Time to get Angr(y)

An Introduction to Symbolic Execution

Hello!

WONG Wai Tuck

MSc Information Security, CMU
BSc Information Systems/Applied Stats, SMU

Nmap Contributor

OSCP/OSCE Certified

You can find me at

@wongwaituck (LinkedIn/Github)

@waituckk (Twitter)

Pre Workshop Administrivia

Stuff to do in your VM

Clone our repo!

- git clone…

Enable 32-bit support

- sudo dpkg --add-architecture i386
- sudo apt-get update
- sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386

Slide/Binary Credits

- https://github.com/jakespringer/angr_ctf

Why Are We Here?

What's the thing about Symbolic Execution?

Finding bugs via test cases

```
user_input = raw_input('Enter the secret letter: ')
if type(user_input) == chr and user_input == 'a':
  shell()
else:
  print 'Try again.'
```

Fuzzing (when it works well)

- User input: **1 char**
- A character is 1 byte == 8 bits
- Each bit can be 0 or 1
- User input space: 2^8

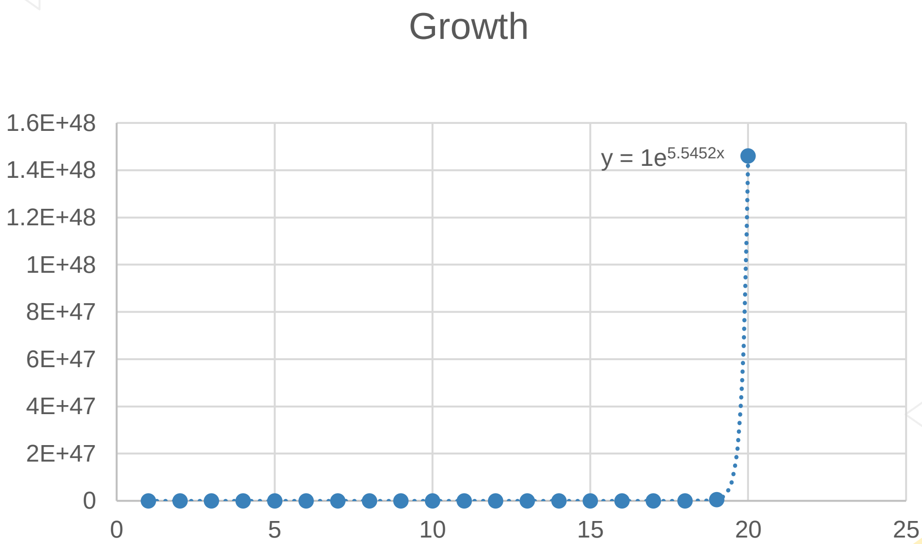
- Bruteforce **2^8 = 256 possibilities!**

Fuzzing (when it doesn’t work so well)

```
user_input = raw_input('Enter the password: ')
if user_input == 'hunter2':
  print 'Success.'
else:
  print 'Try again.'
```

Search Space

- User input space $(2^8)^n$
  - n is the number of letters



Fuzzing (when it doesn't work so well)

```
# A complex guessing game.
def encrypt(string, amount):
  for i in range(0, len(string)):
    string[i] += amount

user_input = raw_input('Enter the password: ')
if encrypt(user_input, amount=1) ==
encrypt('hunter2', amount=2):
  print 'Success.'
else:
  print 'Try again.'
```

You want symbolic execution if...

- You care about the **correctness** of your programs
- You want to **remove dead/unreachable code**
- You want to figure out **what input** is required to get to a certain state in a program

What is Symbolic Execution?

Introduction to Program Analysis

The Golden Question

Will the assertion be hit in this toy program and if so, what are the satisfying inputs?

```
void foo(int x, int y){
    int t = 0;
    if (x > y){
        t = x;
    } else{
        t = y;
    }

    if (t < x){
        assert false;
    }
}
```

Credits: Originally given in Papers We Love by Wai Tuck, edited by Benjamin Lim

Using Concrete Execution

Concrete Execution

- Concrete Values
  - x = 4
  - y = 4

```
void foo(int x, int y){
    int t = 0;
    if (x > y){
        t = x;
    } else{
        t = y;
    }

    if (t < x){
        assert false;
    }
}
```

