SIGPwny

SP2024 Week 12 • 2024-04-11

Symbolic Execution

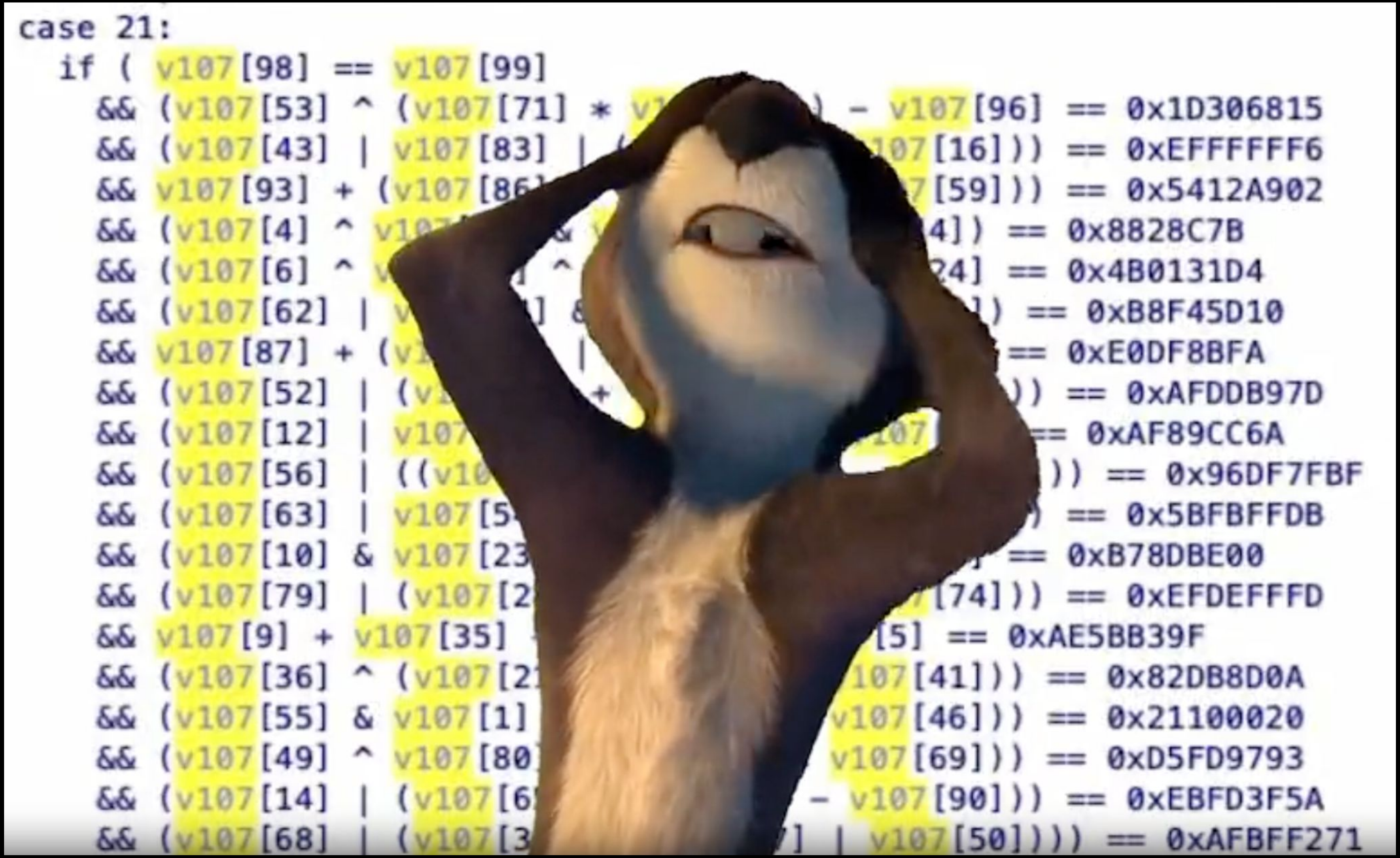
Nikhil Date and Pete Stenger

Announcements

- b01lersCTF 2024 - Tomorrow!
    - Friday 5 PM CST - Sunday 5PM CST
    - Details TBD, we will be playing in some fashion

