

++

Intro to Binary Analysis with Z3 and Angr

Sam Brown

hack.lu 2018



++

whoami

- + Sam Brown – @_samdb_
- + Consultant, Research team @ MWR (F-Secure?)
- + Worky worky – Secure Dev, Code Review, Product Teardowns
- + Research/home time – poking at Windows internals, browser security, playing with Angr and Z3

++
VM

- + Grab a USB
- + Super secure – user/user and root/root
- + Command prompt -> `workon angr`
- + Exercises: `/home/user/smt-workshop`

++
Schedule

13:30 – 15:15

Background & Z3

15:15 – 15:30

Break/Extra exercise time

15:30 – End

Angr

Outline

1. what the Hell is z3?
2. z3-python
3. Lab - Cheating at Logic Challenges
4. Lab - Encoding CPU Instructions
5. z3 in the Real world
6. Angr!
7. Lab - Using Angr in Anger
8. Wrap-up

++

What the Hell is Z3?

- + Z3 is an SMT solver
- + what

++

What the Hell is a SMT Solver?

- + Satisfiability Modulo Theories (SMT) solvers
- + what

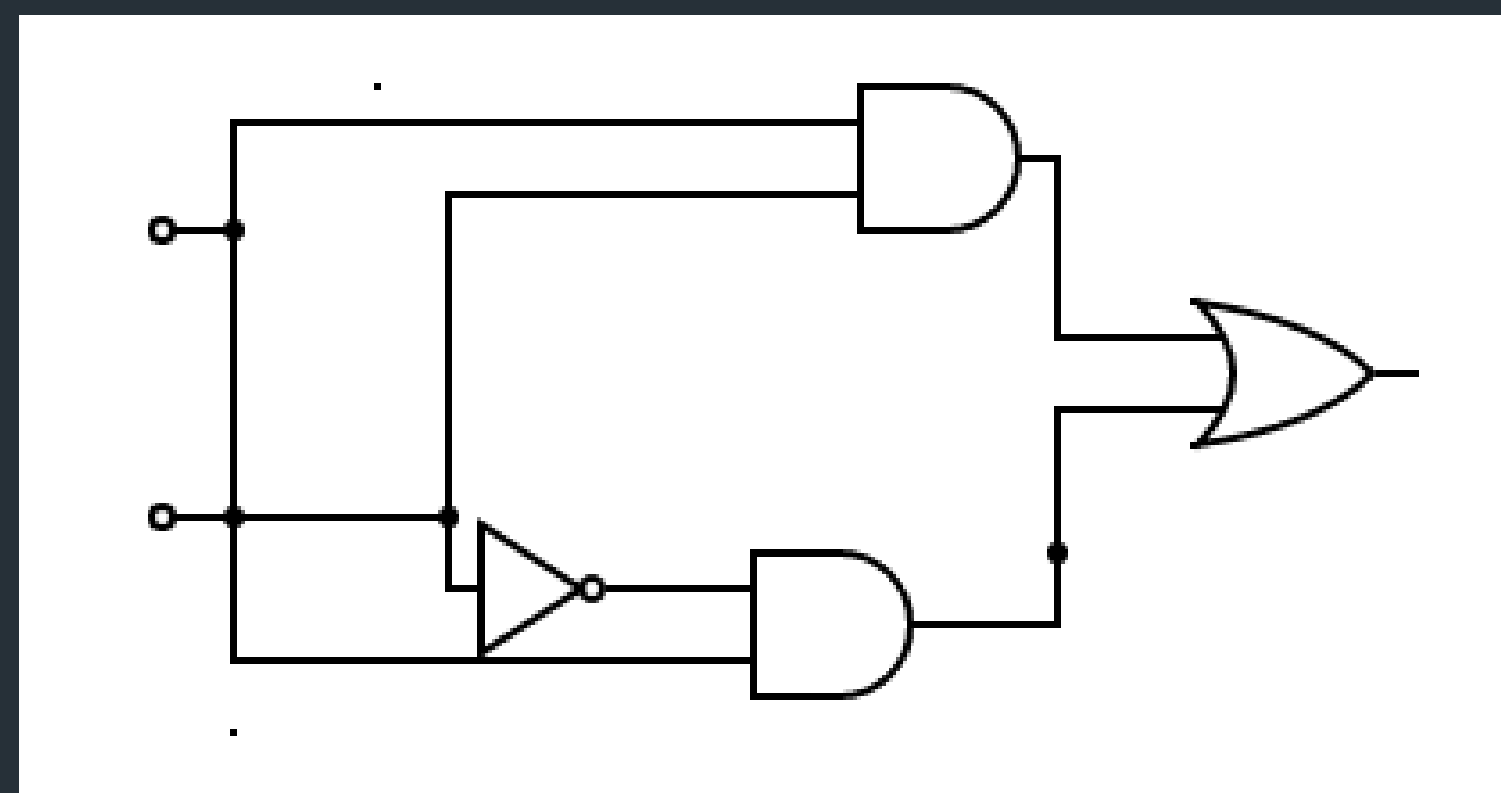
++

What the Hell is a SMT Solver?

- + Built on top of SAT solvers..
- + ‘satisfiability’
- + Boolean formula in \Rightarrow Can it be satisfied?
- + If so give me example values

++

What the Hell is a SAT Solver?



?

Input:

