
          x = 3;
      else
        x = 4;
    else
      x = 5;
  else
    x = 6;

  return 0;
}
```

Loop with independent conditions
0/10/20 iterations

Serie of nested conditions

Motivating example

```c
int main(int argc, char** argv) {

  char buf[BUF_LENGTH];
  int x, y;

  int res = read(0, buf, BUF_LENGTH);

  if (res < BUF_LENGTH) {
    printf("entry too small\n");
    return 0;
  }

  int cpt;
  for (cpt = 16; cpt < 36; cpt++) {
    if (buf[cpt] == cpt % 20)
      y += 1;
  }

  if (buf[0] == 'a')
    if (buf[4] == 'F')
      if (buf[7] == '6')
        if (buf[12] == 'g')
          if (buf[15] == 'L')
            x = 1;
          else
            x = 2;
        else
          x = 3;
      else
        x = 4;
    else
      x = 5;
  else
    x = 6;

  return 0;
}
```

Fuzzing

Loop: isn't aware of it, no problem

Nested conditions: struggles finding a solution

Motivating example

```
int main(int argc, char** argv) {

  char buf[BUF_LENGTH];
  int x, y;

  int res = read(0, buf, BUF_LENGTH);

  if (res < BUF_LENGTH) {
    printf("entry too small\n");
    return 0;
  }

  int cpt;
  for (cpt = 16; cpt < 36; cpt++) {
    if (buf[cpt] == cpt % 20)
      y += 1;
  }

  if (buf[0] == 'a')
    if (buf[4] == 'F')
      if (buf[7] == '6')
        if (buf[12] == 'g')
          if (buf[15] == 'L')
            x = 1;
          else
            x = 2;
        else
          x = 3;
      else
        x = 4;
    else
      x = 5;
  else
    x = 6;

  return 0;
}
```

Fuzzing

Loop: isn't aware of it, no problem

Nested conditions: struggles finding a solution

SE

Loop: tries to explore every paths, path explosion

Nested conditions: solves with SMT solver

Motivating example

```c
int main(int argc, char** argv) {

  char buf[BUF_LENGTH];
  int x, y;

  int res = read(0, buf, BUF_LENGTH);

  if (res < BUF_LENGTH) {
    printf("entry too small\n");
    return 0;
  }

  int cpt;
  for (cpt = 16; cpt < 36; cpt++) {
    if (buf[cpt] == cpt % 20)
      y += 1;
  }

  if (buf[0] == 'a')
    if (buf[4] == 'F')
      if (buf[7] == '6')
        if (buf[12] == 'g')
          if (buf[15] == 'L')
            x = 1;
          else
            x = 2;
        else
          x = 3;
      else
        x = 4;
    else
      x = 5;
  else
    x = 6;