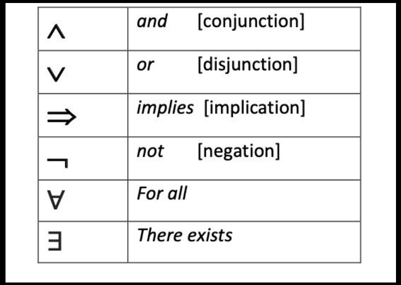
- Last chance for shirts: sigpwny.com/shirt2024 ctf.sigpwny.com

sigpwny{stat3_explos1on}



```
case 21:
  if ( v107[98] == v107[99]
    && (v107[53] ^ (v107[71] * v1          ) - v107[96] == 0x1D306815
    && (v107[43] | v107[83] | (          07[16])) == 0xEFFFFFF6
    && v107[93] + (v107[86               7[59])) == 0x5412A902
    && (v107[4] ^ v10          &         4]) == 0x8828C7B
    && (v107[6] ^ v          ] ^         24] == 0x4B0131D4
    && (v107[62] | v         ] &          ]) == 0xB8F45D10
    && v107[87] + (v          |              == 0xE0DF8BFA
    && (v107[52] | (v         +             )) == 0xAFDDB97D
    && (v107[12] | v107              107    == 0xAF89CC6A
    && (v107[56] | ((v10                     )) == 0x96DF7FBF
    && (v107[63] | v107[5                  ) == 0x5BFBFFDB
    && (v107[10] & v107[23                ] == 0xB78DBE00
    && (v107[79] | (v107[2              7[74])) == 0xEFDEFFFD
    && v107[9] + v107[35]              [5] == 0xAE5BB39F
    && (v107[36] ^ (v107[2              107[41])) == 0x82DB8D0A
    && (v107[55] & v107[1]              v107[46])) == 0x21100020
    && (v107[49] ^ v107[80              v107[69])) == 0xD5FD9793
    && (v107[14] | (v107[6             - v107[90])) == 0xEBFD3F5A
    && (v107[68] | (v107[3             ] | v107[50]))) == 0xAFBFF271
```

SAT/SMT Solvers



∧	*and* [conjunction]
∨	*or* [disjunction]
⇒	*implies* [implication]
¬	*not* [negation]
∀	*For all*
∃	*There exists*

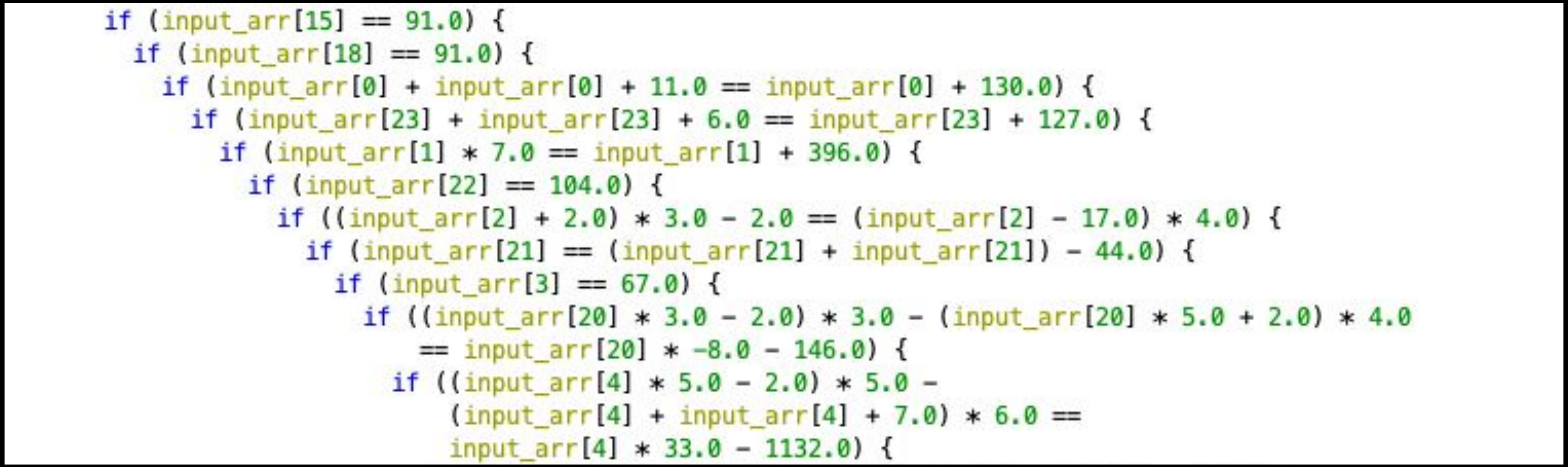
- SAT stands for satisfiability. SAT solvers solve propositional formulas like (¬p ∨ q ∨ r) ∧ (p ∨ ¬q ∨ ¬r)
  - Boolean SAT is NP-complete, but in practice many problems are tractable
- SMT stands for satisfiability modulo theories. SMT solvers allow non-logical operations, depending on the "theory"
  - but still solve a satisfiability problem

