EXERCISE #31

CONCOLIC EXECUTION REVIEW

Write your name and answer the following on a piece of paper

What is the benefit of concolic execution over symbolic execution? How does it compare in terms of soundness / completeness of vulnerability finding?

Quiz 3 is on Friday

ADMINISTRIVIA AND ANNOUNCEMENTS

BOOLEAN SATISFIABILITY

EECS 677: Software Security Evaluation

Drew Davidson

WHERE WE'RE AT

TOOLS / TECHNIQUES UNDERLYING
SYMBOLIC EXECUTION

PREVIOUSLY: ENHANCING SYMBOLIC EXECUTION

OUTLINE / OVERVIEW

GENERATING TEST CASES

PRIORITIZING STATES IN THE SYMBOLIC EXECUTION TREE

PRUNING DUPLICATE STATES

CONCRETIZING (SOME) INPUT TO MAKE PROGRESS

THIS TIME: SATISFIABILITY

OUTLINE / OVERVIEW

THE MAGIC THAT MADE SYMBOLIC EXECUTION WORK WAS THE SOLVER

Determines if a path constraint is feasible

Induces a test case that satisfies the path constraint

Allows for consistent concretization

$$z = \gamma$$
$$\gamma > 5 \wedge \gamma < 9$$

BOOLEAN SATISFIABILITY

SAT AND SMT

At the root of the solver is a mechanism for solving a hard problem:

Given a Boolean expression, provide a satisfying assignment to its variables or indicate no such assignment is possible

n^2 2^n

The search for a solution requires a lot of computation

Constant time

Linear time

n log n time

polynomial time

Exponential time

Search scales rapidly with the size of the problem

$B \wedge A$ ← $B = 1$ $A = 1$

$B \vee A$ ← $B = 0$ $A = 1$, $B = 1$ $A = 0$, $B = 1$ $A = 1$

$\neg A \wedge A$ ← No solution

NP

SAT AND SMT

The class of problems where...

A solution can be generated in polynomial time by a nondeterministic Turing machine

A solution can be verified in polynomial time by a deterministic Turing machine

$$P \stackrel{?}{=} NP$$

NP-COMPLETENESS

SAT AND SMT

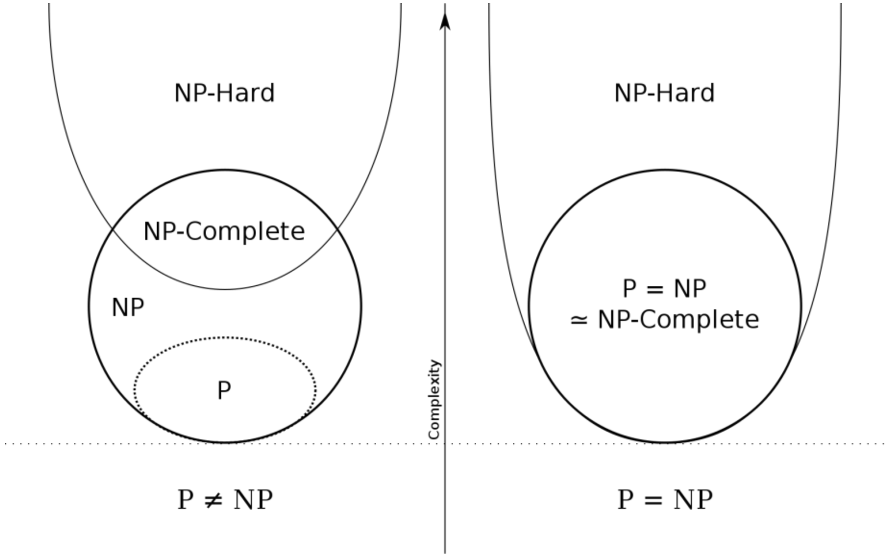
THE CLASS OF PROBLEMS WHERE…

A solution could be used as a solver for any problem in NP

most difficult

The “~~hardest~~ problems in NP”

SAT is the canonical example of an NP-Complete problem



THE MARVEL OF ENGINEERING

SAT AND SMT

NP Reductions once were used to to show that a problem was difficult, now they are used to show that a problem is do-able

HOW DO SAT SOLVERS WORK?