  return 0;
}
```

Fuzzing

Loop: isn't aware of it, no problem

Nested conditions: struggles finding a solution

SE

Loop: tries to explore every paths, path explosion

Nested conditions: solves with SMT solver

ConFuzz

Loop: not really aware of it

Nested conditions: LSE finds constraints, CF solves them

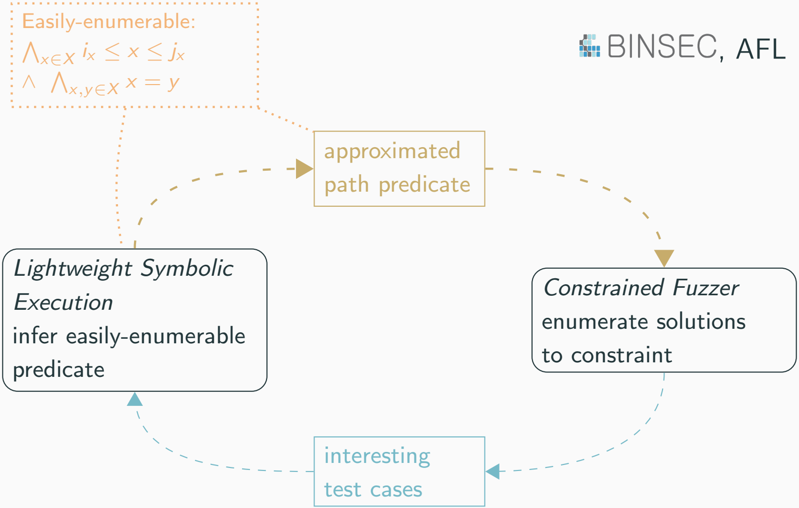
Motivating example - results

Ran 10 times, 20 minutes, KLEE, AFL, ConFuzz, with 0 and 20 loop iterations

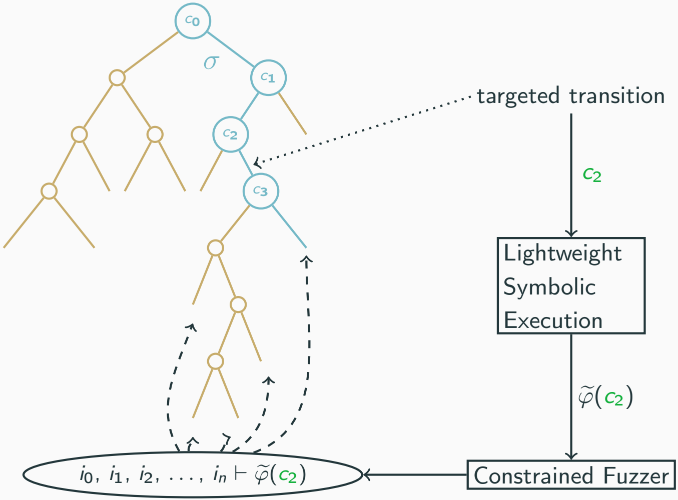
			AFL	KLEE	ConFuzz
0 iterations	**Nb success/Nb tries**		9/10	10/10	10/10
	Time (s) to cover	**Avg**	247.3	0.3	1.0
	all branches	**Dev** (σ)	347.6	0.1	0.2
20 iterations	**Nb success/Nb tries**		9/10	10/10	10/10
	Time (s) to cover	**Avg**	245.6	132.6	1.4
	all branches	**Dev** (σ)	354.9	9.5	0.2

Contents

General Principle



Example



[$c \triangleq c = \texttt{True}$, $\widetilde{\varphi}(c) \triangleq$ easily-enumerable path predicate for the path up to c]

Key challenge: easily-enumerable path constraints

how to define it?

how to compute it?

Easily-Enumerable Constraint Language

[X: input variables, i, j: integers]

Definition (Easily-Enumerable)

Complexity enumerating n solutions: $\mathcal{O}(n \times |X|)$

Definition (Our Constraint Language)

$$\widetilde{\varphi} \triangleq \bigwedge_{x \in X} i_x \leq x \leq j_x \;\wedge\; \bigwedge_{x,y \in X} x = y$$

Easily-enumerable Path Predicate - Example

$i = \{x : 0 \; ; \; y : 1 \; ; \; z : 2 \; ; \; t : 4 \; ; \; v : 5\}$

Program P

```
a = x + 3;
if (a <= 4) {
  b = y;
  e = t;
}
else {
  b = 2;
}
if (b != z) {
  d = 4;
}
else if (c != v) {
  d = 3;
}
```

Trace σ

```
declare x, y, z, t, v;
define a = x + 3;
assert (a <= 4);
define b = y;
define c = t;
assert (b == z);
assert (c != v);
define d = 3;
```

$\varphi(c \neq v)$

$$x \leq 1 \wedge y = z \wedge t \neq v$$

Path predicate

$\widetilde{\varphi}(c \neq v)$

$$x \leq 1 \wedge y = z \wedge t = 4 \wedge v = 5$$

Easily-enumerable path predicate

Computing $\widetilde{\varphi}$ - Backward Dynamic Analyses

Inferring the constraints

$i = \{x : 0 \ ; \ y : 1 \ ; \ z : 2 \ ; \ t : 4 \ ; \ v : 5\}$

```
declare x,y,z,t,v;
define a = x + 3;
assert (a <= 4);
define b = y;
define c = t;
assert (b == z);
assert (c != v);
define c = 3;
```

Computing $\widetilde{\varphi}$ - Backward Dynamic Analyses

Inferring the constraints

$i = \{x : 0 \; ; \; y : 1 \; ; \; z : 2 \; ; \; t : 4 \; ; \; v : 5\}$

```
declare x,y,z,t,v;
define a = x + 3;
assert (a <= 4);
define b = y;
define c = t;
assert (b == z);
assert (c != v);
define c = 3;
```

- **assert** (c != v);
  - concretization
  - backtrack on **define** c = t
  - t, v: input variables
  - $i[t] = 4$, $i[v] = 5$

$cstr := t = 4 \wedge v = 5$

Computing $\widetilde{\varphi}$ - Backward Dynamic Analyses

Inferring the constraints

$i = \{x : 0 \; ; \; y : 1 \; ; \; z : 2 \; ; \; t : 4 \; ; \; v : 5\}$