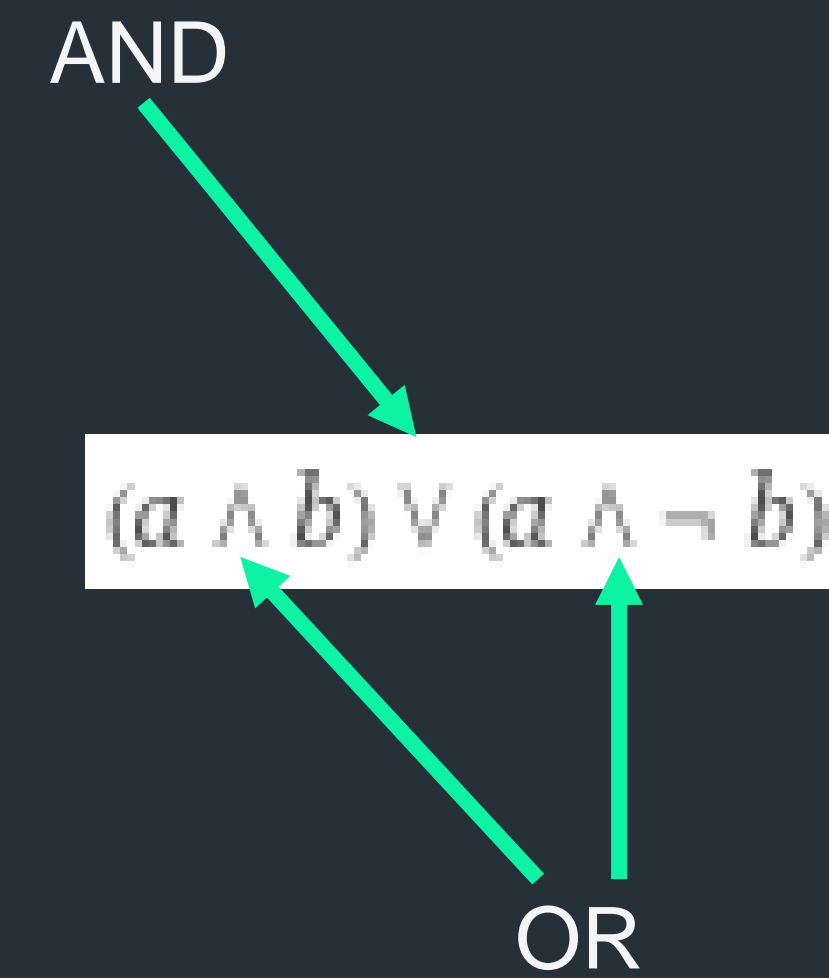
$$(a \wedge b) \vee (a \wedge \neg b)$$

$$(a \text{ AND } b) \text{ OR } (a \text{ AND } (\text{NOT } b))$$

++

CNF?

- + Conjunctive Normal Form
- + an AND of ORs
- + AND, OR, NOT only
- + Used because any propositional logic can be converted to CNF in linear time



++

What the Hell is a SAT Solver?

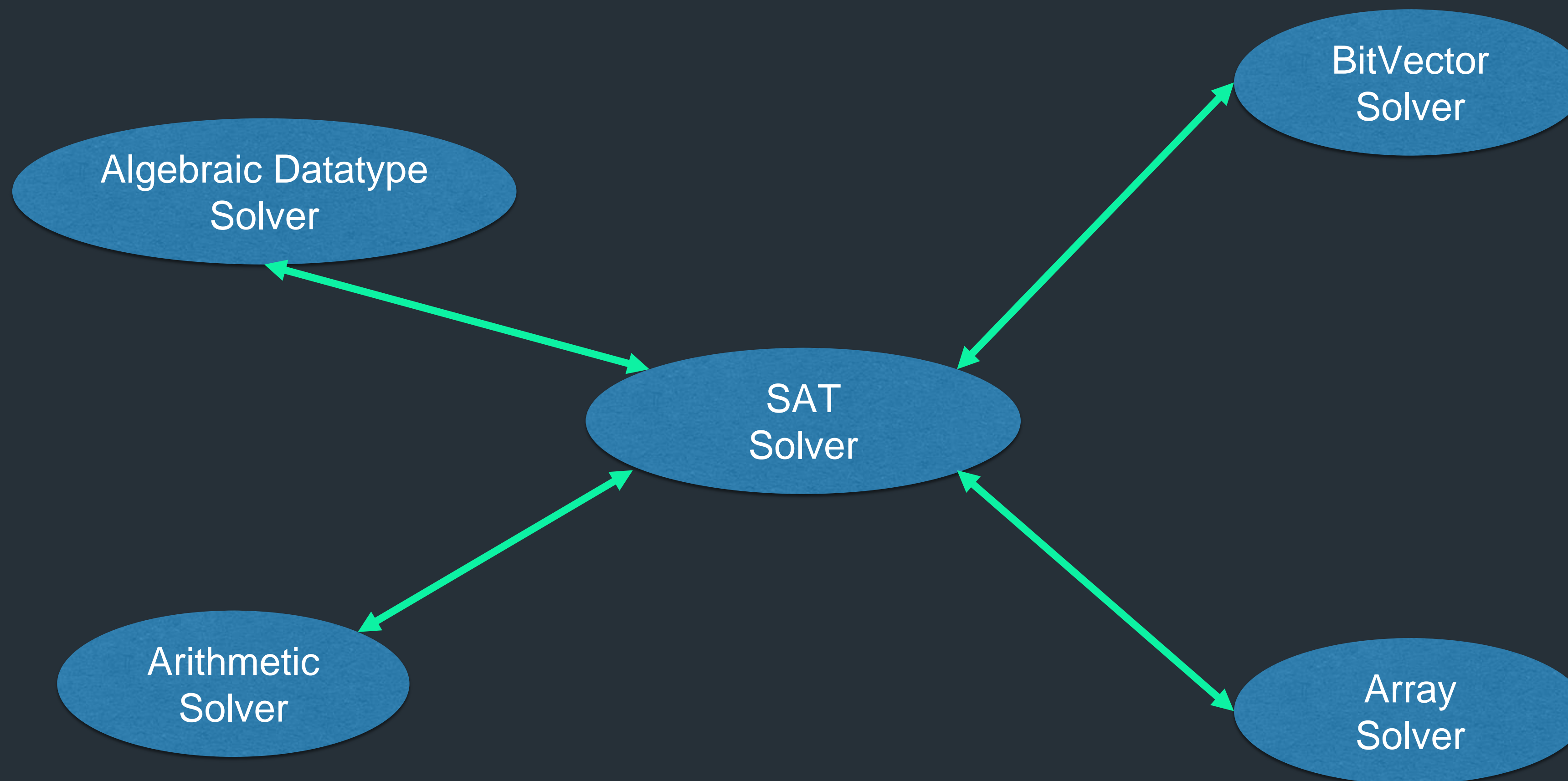
Truth table:

a	b	$(a \wedge b) \vee (a \wedge \neg b)$
T	T	T
T	F	T
F	T	F
F	F	F

- ++
- ## What the Hell is a SMT Solver?
- + SMT builds on this
 - + Converts constraints on integers, vectors, strings etc into forms SAT solvers can work with
 - + Referred to as ‘theories’

++

What the Hell is a SMT Solver?