SMT Theories

- Integers
- Bitvectors
- Arrays
- IEEE Floats
- Reals
- Uninterpreted Functions (Blackbox Pure Functions)

Constraint solving

- Solve complex systems of equations
- z3 is an SMT solver
  - python library for solving constraints
  - `pip install z3-solver`



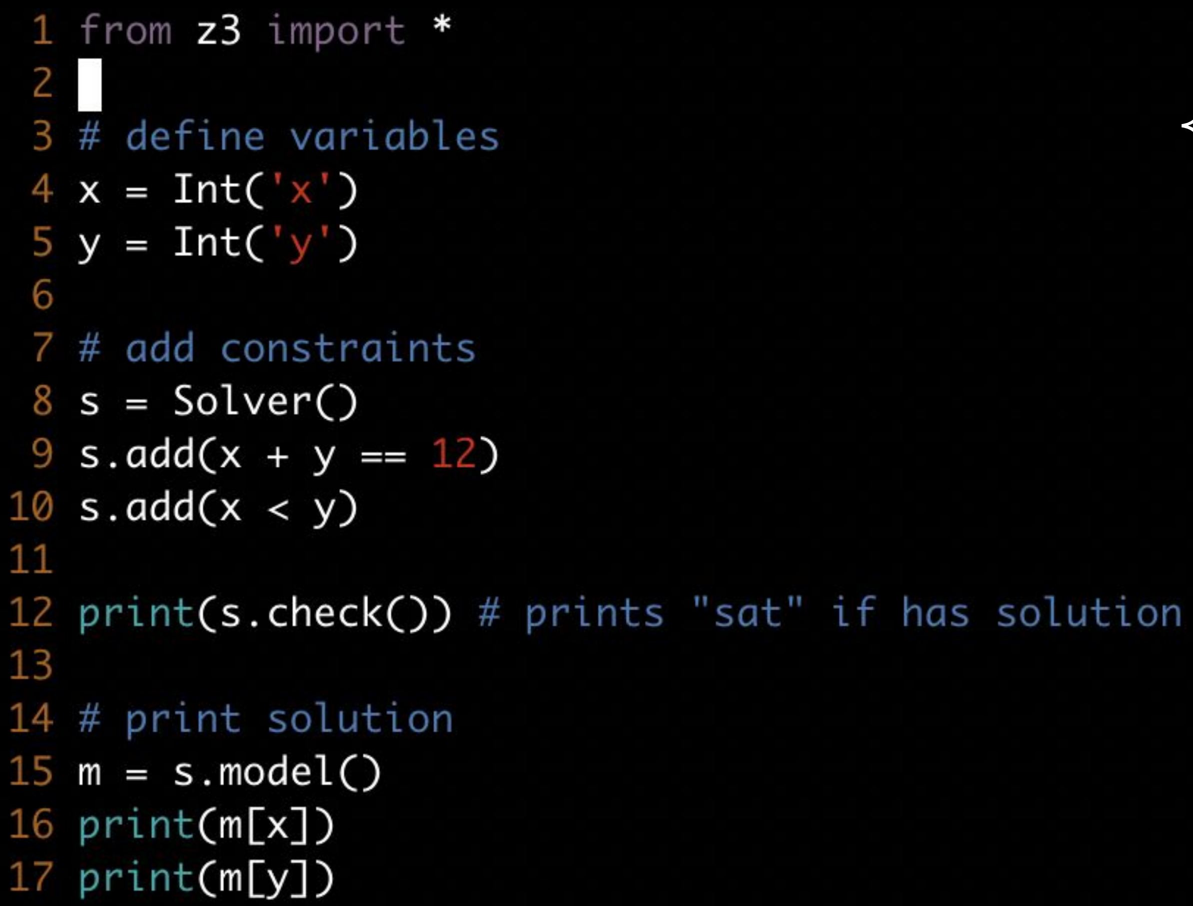
```
if (input_arr[15] == 91.0) {
  if (input_arr[18] == 91.0) {
    if (input_arr[0] + input_arr[0] + 11.0 == input_arr[0] + 130.0) {
      if (input_arr[23] + input_arr[23] + 6.0 == input_arr[23] + 127.0) {
        if (input_arr[1] * 7.0 == input_arr[1] + 396.0) {
          if (input_arr[22] == 104.0) {
            if ((input_arr[2] + 2.0) * 3.0 - 2.0 == (input_arr[2] - 17.0) * 4.0) {
              if (input_arr[21] == (input_arr[21] + input_arr[21]) - 44.0) {
                if (input_arr[3] == 67.0) {
                  if ((input_arr[20] * 3.0 - 2.0) * 3.0 - (input_arr[20] * 5.0 + 2.0) * 4.0
                      == input_arr[20] * -8.0 - 146.0) {
                    if ((input_arr[4] * 5.0 - 2.0) * 5.0 -
                        (input_arr[4] + input_arr[4] + 7.0) * 6.0 ==
                        input_arr[4] * 33.0 - 1132.0) {
```

