# Constraint solving for reverse engineers

Tim Blazytko
⟨tim.blazytko@rub.de⟩

Ruhr-Universität Bochum

23rd February 2017

# Today

- What are SMT solvers?
- How do they work?
- What can we do with them?

# Motivation

## Constraints

```
bool check(uint64_t key)
{
  if (key < 7)
  {
    return (key * 3 > 15);
  }
    return 0;
}
```

1. $key < 7$
2. $3 \cdot key > 15$

$\Rightarrow$ $(key < 7) \land (key > 5)$

$\Rightarrow$ $key = 6$

# Motivation

## Complex constraints

```
bool check(uint64_t key)
{
    return key * key * key * key * key * key * key == 0x90de757572b51cd3;
}
```

We may ask three questions:

- Does a solution exist?
- What is a solution?
- How many solutions do exist?

# Motivation

## Semantic equivalence

$$f(x, y) := (x \oplus y) + 2 \cdot (x \wedge y)$$

We observe

- $f(1, 1) = 2$
- $f(2, 3) = 5$
- $f(10, 20) = 30$

We ask ourselves if

$$x + y \stackrel{?}{=} (x \oplus y) + 2 \cdot (x \wedge y)$$

# Satisfiability modulo theories (SMT)

## What are SMT solvers?

### SAT

Is $(a \vee \neg c) \wedge (a \vee b \vee c) \wedge (a \vee \neg b)$ satisfiable?

### SMT

- SAT + modulo theories
- in the best case: NP-complete
- in the worst case: undecidable

### Modulo theories

- theory of bit vectors
- theory of arrays

$\Rightarrow$ efficient solvers through conflict-driven clause learning

# Conflict-driven clause learning (CDCL)

## Algorithm (simplified)

**Conflict-driven clause learning**

1. choose random assignment
2. unit propagation
3. conflict analysis
4. backtracking

We skip implication graphs and backtracking.

# Conflict-driven clause learning (CDCL)

## Choose random assignment

$$g := (a \vee \neg c) \wedge (a \vee b \vee c) \wedge (a \vee \neg b)$$

- randomly choose $a = 0$

$\Rightarrow$ $(0 \vee \neg c) \wedge (0 \vee b \vee c) \wedge (0 \vee \neg b)$

# Conflict-driven clause learning (CDCL)

## Unit propagation

$$(0 \vee \neg c) \wedge (0 \vee b \vee c) \wedge (0 \vee \neg b)$$

- $c = 0$
- $b = 1$
- $b = 0$ ↯

$\Rightarrow$ $a \vee \neg b$ cannot be satisfied

$\Rightarrow$ $g$ cannot be satisfied

# Conflict-driven clause learning (CDCL)

## Conflict analysis

$$(a = 0, b = 1) \Rightarrow \text{conflict}$$

- $X \Rightarrow Y \Leftrightarrow \neg Y \Rightarrow \neg X$ (contraposition)

$\Rightarrow$ $\neg\text{conflict} \Rightarrow (a = 1, b = 0)$

$\Rightarrow$ $\neg(a \wedge \neg b) \Leftrightarrow \neg a \vee b$

$\Rightarrow$ $cl := \neg a \vee b$ (conflict clause)

# Conflict-driven clause learning (CDCL)

## Next iteration (after backtracking)

$$g' := g \wedge cl = (a \vee \neg c) \wedge (a \vee b \vee c) \wedge (a \vee \neg b) \wedge (\neg a \vee b)$$

- randomly choose $a = 1$
- ...
- randomly choose $b = 1$
- ...
- randomly choose $c = 0$
- ...
- SAT

# SAT + theory solver

## Interaction

$g := t_1 \wedge t_2 \wedge (t_3 \vee t_4)$

- $t_1 : a < b$
- $t_2 : a + b == 100$
- $t_3 : b > 50$
- $t_4 : a == 99$

- SAT solver randomly sets $t_4 = 1$
- queries theory solver with $(t_1, t_2, t_4)$

# SAT + theory solver

## Theory solver

- $t_4 : a = 99$
- $t_2 : a + b = 100 \Leftrightarrow b = 1$
- $t_1 : (a < b) \Leftrightarrow 99 < 1$ ↯
- UNSAT
- $cl := t_1 \lor t_2 \lor t_4$ (conflict clause)