Credits: Originally given in Papers We Love by Wai Tuck, edited by Benjamin Lim

Concrete Execution

- Concrete Values
  - x = 4
  - y = 4
  - t = 0

```
void foo(int x, int y){
    int t = 0;
    if (x > y){
        t = x;
    } else{
        t = y;
    }

    if (t < x){
        assert false;
    }
}
```

Credits: Originally given in Papers We Love by Wai Tuck, edited by Benjamin Lim

Concrete Execution

- Concrete Values
  - x = 4
  - y = 4
  - t = 4

```
void foo(int x, int y){
    int t = 0;
    if (x > y){
        t = x;
    } else{
        t = y;
    }

    if (t < x){
        assert false;
    }
}
```

Credits: Originally given in Papers We Love by Wai Tuck, edited by Benjamin Lim

Concrete Execution

- Concrete Values
  - x = 4
  - y = 4
  - t = 4

```
void foo(int x, int y){
    int t = 0;
    if (x > y){
        t = x;
    } else{
        t = y;
    }

    if (t < x){
        assert false;
    }
}
```

Credits: Originally given in Papers We Love by Wai Tuck, edited by Benjamin Lim

Concrete Execution

- Concrete Values
  - x = 4
  - y = 4
  - t = 4
- t < x ⇔

4 < 4 is **false**!

```
void foo(int x, int y){
    int t = 0;
    if (x > y){
        t = x;
    } else{
        t = y;
    }

    if (t < x){
        assert false;
    }
}
```

Credits: Originally given in Papers We Love by Wai Tuck, edited by Benjamin Lim

Concrete Execution

- Concrete Values
  - x = 4
  - y = 4
  - t = 4
- t < x ⇔

4 < 4 is **false**!

Assertion not reached!

```
void foo(int x, int y){
    int t = 0;
    if (x > y){
        t = x;
    } else{
        t = y;
    }

    if (t < x){
        assert false;
    }
}
```

Credits: Originally given in Papers We Love by Wai Tuck, edited by Benjamin Lim

Using Static Symbolic Execution

We try to lift the entire program into a mathematical model!

Symbolic Execution

- Symbolic Values
  - x = X
  - y = Y

```
void foo(int x, int y){
    int t = 0;
    if (x > y){
        t = x;
    } else{
        t = y;
    }

    if (t < x){
        assert false;
    }
}
```

Credits: Originally given in Papers We Love by Wai Tuck, edited by Benjamin Lim

Symbolic Execution

- Symbolic Values
  - x = X
  - y = Y
  - t = 0

```
void foo(int x, int y){
    int t = 0;
    if (x > y){
        t = x;
    } else{
        t = y;
    }

    if (t < x){
        assert false;
    }
}
```

Credits: Originally given in Papers We Love by Wai Tuck, edited by Benjamin Lim

Symbolic Execution

- Symbolic Values
  - x = X
  - y = Y
  - t = ite(X>Y, X,Y)

```
void foo(int x, int y){
    int t = 0;
    if (x > y){
        t = x;
    } else{
        t = y;
    }

    if (t < x){
        assert false;
    }
}
```

Credits: Originally given in Papers We Love by Wai Tuck, edited by Benjamin Lim

If-then-else

ite(X>Y, X, Y)

```
If (X>Y):
        return X
Else:
        return Y
```

Symbolic Execution

- Symbolic Values
  - x = X
  - y = Y
  - t = ite(X>Y, X,Y)
- Assert condition
- ite(X>Y, X, Y) < X

```
void foo(int x, int y){
    int t = 0;
    if (x > y){
        t = x;
    } else{
        t = y;
    }

    if (t < x){
        assert false;
    }
}
```

Credits: Originally given in Papers We Love by Wai Tuck, edited by Benjamin Lim

Symbolic Execution

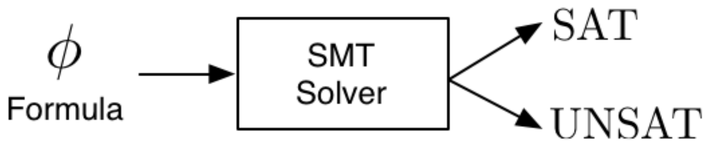
- Symbolic Values
  - x = X
  - y = Y
  - t = ite(X>Y, X,Y)
- Assert condition
- ite(X>Y, X, Y) < X
- Throw into SMT solver!

```
void foo(int x, int y){
    int t = 0;
    if (x > y){
        t = x;
    } else{
        t = y;
    }

    if (t < x){
        assert false;
    }
}
```