API of Z3 Py

- "Sorts": data types (Int, BitVec, Real, Array)
- Operators (are theory-specific)
  - Logical operators (Or, And, Not, Implies)
  - Arithmetic operators (+, -, *, /)
  - Inequalities and equality (==, >, <, >=, <=)
  - Bitvector operators (bitwise operations, bit shifting)
- Constraints
- "Model": assignment of values to "variables" that satisfies all constraints
- Good resource: https://ericpony.github.io/z3py-tutorial/guide-examples.htm

Z3 Basics

```
pip install z3-solver
```



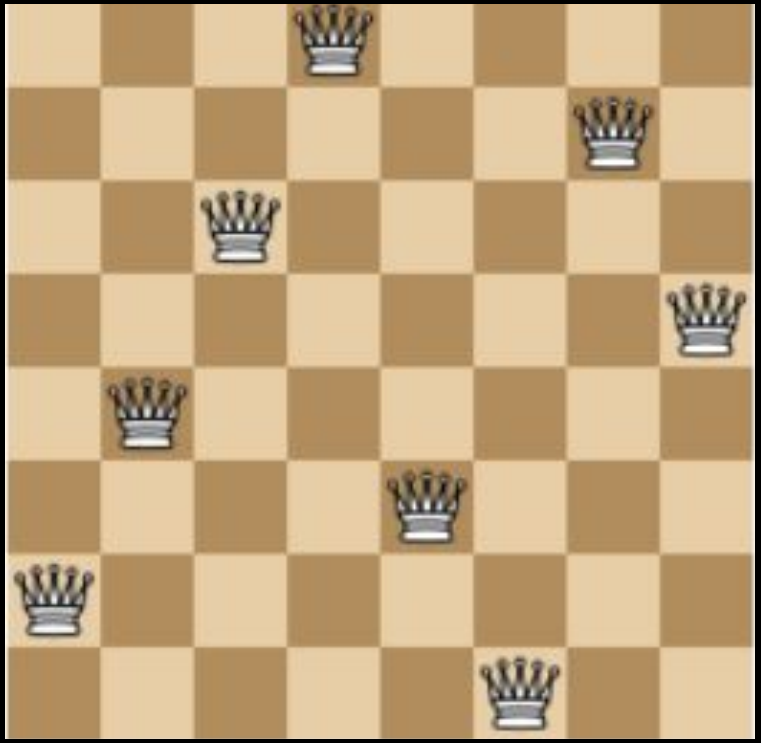
```python
from z3 import *

# define variables
x = Int('x')
y = Int('y')

# add constraints
s = Solver()
s.add(x + y == 12)
s.add(x < y)

print(s.check()) # prints "sat" if has solution

# print solution
m = s.model()
print(m[x])
print(m[y])
```

$$\begin{cases} x + y = 12 \\ x < y \end{cases}$$

(Note: this finds any of the possible solutions)