SAT AND SMT

Naïve Solution: Exponential time 2^n

$(a) \land (b \lor c) \land (\neg a \lor c \lor d) \land (\neg c \lor d) \land (\neg c \lor \neg d \lor \neg a) \land (b \lor d)$

CONJUNCTIVE NORMAL FORM

SAT AND SMT

A CONVENIENT REPRESENTATION FOR A BOOLEAN EXPRESSION

Any Boolean expression can be represented as a conjunction of disjunctions using the standard Boolean transformations

$\neg(P \lor Q) \Longleftrightarrow \neg P \land \neg Q$

$\neg(P \land Q) \Longleftrightarrow \neg P \lor \neg Q$

$\neg\neg P \Longleftrightarrow P$

$(P \land (Q \lor R)) \Longleftrightarrow ((P \land Q) \lor (P \land R))$

$(P \lor (Q \land R)) \Longleftrightarrow ((P \lor Q) \land (P \lor R))$

$(A \lor \neg A) \land \ldots \land$

$A \land (B \lor C)$

$(A \lor B) \land (A \lor C)$

$A \land B \land (D \land (B \land A))$

UNIT PROPAGAION

SAT AND SMT

UNIT PROPAGATION

$\boxed{(a)} \wedge (b \vee c) \wedge (\neg a \vee c \vee d) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d \vee \neg a) \wedge (b \vee d)$

a = true
b = true
c = false
d = true

$\Downarrow$

a literal that exists all alone in a clause is a unit

unit prop. $\rightarrow$ $(b \vee c) \wedge (\text{false} \vee c \vee d) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d \vee \text{false}) \wedge (b \vee d)$

$\Downarrow$

$(b \vee c) \wedge (c \vee d) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d) \wedge (b \vee d)$

pure literal elimination $\rightarrow$ $(c \vee d) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d)$

c true?

$d \wedge \neg d$

d true, c false

$\neg c$

PURE LITERAL ELIMINATION

SAT AND SMT

~~UNIT PROPAGATION~~ Pure literal elimination

(a) ∧ (b ∨ c) ∧ (¬a ∨ c ∨ d) ∧ (¬c ∨ d) ∧ (¬c ∨ ¬d ∨ ¬a) ∧ (b ∨ d)

a literal that occurs only positively, or only negatively, in a formula is pure

DPLL

SAT AND SMT

(a) ∧ (b ∨ c) ∧ (¬a ∨ c ∨ d) ∧ (¬c ∨ d) ∧ (¬c ∨ ¬d ∨ ¬a) ∧ (b ∨ d)

```
function DPLL(φ)
        if φ = true then
                return true
        end if
        if φ contains a false clause then
                return false
        end if
        for all unit clauses l in φ do
                φ ← UNIT-PROPAGATE(l, φ)
        end for
        for all literals l occurring pure in φ do
                φ ← PURE-LITERAL-ASSIGN(l, φ)
        end for
        l ← CHOOSE-LITERAL(φ)
        return DPLL(φ ∧ l) ∨ DPLL(φ ∧ ¬l)
end function
```

$\cancel{a} \wedge \cancel{(a \vee d)} \wedge (\neg a \vee c)$

NO MAGIC BULLET

OUTLINE / OVERVIEW

WE KNOW SOME CONSTRAINTS ARE COMPUTATIONALLY HARD TO UNPACK

```
int main(){
  char s[80];
  scanf("%s", s);
  if (sha256sum(s) == c01b39c7a35ccc3b081a3e83d2c7    a767ebfeb45c6
}
```

NO MAGIC BULLET

OUTLINE / OVERVIEW

WE KNOW SOME CONSTRAINTS ARE COMPUTATIONALLY HARD TO UNPACK

```
int main(){
  char s[80];
  scanf("%s", s);
  if (sha256sum(s) == c01b39c7a35ccc3b081a3e83d2c71fa9a767ebfeb45c69f08e17dfe3ef375a7b){
    return 1 / 0;
  }
}
```

$$\text{if}\,(a^n + b^n == c^n \wedge n > 2)\{$$

$$\}$$

FROM SAT TO SMT

OUTLINE / OVERVIEW

Next Time...

Symbolic execution requires path constraints far more complex than Boolean expressions.

Although a naïve reduction is somewhat straightforward, naivety does not gel well with NP-completeness