Credits: Originally given in Papers We Love by Wai Tuck, edited by Benjamin Lim

SMT Solver

- Solve formula using SMT Solvers (e.g. Z3)
- Input: Boolean formula
- Output:
  -
  -



SMT Solver Demo

```
waituck@DESKTOP-84MPHS7:~⟫ cat solve.py
#!/usr/bin/env python
from z3 import *

x = Int('X')
y = Int('Y')
t = If(x > y, x, y)

solve(t < x)
waituck@DESKTOP-84MPHS7:~⟫ ./solve.py
no solution
```

Symbolic Execution

- Symbolic Values
  - x = X
  - y = Y
  - t = ite(X>Y, X,Y)
- Assert condition
- ite(X>Y, X, Y) < X
- We will never reach the assert.

```
void foo(int x, int y){
    int t = 0;
    if (x > y){
        t = x;
    } else{
        t = y;
    }

    if (t < x){
        assert false;
    }
}
```

Credits: Originally given in Papers We Love by Wai Tuck, edited by Benjamin Lim

Concolic Execution

Concrete + Symbolic

“Dynamic Symbolic Execution”

Formal Definition

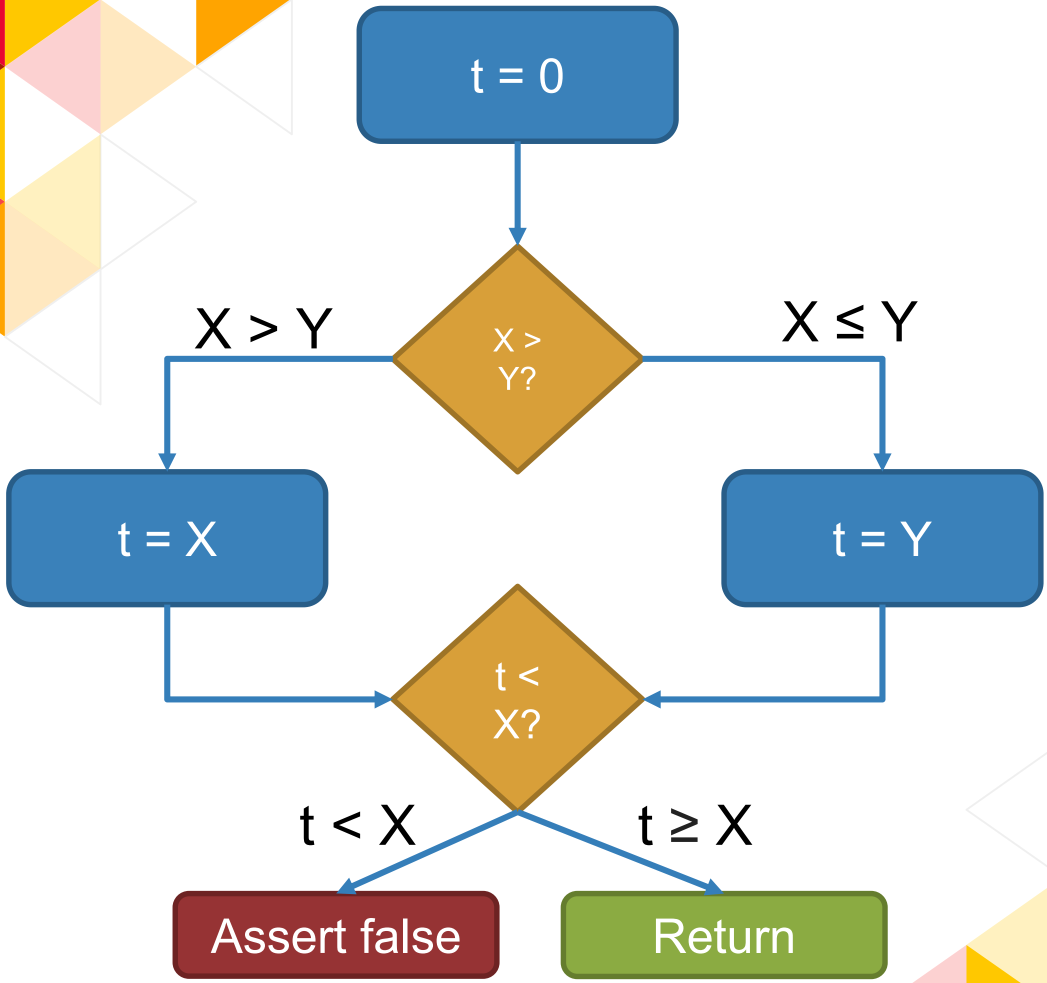
- A symbolic engine is defined by the following:
  - *stmt*: the next statement to evaluate
  - σ: a mapping of program variables to symbolic expressions
  - π: path constraints imposed at point of execution