Z3 is Powerful

```
pip install z3-solver
```

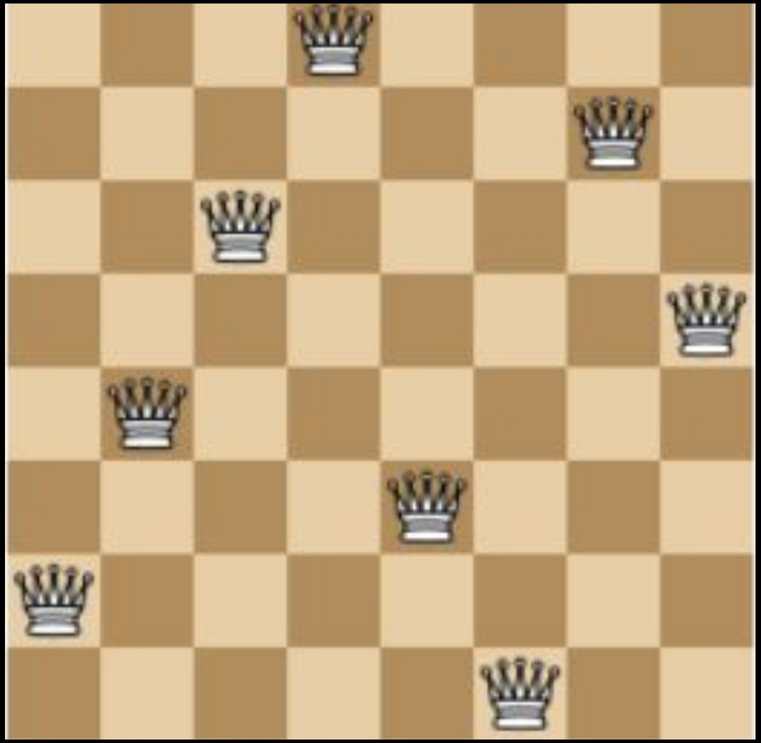


```
Q = [ Int('Q_%i' % (i + 1)) for i in range(8) ]

XXX = [ And(1 <= Q[i], Q[i] <= 8) for i in range(8) ]

YYY = [ Distinct(Q) ]

ZZZ = [ If(i == j,
             True,
             And(Q[i] - Q[j] != i - j, Q[i] - Q[j] != j - i))
         for i in range(8) for j in range(i) ]

solve(XXX + YYY + ZZZ)
```

What does this line do?