```
declare x,y,z,t,v;
define a = x + 3;
assert (a <= 4);
define b = y;
define c = t;
assert (b == z);
assert (c != v);
define c = 3;
```

- $cstr := t = 4 \wedge v = 5$
- **assert** (b == z);
  - equality analysis
  - backtrack on **define** b = y
  - y, z: input variables

$cstr := y = z$

Computing $\widetilde{\varphi}$ - Backward Dynamic Analyses

Inferring the constraints

$i = \{x : 0 \; ; \; y : 1 \; ; \; z : 2 \; ; \; t : 4 \; ; \; v : 5\}$

```
declare x,y,z,t,v;
define a = x + 3;
assert (a <= 4);
define b = y;
define c = t;
assert (b == z);
assert (c != v);
define c = 3;
```

- $cstr := t = 4 \wedge v = 5$
- $cstr := y = z$
- **assert** (a <= 4);
  - value analysis: $a \leq 4$
  - backtrack on **define** a = x + 3: $x \leq 1$
  - x: input variable

$cstr := x \leq 1$

Computing $\widetilde{\varphi}$ - Backward Dynamic Analyses

Inferring the constraints

$i = \{x : 0 \; ; \; y : 1 \; ; \; z : 2 \; ; \; t : 4 \; ; \; v : 5\}$

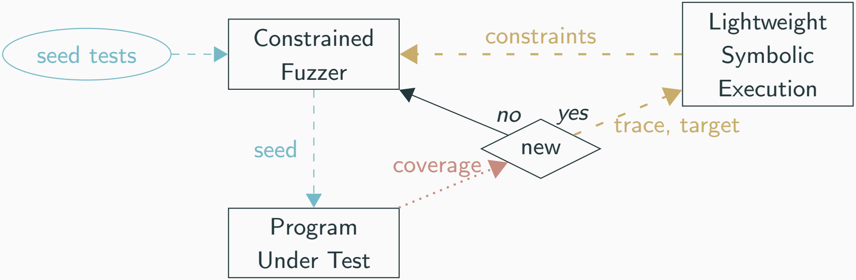
```
declare x,y,z,t,v;
define a = x + 3;
assert (a <= 4);
define b = y;
define c = t;
assert (b == z);
assert (c != v);
define c = 3;
```

- $cstr := t = 4 \wedge v = 5$
- $cstr := y = z$
- $cstr := x \leq 1$

$\widetilde{\varphi}(c! = v)$

$x \leq 1 \wedge y = z \wedge t = 4 \wedge v = 5$

Integration with Constrained Fuzzing



Contents

Implementation - ConFuzz

Lightweight Symbolic Execution

- BINSEC
- 6kloc OCaml
- only $i \leq x \leq j$ and concretization

Constrained Fuzzer

- AFL
- 4kloc C
- modifed mutations to make them constrained

Protocol

Tools

- ConFuzz
- AFL [it was built on]
- AFL++ [SoA fuzzing]
- KLEE [SoA SE]

Benchmark

- LAVA-M: real-world programs, with injected bugs
- Metric: number detected bugs
- 1 hour timeout
- Stats on 5 runs

Vargha-Delaney statistic ($\hat{A}_{12}$)

Probability for ConFuzz to do better than compared technique

Results

		AFL	AFL++	KLEE	ConFuzz
base64	**Avg**	0	0.2	10.0	38.8
3kloc	**Dev** (σ)	0	0.4	1.3	0.4
44 bugs	$\hat{A}_{12}$	1.0	1.0	1.0	-
md5sum	**Avg**	0	0	0	9
3kloc	**Dev** (σ)	0	0	0	1.7
57 bugs	$\hat{A}_{12}$	1.0	1.0	1.0	-
uniq	**Avg**	0	0.4	5	26.9
3kloc	**Dev** (σ)	0	0.5	0	3.6
28 bugs	$\hat{A}_{12}$	1.0	1.0	1.0	-

Conclusion

- *Lightweight Symbolic Execution*
  - uses easy-to-enumerate path predicates
  - no need for constraint solver
  - offers guarantees on solutions
- integrated with *Constrained Fuzzer*
  - quickly generate solutions
- $\Rightarrow$ promising early results
- Future work
  - formalize "easy-enumerability"
  - extend the constraint language
  - more extensive experimentation

Find BINSEC: binsec.github.io and @BinsecTool!