Z3 – SATISFIABILITY MODULO THEORY SOLVER

WHY Z3

- Solving equation is widely used in program analysis
  - Program Testing
  - Program Verification
  - …
- In CTF games,
  - Automatic solve weak encryption and hash
  - Find the data leaded to certain program address

SATISFIABILITY

- Can the formula be solved?

Solution/Model

$x^2 + y^2 < 1 \text{ and } xy > 0.1$ ➡ **sat,** $x = 1/8 , y = 7/8$

$x^2 + y^2 < 1 \text{ and } xy > 1$ ➡ **unsat, Proof**

Leonardo de Moura, et al. "Advances in Automated Theorem Proving", http://www.ugrad.cs.ubc.ca/~cs418/2012-1/lecture/AdvancesInATPshort.pdf

Z3

- Z3, Automated Theorem Provider

DPLL

Simplex

Rewriting

Superposition

Z3 is a collection of

Symbolic Reasoning Engines

Congruence Closure

Groebner Basis

∀∃ elimination

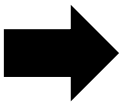
Euclidean Solver

Leonardo de Moura, et al. "Advances in Automated Theorem Proving", http://www.ugrad.cs.ubc.ca/~cs418/2012-1/lecture/AdvancesInATPshort.pdf

PLAY WITH Z3

- Visit Z3 tutorial website rise4fun
  - http://rise4fun.com/z3/tutorial

```
( set-logic QF_LIA )
( declare-fun x () Int )
( declare-fun y () Int )
( declare-fun z () Int )
( assert
   ( or
        ( <= (+ x 3) (* 2 y ) )
        ( >= (+ x 4) z )
) )
( check-sat )
( get-model )
```



```
sat
(model
     (define-fun y () Int 0)
     (define-fun x () Int (- 3))
     (define-fun z () Int 2)
)
```

SMT-LIB FORMAT

- The format is defined in SMT-Lib
- Hard to write in manual
  - So we need some language binding, such as c, python

```
( set-logic QF_LIA )
( declare-fun x () Int )
( declare-fun y () Int )
( declare-fun z () Int )
( assert
   ( and
      ( or
         ( <= (+ x 3) (* 2 y ) )
         ( >= (+ x 4) z )
) ) )
( check-sat )
( get-model )
```

Z3PY

- Z3py is the z3 binding for python
  - https://github.com/Z3Prover/z3

```
from z3 import *
a, b = BitVecs('a b', 32)
s = Solver()
s.add((a + b) == 1337)
if s.check() == sat:
    print s.model()
else: print 'Unsat'
```

```
In [14]: from z3 import *

In [15]: a, b = BitVecs('a b', 32)

In [16]: s = Solver()

In [17]: s.add((a + b) == 1337)

In [18]: s.check()
Out[18]: sat

In [19]: s.model()
Out[19]: [b = 0, a = 1337]

In [20]: s.add(a != 0)

In [21]: s.check()
Out[21]: sat

In [22]: s.model()
Out[22]: [b = 0, a = 1337]

In [23]: s.add(b != 0)

In [24]: s.check()
Out[24]: sat

In [25]: s.model()
Out[25]: [b = 2147483648, a = 2147484985]
```

Axel '@0vercl0k' Souchet, "Theorem prover, symbolic execution and practical reverse-engineering," https://doar-e.github.io/presentations/securityday2015/SecDay-Lille-2015-Axel-0vercl0k-Souchet.html#/4/3

Z3 TYPES

- Z3 supports types
  - BitVecs is arrays of 0 & 1's
    - With 32 bits, as C int type
  - Ints
    - Integer in math, infinite and never overflow
    - Different of our machine
  - Bools
  - Arrays

```
from z3 import *
a, b = BitVecs('a b', 32)
s = Solver()
s.add((a + b) == 1337)
if s.check() == sat:
    print s.model()
else: print 'Unsat'
```

Z3 OPERATOR

- Z3 data type has no singed information inside
- The signed information held by z3 operator

- Signed operators
  - <, <=, =, >, >=, /, % , >>
- Unsigned operators
  - ULT, ULE, UGT, UGE, Udiv, Uremand, LShR

```
from z3 import *
a, b = BitVecs('a b', 32)
s = Solver()
s.add((a + b) == 1337)
if s.check() == sat:
    print s.model()
else: print 'Unsat'
```

Z3 SOLVER

- The solver will try to find an answer satisfying your formula
- Return the concrete value for your symbolic variables
  - We call it a model
  - Z3 return one model even there are multiple solutions

```
In [101]: from z3 import *

In [102]: a, b = BitVecs('a b', 32)

In [103]: solve(a+b == 1337)
[b = 0, a = 1337]

In [104]: s = Solver()

In [105]: s.add(a+b == 1337)

In [106]: s.check()
Out[106]: sat

In [107]: m = s.model()

In [108]: m[a], m[b]
Out[108]: (1337, 0)
```