# SAT + theory

## Final moves

$$g' := g \wedge cl = t_1 \wedge t_2 \wedge (t_3 \vee t_4) \wedge (t_1 \vee t_2 \vee t_4)$$

- SAT solver: $t_1 = 1, t_2 = 1, t_3 = 1, t_4 = 0$
- theory solver
  - $t_1 : a < b$
  - $t_2 : a + b == 100$
  - $t_3 : b > 50$

  $\Rightarrow$ SAT for $a = 1, b = 99$

$\Rightarrow$ SAT

# Satisfiability modulo theories (SMT)

## What are SMT solvers?

### SAT

Is $(a \vee \neg c) \wedge (a \vee b \vee c) \wedge (a \vee \neg b)$ satisfiable?

### SMT

- SAT + modulo theories
- in the best case: NP-complete
- in the worst case: undecidable

### Modulo theories

- theory of bit vectors
- theory of arrays

Slide is duplicated from before

# Satisfiability modulo theories (SMT)

## Theory of bit vectors

### Bit vector

A bit vector $b$ is a vector of bits with a given length $l$:

$$b : \{0, \dots, l-1\} \rightarrow \{0, 1\}.$$

- $b \mod 2^l, b \in BV$
- arithmetic operations $(+, -, *, /, \dots)$
- bitwise operations $(\wedge, \vee, \oplus, \ll, \dots)$
- $\mathtt{eax} = (\mathtt{eax} + \mathtt{ebx}) \ll 1$

# Satisfiability modulo theories (SMT)

## Theory of arrays

### Operations

- read: ARRAY $\times$ INDEX $\rightarrow$ ELEMENT
- write: ARRAY $\times$ INDEX $\times$ ELEMENT $\rightarrow$ ARRAY

- `mov eax, [ebp]`
  - $\mathtt{eax} = \text{read}(M, \mathtt{ebp})$
- `mov [ebp], eax`
  - $M' = \text{write}(M, \mathtt{ebp}, \mathtt{eax})$

# Applications

## Complex constraints

```
bool check(uint64_t key)
{
    return key * key * key * key * key * key * key == 0x90de757572b51cd3;
}
```

- Does a solution exist? yes
- What is a solution? *0xe80e9aac619831fb*
- How many solutions do exist? 1

# Complex constraints

**DEMO**

# Model counting

## How many solutions do exist?

### Naive approach

1. $counter := 0$
2. WHILE SMT($\varphi$) $\in$ SAT:
   1. generate conjunction $c$ from model assignment
   2. $\varphi := \varphi \land \neg c$
   3. $counter := counter + 1$

$$(k \cdot k \cdot k \cdot k \cdot k \cdot k \cdot k == 0x90de757572b51cd3) \land (k \neq 0xe80e9aac619831fb)$$

- might not terminate
- does not work for every theory
- independent research branch

# Applications

## Semantic equivalence

$$f(x, y) := (x \oplus y) + 2 \cdot (x \wedge y)$$

We observe

- $f(1, 1) = 2$
- $f(2, 3) = 5$
- $f(10, 20) = 30$

We ask ourselves if

$$x + y \stackrel{?}{=} (x \oplus y) + 2 \cdot (x \wedge y)$$

# Semantic equivalence

$$\varphi \stackrel{?}{=} \psi$$

- SMT($\varphi == \psi$) $\in$ SAT
  - $\Rightarrow$ *single* instance that satisfies the constraints
    - not what we are looking for
- SMT($\varphi \neq \psi$) $\in$ UNSAT
  - $\Rightarrow$ no instance that satisfies the constraints
  - $\Rightarrow$ we *proved* that $\varphi$ and $\psi$ are semantically equivalent

# Semantic equivalence

**DEMO**

# Applications

## Symbolic execution

$$\texttt{add eax, eax} \Rightarrow \texttt{eax} := \texttt{eax} + \texttt{eax}$$

- perform symbolic computations on basic blocks

$\Rightarrow$ automated derivation of constraints

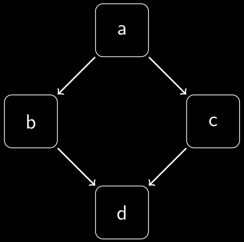
- query SMT solver to prove characteristics of constraints

# Symbolic execution

**DEMO**

# Advanced applications

## Graph search




- $t_1 := b \Rightarrow d$
- $t_2 := c \Rightarrow d$
- $t_3 := a \Rightarrow (b \wedge \neg c) \vee (\neg b \wedge c)$
- $\varphi := t_1 \wedge t_2 \wedge t_3$