Z3 is Powerful

```
pip install z3-solver
```

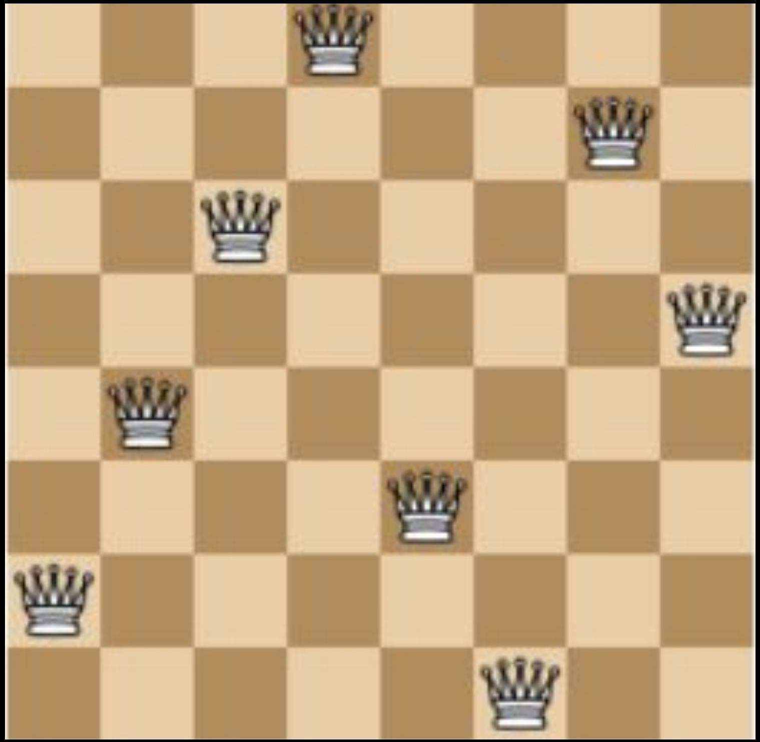


```
Q = [ Int('Q_%i' % (i + 1)) for i in range(8) ]

# Each queen is in a column {1, ... 8 }
val_c = [ And(1 <= Q[i], Q[i] <= 8) for i in range(8) ]

YYY = [ Distinct(Q) ]

ZZZ = [ If(i == j,
             True,
             And(Q[i] - Q[j] != i - j, Q[i] - Q[j] != j - i))
          for i in range(8) for j in range(i) ]

solve(val_c + YYY + ZZZ)
```

What does this line do?

Z3 is Powerful

```
pip install z3-solver
```

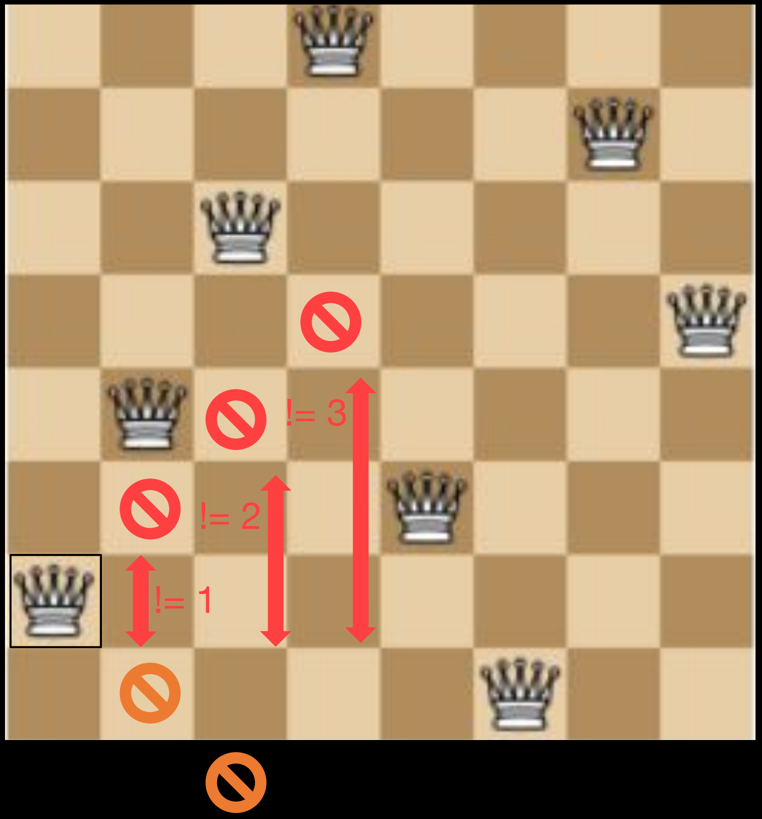


```
Q = [ Int('Q_%i' % (i + 1)) for i in range(8) ]

# Each queen is in a column {1, ... 8 }
val_c = [ And(1 <= Q[i], Q[i] <= 8) for i in range(8) ]

# At most one queen per column
col_c = [ Distinct(Q) ]

ZZZ = [ If(i == j,
             True,
             And(Q[i] - Q[j] != i - j, Q[i] - Q[j] != j - i))
          for i in range(8) for j in range(i) ]

solve(val_c + col_c + ZZZ)
```

What does this line do?