ARRAY

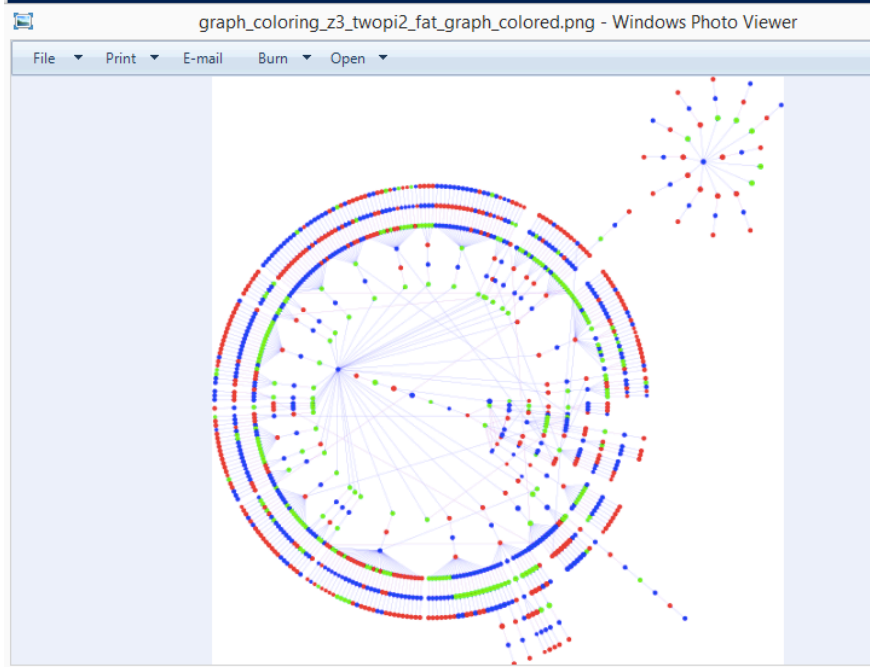
- Useful to simulate memory state
  - Write via Store()
  - Read via [] and Select()

```
In [85]: from z3 import *
In [86]: tab = Array('tab', BitVecSort(8), BitVecSort(32))
In [87]: tab = Store(tab, BitVecVal(0, 8), BitVecVal(1, 32))
In [88]: tab = Store(tab, BitVecVal(1, 8), BitVecVal(1, 32))
In [89]: tab = Store(tab, BitVecVal(2, 8), BitVecVal(0, 32))
In [90]: tab = Store(tab, BitVecVal(3, 8), BitVecVal(1337, 32))
In [91]: idx = BitVec('idx', 8)
In [92]: solve(tab[idx] == 1337)
[idx = 3]
```

Z3PY TOYS: GRAPH COLORING

- Graph coloring problem

```
D:\Codes\z3-playground>"c:\Program Files (x86)\Python\Python275\python.exe"  graph_coloring_z3.py
Building the graph..
Trying to color 'petersnon_3_coloring_graph' now (10 nodes, 17 edges)..
OK, found a solution with 3 colors
Here is the solution (in 0s):
{u'1': 2L, u'0': 1L, u'3': 0L, u'2': 2L, u'5': 0L, u'4': 2L, u'7': 1L, u'6': 0L, u'9': 2L, u'8': 1L}
Setting the colors on the nodes..
Saving it in the current directory with the layout 'circo'..
['C:\\Program Files (x86)\\Graphviz2.38\\bin\\circo.exe', '-T', 'dot']
['C:\\Program Files (x86)\\Graphviz2.38\\bin\\neato.exe', '-n2', '-Tpng']
---
Trying to color 'twopi2_fat_graph' now (1090 nodes, 1118 edges)..
OK, found a solution with 3 colors
Here is the solution (in 1s):
Too long, see the .png!
Setting the colors on the nodes..
Saving it in the current directory with the layout 'twopi'..
['C:\\Program Files (x86)\\Graphviz2.38\\bin\\twopi.exe', '-T', 'dot']
['C:\\Program Files (x86)\\Graphviz2.38\\bin\\neato.exe', '-n2', '-Tpng']
---
```



PRACTICE: HARDER SERIAL

- PicoCTF 2013
- Break serial number
  - https://gist.github.com/ekse/baee0cabbe12861443a5#file-harder_serial-py
  - harder_serial.py
- Write z3 script tp solve it!!

PRACTICE: CSAW 300-CRACKME

- nc 140.113.194.85 49204
- Binary: Problem/CSAW2013_Crackme300
- How to:
  - Reverse Binary
  - Write z3 constraints
  - Solve It!

Q&A