++
More

A Peek Inside SAT Solvers – Jon Smock

<https://www.youtube.com/watch?v=d76e4hV1ijY>

Outline

1. what the Hell is a SMT solver?
2. Z3-python
3. Lab - Cheating at Logic Challenges
4. Lab - Encoding CPU Instructions
5. Z3 in the Real world
6. Angr!
7. Lab - Using Angr in Anger
8. Wrap-up

++
Z3

- + Theorem prover from Microsoft Research
- + High performance
- + Well engineered
- + Open source: <https://github.com/Z3Prover/z3>

++

Z3

```
1 (declare-const a Int)
2 (declare-fun f (Int Bool) Int)
3 (assert (> a 10))
4 (assert (< (f a true) 100))
5 (check-sat)
```

Declare an integer constant 'a'

Declare a function 'f' 'int f(int, bool);'

'a' > 10

f(a, true) -> ret < 100

Is this possible?

++

Z3

```
1 (declare-const a Int)
2 (declare-fun f (Int Bool) Int)
3 (assert (> a 10))
4 (assert (< (f a true) 100))
5 (check-sat)
6 (get-model)
```

If so, what might everything look like?



++

Z3

```
sat
(model
  (define-fun a () Int
    11)
  (define-fun f ((x!1 Int) (x!2 Bool)) Int
    (ite (and (= x!1 11) (= x!2 true)) 0
          0))
)
```

A = 11

Function Definition

Ite = if then else

else

If arg1 == 11 and arg2 == true

++
Z3

Theories:

+ Functions

+ Arithmetic

+ Bit-Vectors

+ Algebraic Data Types

+ Arrays

+ Polynomial Arithmetic

Lists, trees,
enums, etc

Exponentials,
sine, cosine,
etc


++
Z3

In browser tutorial: <https://rise4fun.com/z3/tutorial>

++
Z3-python
+ <https://github.com/Z3Prover/z3/tree/master/src/api/python>

+ `pip install z3`

+ Secretly a DSL

 `Z3_var_one and Z3_var_two !=
And(Z3_var_one, Z3_var_two)`

++

Z3-python

```
1 file changed, 12 insertions(+), 12 deletions(-)
(angr) user@workshop:~/smt-workshop$ python
Python 2.7.13 (default, Nov 24 2017, 17:33:09)
[GCC 6.3.0 20170516] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from z3 import *
>>> a = Int('x')
>>> b = Int('y')
>>> s = Solver()
>>> s.add(a * 15 == b)
>>> s.check()
sat
>>> s.model()
[x = 0, y = 0]
>>> s.add(a != 0)
>>> s.check()
sat
>>> s.model()
[y = 15, x = 1]
>>> █
```

++
Z3-python

- + `And(condition_one, condition_two)`
- + `Or(condition_one, condition_two)`
- + `Not(boolean_expression)`

++
Z3-python

- + `Distinct($list)` – set constraint all items in list must be unique
- + `BitVec('name', size)` – create a bit vector of size bits
- + `Bool('name'), Real('name')` – Boolean and real symbolic variables
- + `If(condition, true_result, false_result)` – decision logic in Z3!

++

Z3-python

```
Color = Datatype('Color')
Color.declare('red')
Color.declare('green')
Color.declare('blue')
Color = Color.create()
```

```
Color, (red, green, blue) = EnumSort('Color', ('red', 'green', 'blue'))
```

++
Z3-python

Good tutorials:

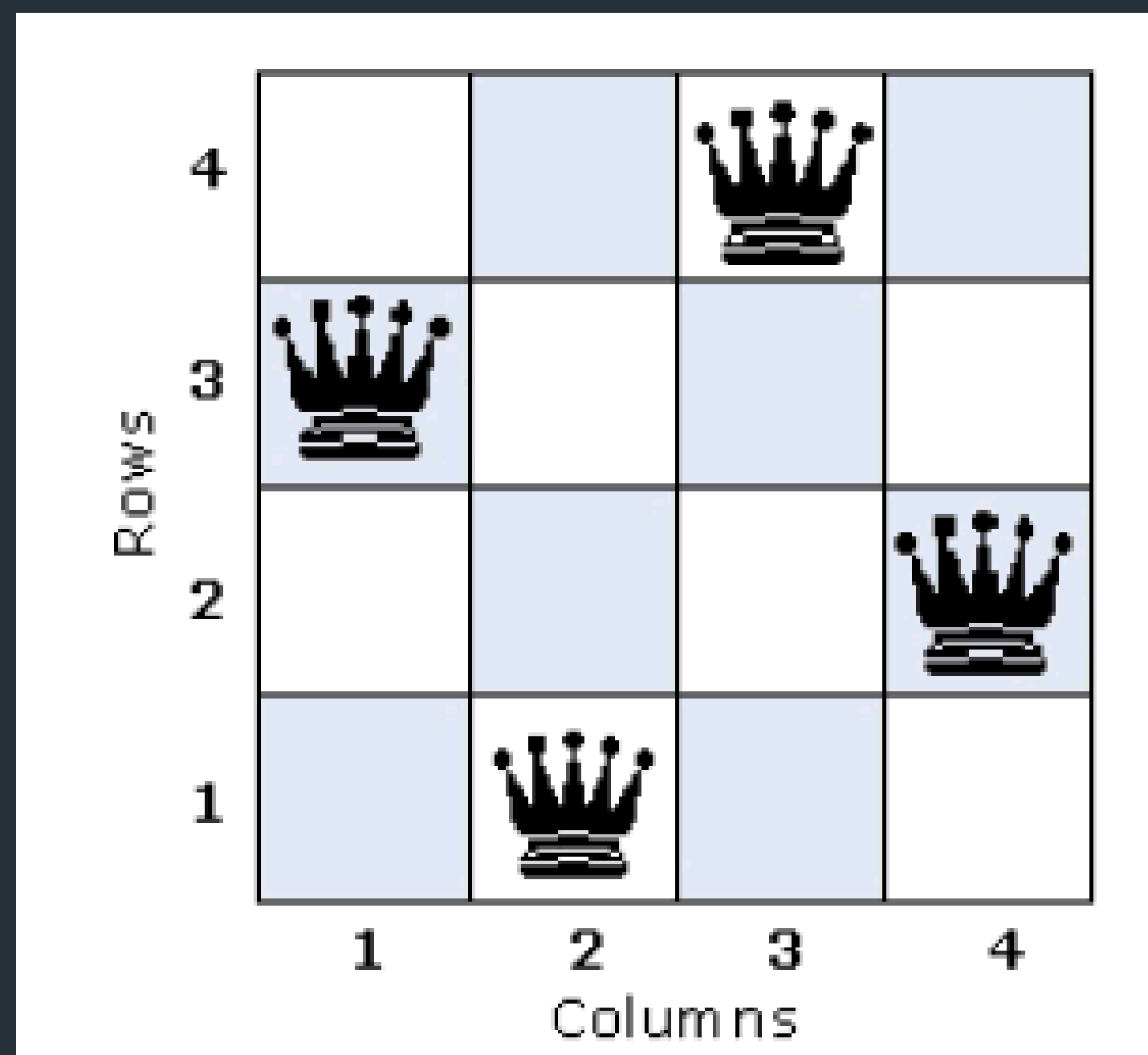
- + <http://ericpony.github.io/z3py-tutorial/guide-examples.htm>
- + <http://ericpony.github.io/z3py-tutorial/advanced-examples.htm>