Z3 is Powerful

```
pip install z3-solver
```



```
Q = [ Int('Q_%i' % (i + 1)) for i in range(8) ]

# Each queen is in a column {1, ... 8 }
val_c = [ And(1 <= Q[i], Q[i] <= 8) for i in range(8) ]

# At most one queen per column
col_c = [ Distinct(Q) ]

# Diagonal constraint
diag_c = [ If(i == j,
              True,
              And(Q[i] - Q[j] != i - j, Q[i] - Q[j] != j - i))
           for i in range(8) for j in range(i) ]

solve(val_c + col_c + diag_c)
```

Z3 Challenge

```
pip install z3-solver
```

System of diophantine equations

- (only integer solutions)
- Hard to solve normally

```
(y - 123456)^2 = (x - 234567)^3 - 2
```

```
from z3 import *
x = Int('x')
// ??
s = Solver()

// change line below
s.add(???)

if s.check():
 print(s.model())
```

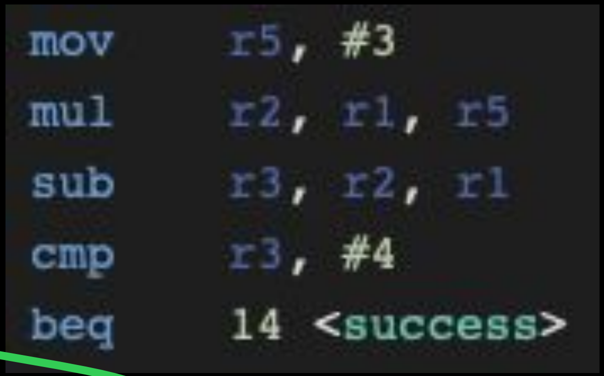
submit: `sigpwny{x + 2y}`

Your turn! ~2 minutes to try this out

Symbolic Execution

- Solve for inputs
  - Generate constraints from program **automatically**

```
x = ?
y = x * 3
z = y - x
```



```
mov     r5, #3
mul     r2, r1, r5
sub     r3, r2, r1
cmp     r3, #4
beq     14 <success>
```

- Solve for x such that z == 4

Input

Constraint

Symbolic Execution Usages

- Reversing without reversing
  - Solve for input on stdin (flag) such that the flag checker prints “That flag is correct!”
- Automated PWN
  - Solve for input such that the instruction pointer is overwritten
- Research: binary instrumentation and analysis

Angr

- Symbolic execution on binaries
- Angr can be used for automating CTF chals
- Install with `pip install angr`
- Template (e.g. for **angry challenge**):
  - https://gist.github.com/richyliu/33489063d02c0a2afe0d6de6ec8d3e07

Angr CTF Challenge

- https://github.com/angr/angr-examples/tree/master/examples/b01lersctf2020_little_engine
- Standard (basic) rev challenge
  - gets input from the user
  - does some validation
  - tells you if it's correct

Angr Tips

- Running out of memory?
  - Set environment variable `REUSE_Z3_SOLVER=1`
    - Avoids cloning z3 solver when state splits
  - Add `veritesting=True` argument to `simulation_manager`
    - Automatically identifies merge points
  - Set LAZY_SOLVES flag
    - Defer evaluation as far as possible

Angr Internals

- Uses z3 for constraint solving and symbolic manipulation
- Steps through program
  - splits states when it encounters a branch
- **“State”**: represents program state (memory, registers, etc.)
  - States have "path conditions"
- **Stashes**: collections of states (active, found, deadended, errored)
- **Simulation Managers**: control how search proceeds

A Problem

- State explosion
  - Repeated branching can cause the number of states to become unmanageable