Concolic Execution

- Symbolic Values
  - x = X
  - y = Y

```
void foo(int x, int y){
    int t = 0;
    if (x > y){
        t = x;
    } else{
        t = y;
    }

    if (t < x){
        assert false;
    }
}
```

Credits: Originally given in Papers We Love by Wai Tuck, edited by Benjamin Lim


t = 0
X > Y?
X > Y
X ≤ Y
t = X
t = Y
t < X?
t < X
t ≥ X
Assert false
Return

Concolic Execution

- Symbolic Values
  - x = X
  - y = Y
  - t = 0

```
void foo(int x, int y){
    int t = 0;
    if (x > y){
        t = x;
    } else{
        t = y;
    }

    if (t < x){
        assert false;
    }
}
```

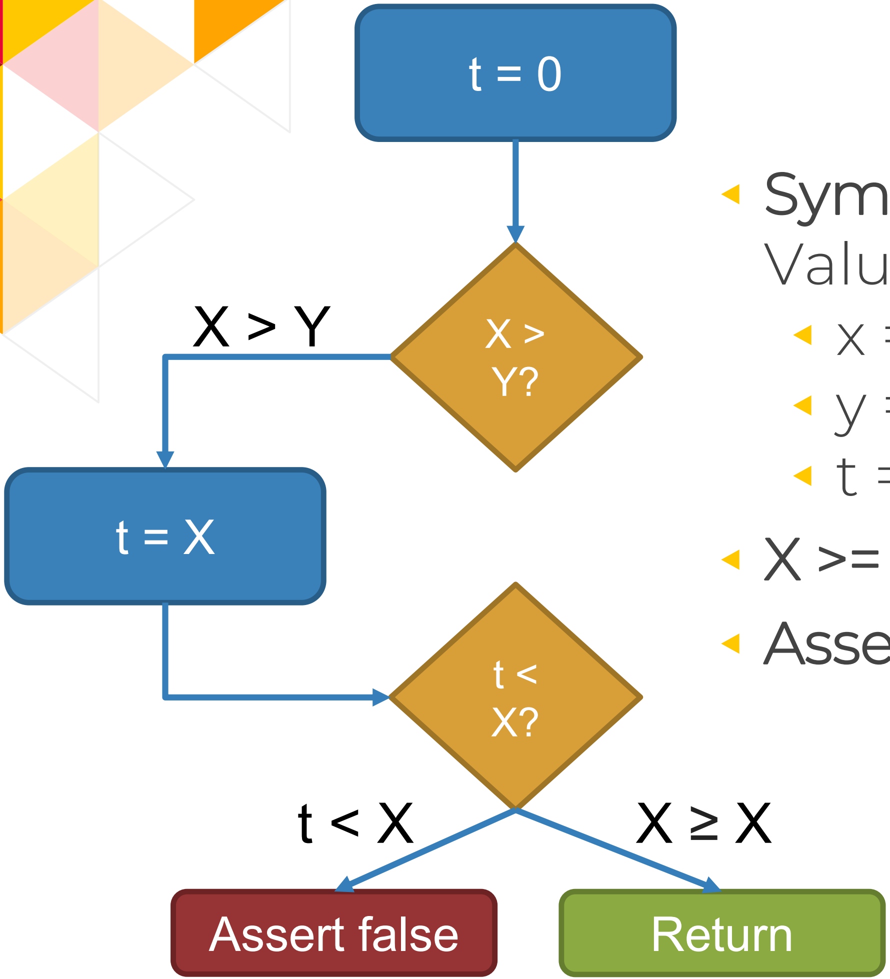
Credits: Originally given in Papers We Love by Wai Tuck, edited by Benjamin Lim