Outline

1. what the Hell is a SMT solver?
2. Z3-python
3. Lab - Cheating at Logic Challenges
4. Lab - Encoding CPU Instructions
5. Z3 in the Real world
6. Angr!
7. Lab - Using Angr in Anger
8. Wrap-up

++ Cheating at Logic Challenges – N-Queens

How can N queens be placed on an NxN chessboard so that no two of them attack each other? `~/smt-workshop/z3/n_queens`



++

Cheating at Logic Challenges – N-Queens

How can N queens be placed on an NxN chessboard so that no two of them attack each other?

```
columns = [Int('col_%d' % i) for i in range(n)]  
rows = [Int('row_%d' % i) for i in range(n)]  
s = Solver()
```

++

Cheating at Logic Challenges – N-Queens

How can N queens be placed on an NxN chessboard so that no two of them attack each other?

```
for i in range(n):
    s.add(columns[i] >= 0, columns[i] < n, rows[i] >= 0, rows[i] < n)
s.add(Distinct(rows))
s.add(Distinct(columns))
for i in range(n - 1):
    for j in range(i + 1, n):
        s.add(abs(columns[i] - columns[j]) != abs(rows[i] - rows[j]))
```

++

Cheating at Logic Challenges – N-Queens

How can N queens be placed on an NxN chessboard so that no two of them attack each other?

```
if s.check() == sat:  
    m = s.model()
```


++

Cheating at Logic Challenges – N-Queens

How can N queens be placed on an NxN chessboard so that no two of them attack each other?

```
(angr) user@workshop:~/smt-workshop/z3/n_queens$ python solution.py 4
Solving N Queens for a 4 by 4 chess board
  Q   .   .   .
.   .   .   Q
Q   .   .   .
.   .   Q   .
(angr) user@workshop:~/smt-workshop/z3/n_queens$
```

++
Cheating at Logic Challenges – Hackvent 15

We've captured a strange message. It looks like it is encrypted somehow ...
 iw, hu, fv, lu, dv, cy, og, lc, gy, fq, od, lo, fq, is, ig, gu, hs, hi, ds, cy, oo, os,
 iu, fs, gu, lh, dq, lv, gu, iw, hv, gu, di, hs, cy, oc, iw, gc

We've also intercepted what seems to be a hint to the key:

```

bytwycju + yzvyjjdy ^ vugljtyn + ugdztnwv | xbfziozy = bzuwtwol
  ^           ^           ^           ^           ^
wwnnnqbw - uclfqvdu & oncycbxh | oqcnwbsd ^ cgyoyfjg = vyhyjivb
  &           &           &           &           &
yzdgotby | oigsjgoj | ttligxut - dhcqxlfw & szblgodf = sfgsoxdd
  +           +           +           +           +
yjjowdqh & niqztgs + ctvtwysu & diffhlnl - thhwohwn = xsvuojtx
  -           -           -           -           -
nttuhlnq ^ oqbctlzh - nshtztns ^ htwizvwi + udluvhez = syhjizjq
  =           =           =           =           =
fjivucti  zoljwdf1  sugvqgww  uxztiywn  jqxizzxq
  
```

++ Cheating at Logic Challenges – DIY Sudoku

```
~/smt-workshop/z3/sudoku
```

```
$ workon angr
```

```
$ python skeleton.py tests.txt
```

Files:

+ skeleton.py => **Your solution**

+ solution.py => **My solution**

+ tests.txt => Random puzzles from <http://magictour.free.fr/top100>

++ Cheating at Logic Challenges – DIY Sudoku

Steps:

1. `puzzle` is a string with 81 chars, `.` for unknowns, ints for known values
2. Create a 9*9 grid of symbolic variables
3. Add baseline value constraints on every square
4. Add constraints for known int values to hold
5. Add unique constraints on rows/columns/squares

- ++
 - Cheating at Logic Challenges**
 - DIY (If time) Java RNG seed recovery
 - `~/smt-workshop/z3/rng`
 - Recover the seed from Java's default insecure Random Number Generator!
 - See: `README.md`

Outline

1. what the Hell is a SMT solver?
2. Z3-python
3. Lab - Cheating at Logic Challenges
4. Lab - Encoding CPU Instructions
5. Z3 in the Real world
6. Angr!
7. Lab - Using Angr in Anger
8. Wrap-up

- ++
- ## Encoding CPU Instructions
- + Automatically analyse assembly
 - + Transform instructions into constraints on registers, flags
 - + Answer Q's about sequences of instructions

~/smt-workshop/z3/x86

++

Encoding CPU Instructions

registers.py

```
from itertools import *  
  
class Registers:  
  
    def __init__(self):  
        self.eax = BitVec('eax', 32)  
        self.ebx = BitVec('ebx', 32)  
        self.ecx = BitVec('ecx', 32)  
        self.edx = BitVec('edx', 32)  
        self.edi = BitVec('edi', 32)  
        self.esi = BitVec('esi', 32)  
        self.ebp = BitVec('ebp', 32)  
        self.esp = BitVec('esp', 32)  
  
        self.eip = BitVec('eip', 32)
```