State Explosion Example

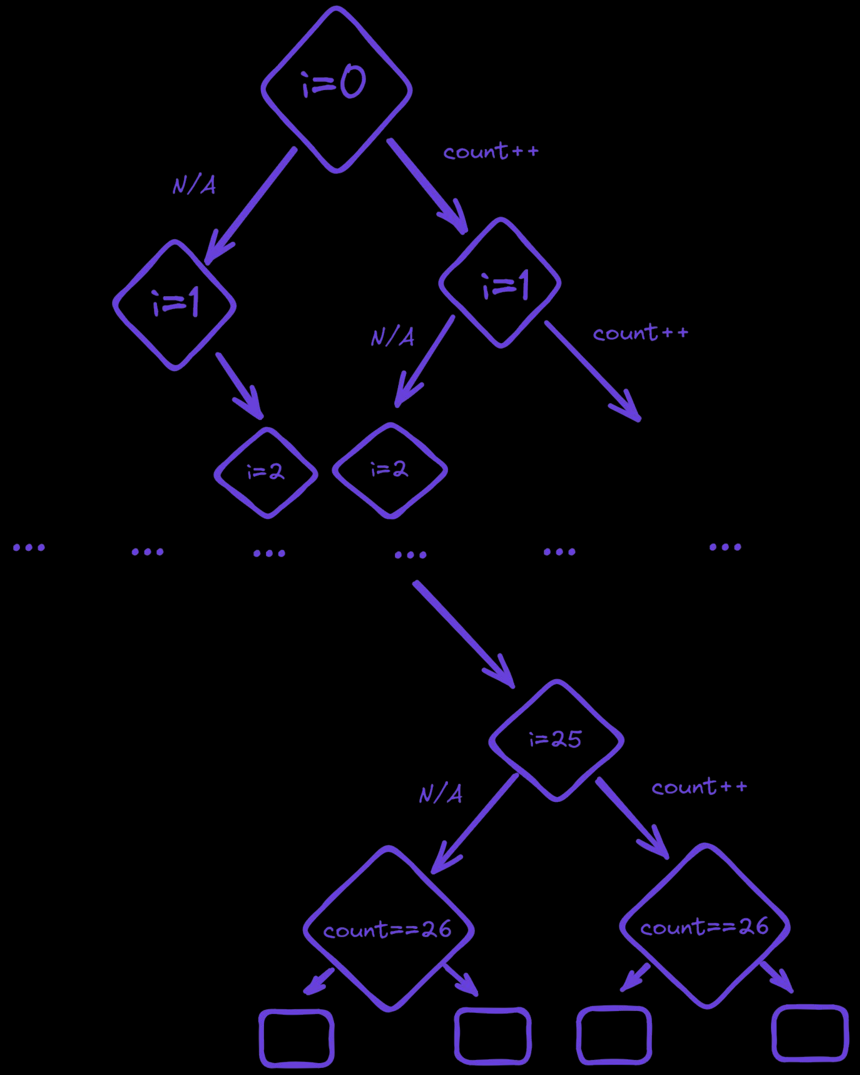
```c
#include <stdio.h>
int main() {
    char buf[27];
    fgets(buf, 27, stdin);
    char target[] = "abcdefghijklmnopqrstuvwxyz";
    int count = 0;
    for (int i = 0; i < 26; i++) {
        if (buf[i] == target[i]) {
            count++;
        }
    }
    if (count == 26) {
        printf("correct\n");
    } else {
        printf("wrong\n");
    }
}
```

How many branches would this create?

State Explosion Example

```c
#include <stdio.h>
int main() {
    char buf[27];
    fgets(buf, 27, stdin);
    char target[] = "abcdefghijklmnopqrstuvwxyz";
    int count = 0;
    for (int i = 0; i < 26; i++) {
        if (buf[i] == target[i]) {
            count++;
        }
    }
    if (count == 26) {
        printf("correct\n");
    } else {
        printf("wrong\n");
    }
}
```

2^(26+1) = a lot



Going Further

- Angr's behavior can be modified/instrumented/customized
- Research
  - Fuzzware
    - uses angr for more effective fuzzing
    - reduces "input overhead"
  - Libmatch
    - uses angr as a static analysis tool

Next Meetings

`2024-04-14` • **Tomorrow (Friday)**

- b01lersCTF 2024 starts at 5 PM
- More info in Discord soon

`2024-04-18` • **This Sunday**

- Location-based OSINT with Henry
- Become rainbolt

`YYYY-MM-DD` • **Next Thursday**

- Social Engineering with Emma and Sagnik
- Learn how to manipulate people

ctf.sigpwny.com

sigpwny{stat3_explos1on}

Meeting content can be found at sigpwny.com/meetings.

SIGPwny