Concolic Execution

- Symbolic Values
  - x = X
  - y = Y
  - t = 0
- Case split on Conditional

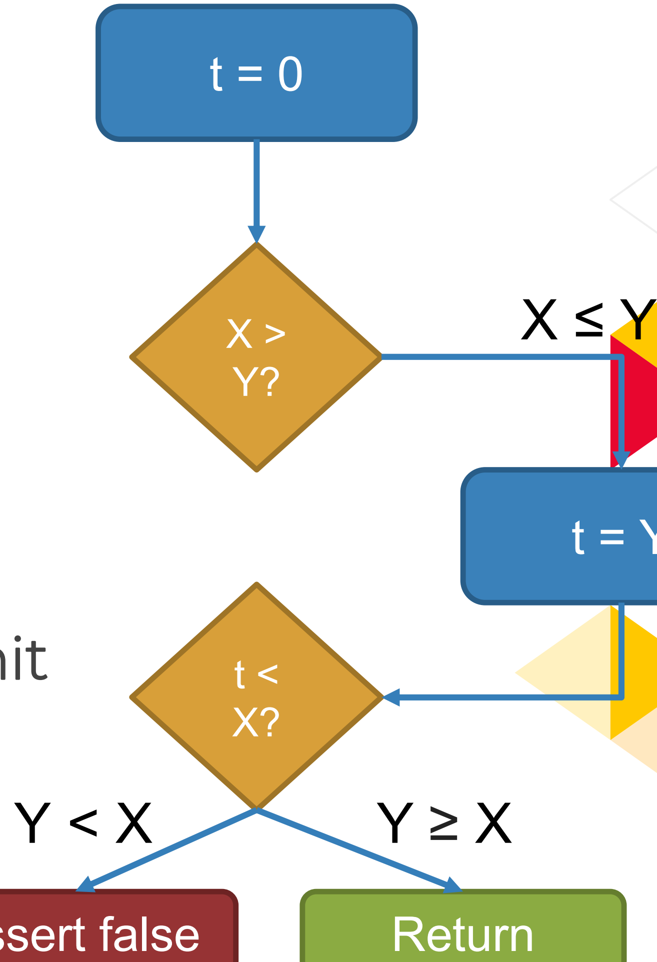
```
void foo(int x, int y){
    int t = 0;
    if (x > y){
        t = x;
    } else{
        t = y;
    }

    if (t < x){
        assert false;
    }
}
```

Credits: Originally given in Papers We Love by Wai Tuck, edited by Benjamin Lim



- Symbolic Values
  - x = X
  - y = Y
  - t = X
- X >= X
- Assert not hit

- Symbolic Values
  - x = X
  - y = Y
  - t = Y
- Y >= X
- Assert not hit



Caveat: Symbolic Forward Execution vs Backward Execution

- We have talked about **forward** symbolic execution thus far
- Note that **backward** symbolic execution can also be done

Angr API Walkthrough

Challenge Binary Details

- Each binary takes in a password and prints success

- Our goal is to find that password!