++

Encoding CPU Instructions

```
from sys import *  
  
class Registers:  
  
    def __init__(self):  
        self.eax = BitVec('eax', 32)  
        self.ebx = BitVec('ebx', 32)  
        self.ecx = BitVec('ecx', 32)  
        self.edx = BitVec('edx', 32)  
        self.edi = BitVec('edi', 32)  
        self.esi = BitVec('esi', 32)  
        self.ebp = BitVec('ebp', 32)  
        self.esp = BitVec('esp', 32)  
  
        self.eip = BitVec('eip', 32)  
  
        self.cf = Bool('cf')  
        self.zf = Bool('zf')  
        self.sf = Bool('sf')  
        self.of = Bool('of')
```

Carry

Zero

Sign

Overflow

++

Encoding CPU Instructions

Xor

Performs a bitwise exclusive OR (XOR) operation on the destination (first) and source (second) operands and stores the result in the destination operand location. The source operand can be an immediate, a register, or a memory location; the destination operand can be a register or a memory location. (However, two memory operands cannot be used in one instruction.) Each bit of the result is 1 if the corresponding bits of the operands are different; each bit is 0 if the corresponding bits are the same.

The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined.

https://c9x.me/x86/html/file_module_x86_id_330.html

++

Encoding CPU Instructions

Add

Adds the first operand (destination operand) and the second operand (source operand) and stores the result in the destination operand. The destination operand can be a register or a memory location; the source operand can be an immediate, a register, or a memory location. (However, two memory operands cannot be used in one instruction.) When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The ADD instruction performs integer addition. It evaluates the result for both signed and unsigned integer operands and sets the OF and CF flags to indicate a carry (overflow) in the signed or unsigned result, respectively. The SF flag indicates the sign of the signed result.

The OF, SF, ZF, AF, CF, and PF flags are set according to the result.

https://c9x.me/x86/html/file_module_x86_id_5.html

++

Encoding CPU Instructions

Or – DIY / Skeleton: `or.py` My solution: `or_solution.py`

Performs a bitwise inclusive OR operation between the destination (first) and source (second) operands and stores the result in the destination operand location. The source operand can be an immediate, a register, or a memory location; the destination operand can be a register or a memory location. (However, two memory operands cannot be used in one instruction.) Each bit of the result of the OR instruction is set to 0 if both corresponding bits of the first and second operands are 0; otherwise, each bit is set to 1.

The OF and CF flags are cleared; the SF, ZF, and PF flags are set according to the result. The state of the AF flag is undefined.

https://c9x.me/x86/html/file_module_x86_id_219.html

++

Encoding CPU Instructions

sub – DIY / Skeleton: `sub.py`, My solution: `sub_solution.py`

Subtracts the second operand (source operand) from the first operand (destination operand) and stores the result in the destination operand. The destination operand can be a register or a memory location; the source operand can be an immediate, register, or memory location.

(However, two memory operands cannot be used in one instruction.) When an immediate value is used as an operand, it is sign-extended to the length of the destination operand format.

The SUB instruction performs integer subtraction. It evaluates the result for both signed and unsigned integer operands and sets the OF and CF flags to indicate an overflow in the signed or unsigned result, respectively. The SF flag indicates the sign of the signed result.

The OF, SF, ZF, AF, PF, and CF flags are set according to the result.

++

Encoding CPU Instructions

jmp

Fake this – just goes directly to operand address

++

Encoding CPU Instructions

jnz

Instruction	Description	signed-ness	Flags	short jump opcodes	near jump opcodes
JNE JNZ	Jump if not equal Jump if not zero		ZF = 0	75	0F 85

++

Encoding CPU Instructions

`jpg` – DIY, Skeleton: `jpg.py`, My solution: `jpg_solution.py`

Instruction	Description	signed-ness	Flags	short jump opcodes	near jump opcodes
JG JNLE	Jump if greater Jump if not less or equal	signed	ZF = 0 and SF = OF	7F	0F 8F

++

Encoding CPU Instructions – Equivalents

Given two sequences of assembly instructions – do they have the exact same effect?

`~/smt-workshop/z3/equivalence_checking`

<http://zubcic.re/blog/experimenting-with-z3-dead-code-elimination>

++ Encoding CPU Instructions – Opaque Predicates

- + Opaque Predicate: A conditional jump which is always taken or not taken
- + Code Obfuscation
- + Can we auto detect them to remove them?

`~/smt-workshop/z3/opaque_predicates`

<http://zubcic.re/blog/experimenting-with-z3-proving-opaque-predicates>

++

Real World

- + Memory, stack, full flags, oddities make this harder
- + ‘Lift’ instructions to a simpler (to a program!) representation
- + Write constraints for the ‘simpler’ representation

Outline

1. what the Hell is a SMT solver?
2. Z3-python
3. Lab - Cheating at Logic Challenges
4. Lab - Encoding CPU Instructions
5. Z3 in the Real world
6. Angr!
7. Lab - Using Angr in Anger
8. Wrap-up

++

Z3 in the Real World

Bond Allocations

Microsoft Sage/Microsoft Security Risk Detection

Angr

MSR's Project Everest

Goal: verified HTTPS replacement

CLOUDY, WITH A CHANCE OF EXPLOITS —

Microsoft launches “fuzzing-as-a-service” to help developers find security bugs

Project Springfield, Microsoft's "million-dollar bug detector" now available in cloud.

++

Z3 in the Real World

+ HACL* in Mozilla Firefox

<https://www.youtube.com/watch?v=xrZTVRICpSs>

+ AWS Security

<https://aws.amazon.com/blogs/security/protect-sensitive-data-in-the-cloud-with-automated-reasoning-zelkova/>

Outline

1. what the Hell is a SMT solver?
2. Z3-python
3. Lab - Cheating at Logic Challenges
4. Lab - Encoding CPU Instructions
5. Z3 in the Real world
6. Angr!
7. Lab - Using Angr in Anger
8. Wrap-up

++
Angr!