# Advanced applications

## Exploit generation

```
int vuln(char input[])
{

    char output[15];
    int pass = 0;


    strcpy(output, input);

    if (pass)
        return 1;

    return 0;
}
```

- stack variable pass is set to 0
- vuln returns 1 if pass $\neq 0$
- buffer overflow at strcpy overwrites pass

# Bounded model checking

## Overview

$$\varphi := \mathit{preconditions} \land \mathit{prog} \land \neg \mathit{postconditions}$$

- *preconditions*: initial program state
- *prog*: $k$ times unwound control-flow graph
- *postconditions*: memory layout for exploitation
- SMT($\varphi$) $\in$ *UNSAT*: no bug in the bounded program execution
- SMT($\varphi$) $\in$ *SAT*: bug in the bounded program execution

# Bounded model checking

## Workflow

1. create memory dump
2. translate assembly code into intermediate representation
3. inline functions
4. unroll loops
5. apply static single assignment (SSA)
6. apply preconditions and postconditions
7. generate SMT formula

# Bounded model checking

**DEMO**

# Advanced applications

## Breaking weak cryptography

- *Petya* ransomware
- modified salsa20 cipher
  - 10 instead of 20 rounds
  - operates on 16-bit instead of 32-bit words
- broken by genetic algorithm in 10 to 30 seconds
- SMT solver break it in less than 1 second

# Further applications

- deobfuscation
- ROP gadget chaining (compiler)
- shellcode construction
- program synthesis
- . . .

# General notes

- SMT solvers are very efficient for real-world problems
- different SMT solvers for different use cases
  - *boolector* for arrays and bit vectors
  - *z3* has a powerful API and supports many theories
- generic SMT interface defined by SMT-LIB standard

# Limitations

- buggy in some edge cases

  $\Rightarrow$ try out different SMT solvers

- in general, problems are at least NP-complete
- confusion and diffusion

  $\Rightarrow$ they cannot break strong cryptography

# Conclusion

- SAT solvers
- conflict-driven clause learning
- SAT + theory interaction
- theory of bit vectors and arrays
- solving complex constraints
- model counting
- proving semantic equivalence
- symbolic execution
- graph search
- bounded model checking
- breaking weak cryptography

# References I


- Research Group Verification meets Algorithm Engineering. *LLBMC – The Low-Level Bounded Model Checker*. URL: http://llbmc.org.
- Clark Barrett, Pascal Fontaine and Cesare Tinelli. *The SMT-LIB Standard: Version 2.5*. Tech. rep. Department of Computer Science, The University of Iowa, 2015. URL: http://smtlib.cs.uiowa.edu.
- Tim Blazytko. *Static data flow analysis and constraint solving to craft inputs for binary programs*. 2015. URL: https://archive.org/details/static_data_flow_analysis_and_constraint_solving_to_craft_inputs_for_binary_programs.
- A.R. Bradley and Z. Manna. *The Calculus of Computation: Decision Procedures with Applications to Verification*. Springer Berlin Heidelberg, 2007. ISBN: 9783540741138.

# References II


- D. Kroening, R.E. Bryant and O. Strichman. *Decision Procedures: An Algorithmic Point of View*. Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, 2008. ISBN: 9783540741046.
- Aina Niemetz, Mathias Preiner and Armin Biere. *Boolector*. URL: http://fmv.jku.at/boolector/.
- Microsoft Research. *The Z3 Theorem Prover*. URL: https://github.com/Z3Prover/z3.
- Rolf Rolles. *The Case for Semantics-Based Methods in Reverse Engineering*. 2012. URL: http://www.msreverseengineering.com/blog/2014/6/23/recon-2012-keynote-the-case-for-semantics-based-methods-in-reverse-engineering.

# References III


- Edward J Schwartz, Thanassis Avgerinos and David Brumley. 'All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)'. In: *IEEE Symposium on Security and Privacy 2010*. IEEE. 2010, pp. 317–331.
- CEA IT Security. *Miasm2*. URL: https://github.com/cea-sec/miasm.
- leo stone. *hack-petya*. URL: https://github.com/leo-stone/hack-petya.
- Julien Vanegue, Sean Heelan and Rolf Rolles. 'SMT Solvers in Software Security.' In: *WOOT*. 2012, pp. 85–96.