++

Angr!

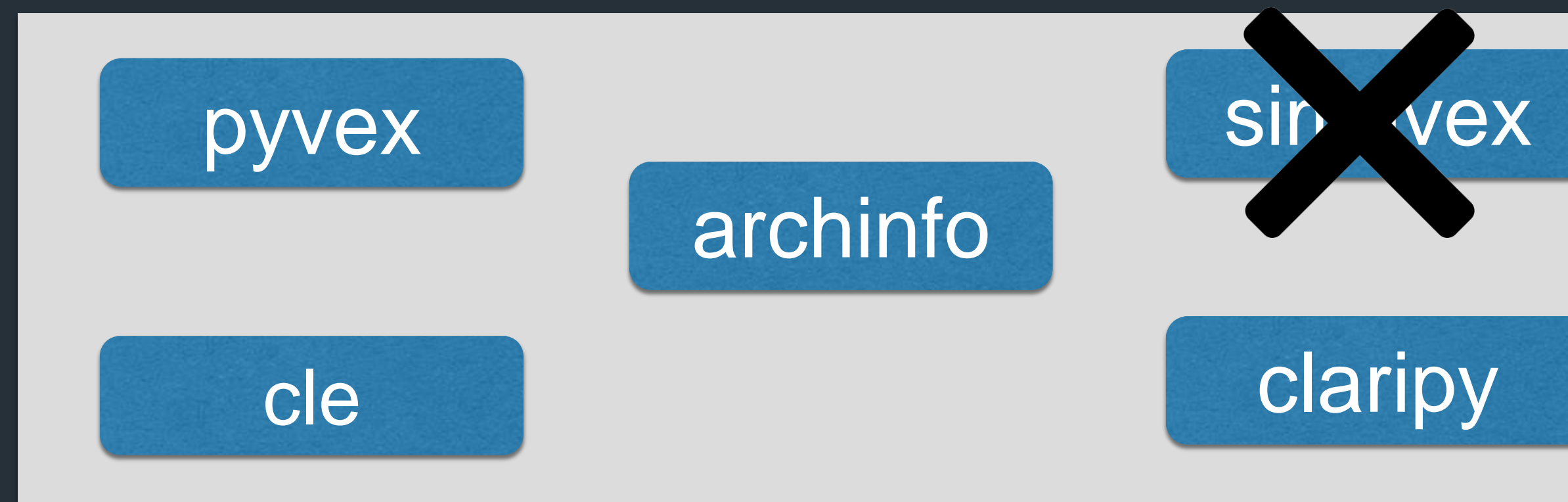
<http://angr.horse>

<https://github.com/angr/angr-doc/blob/master/docs/examples.md>



++
Angr!

Components



Shoulders of Giants...



VEX



Unicorn Engine



Capstone Engine



++

Angr!

Features:

- + Binary Loading
- + Static Analysis
- + Symbolic Execution

++

Binary Loading

- + CLE (CLE Loads Everything)
<https://github.com/angr/cle> (ELF, IdaBin, PE, Mach-O, Blob)
- + Capstone/VEX – x86, mips, arm, ppc
- + Archinfo – <https://github.com/angr/archinfo>

++

VEX

+ IR from Valgrind

The following ARM instruction:

```
subs R2, R2, #8
```

Becomes this VEX IR:

```
1  t0 = GET:I32(16)
2  t1 = 0x8:I32
3  t3 = Sub32(t0,t1)
4  PUT(16) = t3
5  PUT(68) = 0x59FC8:I32
```

<https://docs.angr.io/advanced-topics/ir>

++

Static Analysis

- + Control Flow Graphs
- + Data Flow Graphs
- + Value Set Analysis

‘VSA is a combined numeric-analysis and pointer analysis algorithm that determines a safe approximation of the set of numeric values or addresses that each register and a-loc holds at each program point’

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1.76.637&rep=rep1&type=pdf>

- ++
- ## Symbolic Execution
- + Execute binary using 'symbolic values'
 - + Pass constraints for each path to a constraint solver to get inputs that will reach 'x' point

++

OS Knowledge

advapi32	Add prototypes and MS 64-bit CC (#925)	6 months ago
cgc	Remove IS_SYSCALL and set is_syscall when it actually matters	13 days ago
definitions	Add stub for sigaction	13 days ago
glibc	Remove deprecated names, including .se, god rest their soul	a month ago
libc	Implement syslog simprocedure	13 days ago
linux_kernel	Remove IS_SYSCALL and set is_syscall when it actually matters	13 days ago
linux_loader	Remove deprecated names, including .se, god rest their soul	a month ago
msvcr	Remove deprecated names, including .se, god rest their soul	a month ago
ntdll	Add inhibit_autoret to simprocedures, set it whenever the user perfor...	11 months ago
posix	Clean up after myself, like an adult, I guess	13 days ago
stubs	Remove IS_SYSCALL and set is_syscall when it actually matters	13 days ago
testing	A little better. things import, but SimLibraries aren't hooked up and...	a year ago
tracer	Remove IS_SYSCALL and set is_syscall when it actually matters	13 days ago
uclibc	A little better. things import, but SimLibraries aren't hooked up and...	a year ago
win32	Remove deprecated names, including .se, god rest their soul	a month ago

<https://github.com/angr/angr/tree/master/angr/procedures>

++

OS Knowledge

```
1  import angr
2  from angr.sim_type import SimTypeString
3
4  class strdup(angr.SimProcedure):
5      #pylint:disable=arguments-differ
6
7      def run(self, s):
8          self.argument_types = {0: self.ty_ptr(SimTypeString())}
9          self.return_type = self.ty_ptr(SimTypeString())
10
11         strlen = angr.SIM_PROCEDURES['libc']['strlen']
12         strncpy = angr.SIM_PROCEDURES['libc']['strncpy']
13         malloc = angr.SIM_PROCEDURES['libc']['malloc']
14
15         src_len = self.inline_call(strlen, s).ret_expr
16         new_s = self.inline_call(malloc, src_len+1).ret_expr
17
18         self.inline_call(strncpy, new_s, s, src_len+1, src_len=src_len)
19
20         return new_s
```

++

Angr 8!

New features:

- + All Python 3
- + Byte strings everywhere
- + Simuvex fully integrated
- + Big speed ups in CFG recovery

http://angr.io/blog/moving_to_angr_8/

++ Symbolic Execution

```
unsigned int a = read_int();  
unsigned int b = read_int();  
if( a > 1 ){  
    if( b < 20) {  
        if( a * b > 30){  
            func_one();  
        } else {  
            func_two();  
        }  
    } else {  
        func_three();  
    }  
} else {  
    func_four();  
}
```

a = X
b = Y



++ Symbolic Execution

```
unsigned int a = read_int();  
unsigned int b = read_int();  
if( a > 1 ){  
    if( b < 20) {  
        if( a * b > 30){  
            func_one();  
        } else {  
            func_two();  
        }  
    } else {  
        func_three();  
    }  
} else {  
    func_four();  
}
```

$X > 1$ and $Y < 20$

$X > 1$ and $Y < 20$
and
 $X * Y > 30$

$X > 1$ and $Y < 20$
and
 $X * Y < 30$

++

More!

(State of) The Art of War: Offensive Techniques in Binary Analysis

https://www.cs.ucsb.edu/~vigna/publications/2016_SP_angrSoK.pdf

Outline

1. what the Hell is a SMT solver?
2. Z3-python
3. Lab - Cheating at Logic Challenges
4. Lab - Encoding CPU Instructions
5. Z3 in the Real world
6. Angr!
7. Lab - Using Angr in Anger
8. Wrap-up

++

Using Angr in Anger

Lots of crackme/CTF examples:

<https://github.com/angr/angr-doc/tree/master/examples>

++

Using Angr in Anger

Opaque predicates, easy mode

A nice work on malware deobfuscation, in which Capstone disassembler + Z3 are used to remove opaque predicates:
[zubcic.re/blog/experimen ...](http://zubcic.re/blog/experimen...)


6:45 AM - 19 Aug 2016

31 Retweets 44 Likes



1 31 44

Triton DBA framework @qb_triton · 20 Aug 2016
Replying to @capstone_engine
fun blog post. Here our solution in Python (using Capstone + Z3) of his examples =>



Jonathan Salwan/Triton
Triton is a Dynamic Binary Analysis (DBA) framework. It provides internal components like a Dynamic Symbolic Execution (DSE) engine, a Taint Engine, AST represen...
github.com

https://twitter.com/capstone_engine/status/766632168260972547

++

Opaque Predicates

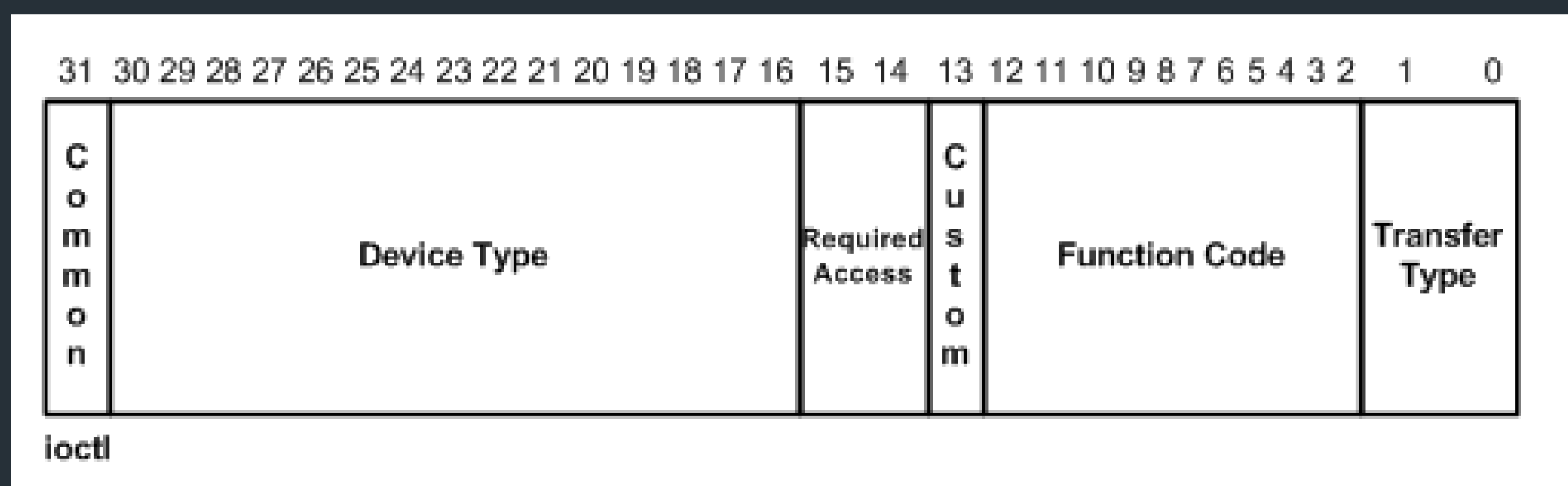
`~/smt-workshop/angr/opaque_predicates`

++

Dumping IOCTL Codes

‘I/O control codes (IOCTLs) are used for communication between user-mode applications and drivers, or for communication internally among drivers in a stack.’

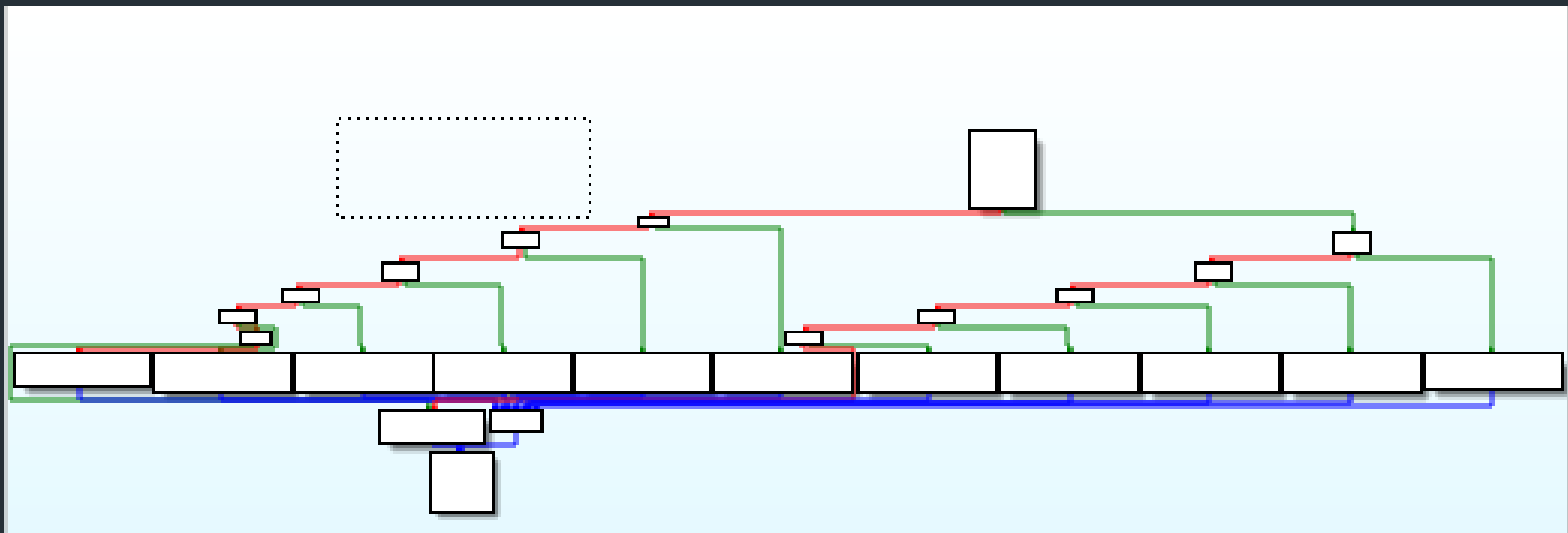
<https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/defining-i-o-control-codes>



++

Dumping IOCTL Codes

- + Consistent structure and values
- + Complex dispatch functions



++

Dumping IOCTL Codes

- + Step through instruction by instruction
- + Save all evaluable register values
- + Process them to find potential IOCTL codes
- + Device code must match
- + Function codes should be in a set range

++

Dumping IOCTL Codes

```
~/smt-workshop/angr/ioctls
```

++

Using Angr in Anger

Used in

https://github.com/mwrlabs/win_driver_plugin

++

Hello World!

```
workon angr
```

```
~/smt-workshop/angr/hello_world
```

```
objdump -d serial.o > disas.txt
```

```
generate_serial_skeleton.py  Makefile  serial.c  serial.o  
generate_serial_solution.py  README.md  serial.h
```

++

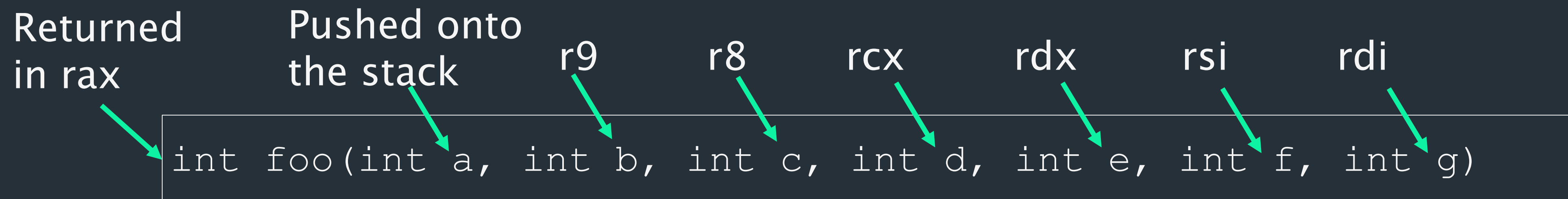
Hello World!

- + Library validates serial codes
- + Several routines with harsher constraints
- + Let's walk through the examples!

++

Calling Conventions - AMD64

- + Arguments are passed in Right-to-Left
- + Return values are returned in `rax`
- + First six arguments passed in registers: `rdi`, `rsi`, `rdx`, `rcx`, `r8`, `r9`
- + Any other arguments passed on the stack



++

Passing arguments

```
arg = state.solver.BVS('serial', 8 * 128)
# Place the symbolic variable at a specific address
rand_addr = 0x0000000041414141
state.memory.store(rand_addr, arg)
# And then make rdi hold a pointer to it as the first argument
state.add_constraints(state.regs.rdi == rand_addr)
```

++

Evaluating symbolic variables

```
sm = p.factory.simulation_manager(state)
sm.explore(find=base + 0x870, avoid=base + 0xaf5)
found = sm.found[0]
answer = found.solver.eval(arg, cast_to=str)
```

++

Bomb lab! DIY!

```
workon angr
```

```
~/smt-workshop/angr/bomb_lab
```

```
ls  
bomb  bomb.c  bomb.h  Makefile  README.md  skeleton.py  solution.py
```

++

Bomb Lab

+ `objdump -d bomb > disas.txt`

+ **Note:** kaboom **and** phase_defused

```
#pragma once
void phase_defused(void);
void kaboom(void);
void phase_one(char *);
void phase_two(char *, char *, char *);
void phase_three(int, int, int, int);
void phase_four(unsigned int *);
void phase_five(unsigned int *);
void phase_six(char *);
```

++

Cool Angr Projects

- + CGC entry: <https://github.com/mechaphish>
- + AFL + Angr for fuzzing:
<https://github.com/shellphish/driller>
- + Heap analysis:
<https://github.com/angr/heaphopper>

Outline

1. what the Hell is a SMT solver?
2. Z3-python
3. Lab - Cheating at Logic Challenges
4. Lab - Encoding CPU Instructions
5. Z3 in the Real world
6. Angr!
7. Lab - Using Angr in Anger
8. wrap-up

++

Wrap-up

Hopefully this got you started!

Grab me for any questions:

- + Now
- + at the con
- + with beer
- + via email (sam.brown@mwrinfosecurity.com)
- + via twitter [@_samdb_](https://twitter.com/_samdb_)
- + Whatever works...

++

Further Reading

Great resource:

https://yurichev.com/writings/SAT_SMT_draft-EN.pdf

Great paper:

http://openwall.info/wiki/_media/people/jvanegue/files/woot12.pdf