For those that want to go fast

https://docs.angr.io/

http://angr.io/api-doc

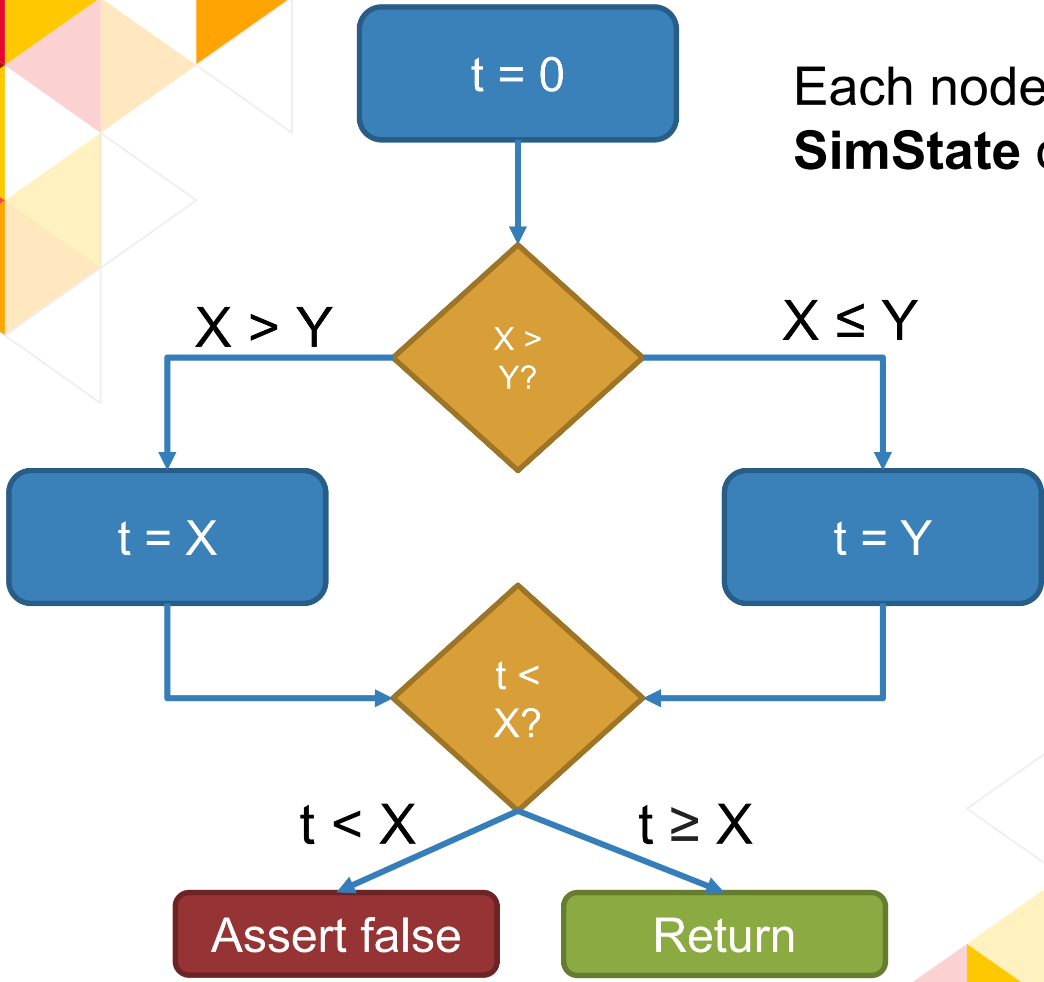
Challenge 1:

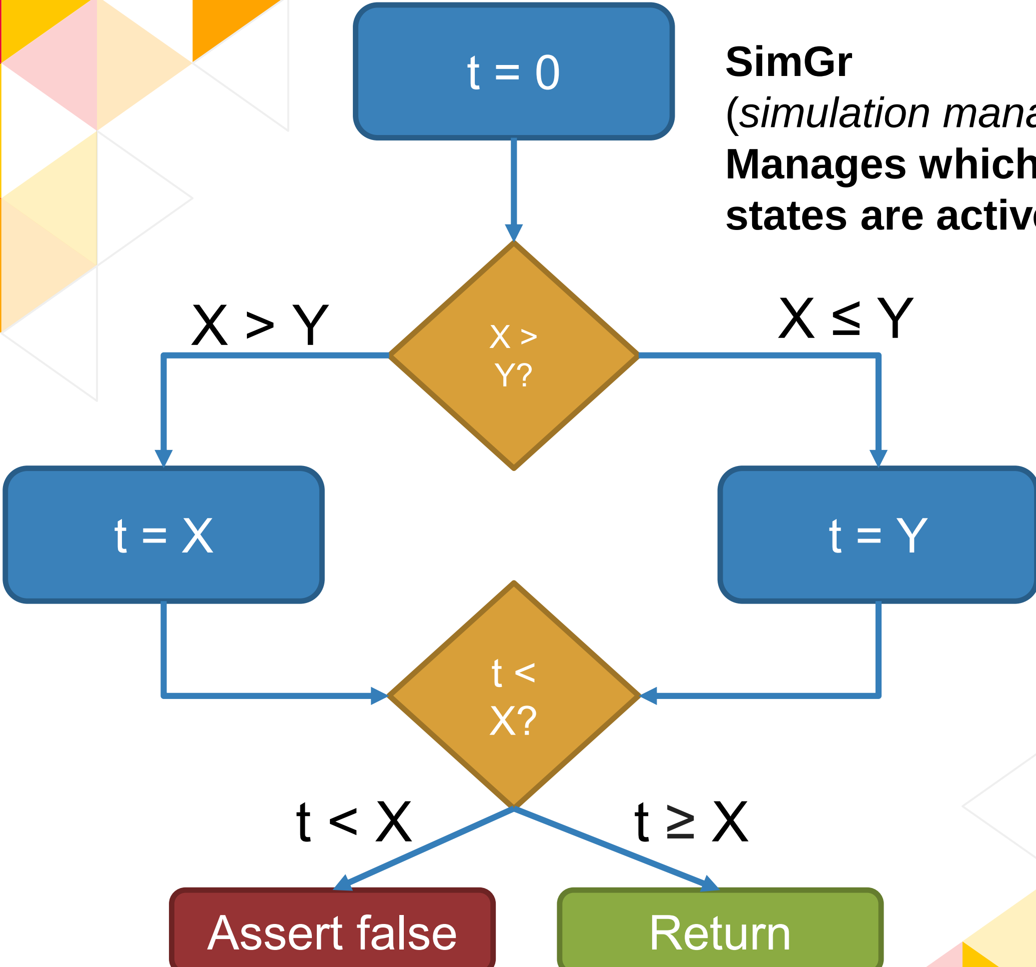
Find/Avoid

What is Angr?

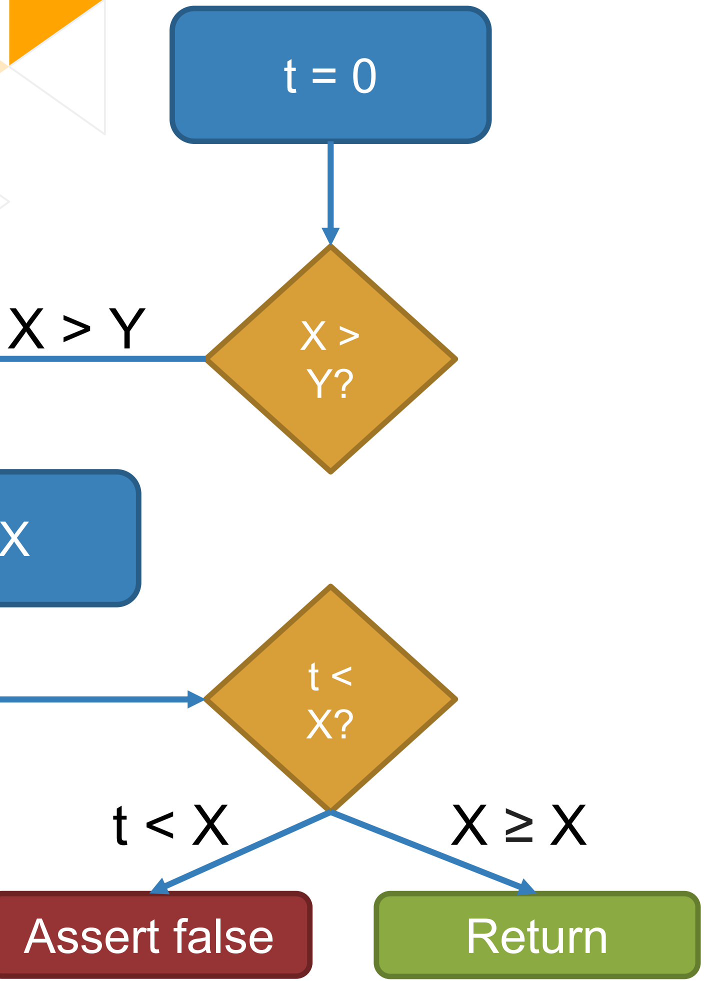
- Symbolic execution engine written in Python
  - Step through binaries and follow any feasible branch
  - Search for a program state satisfying some constraint
  - Solve for symbolic variables

Each node is a **SimState** object

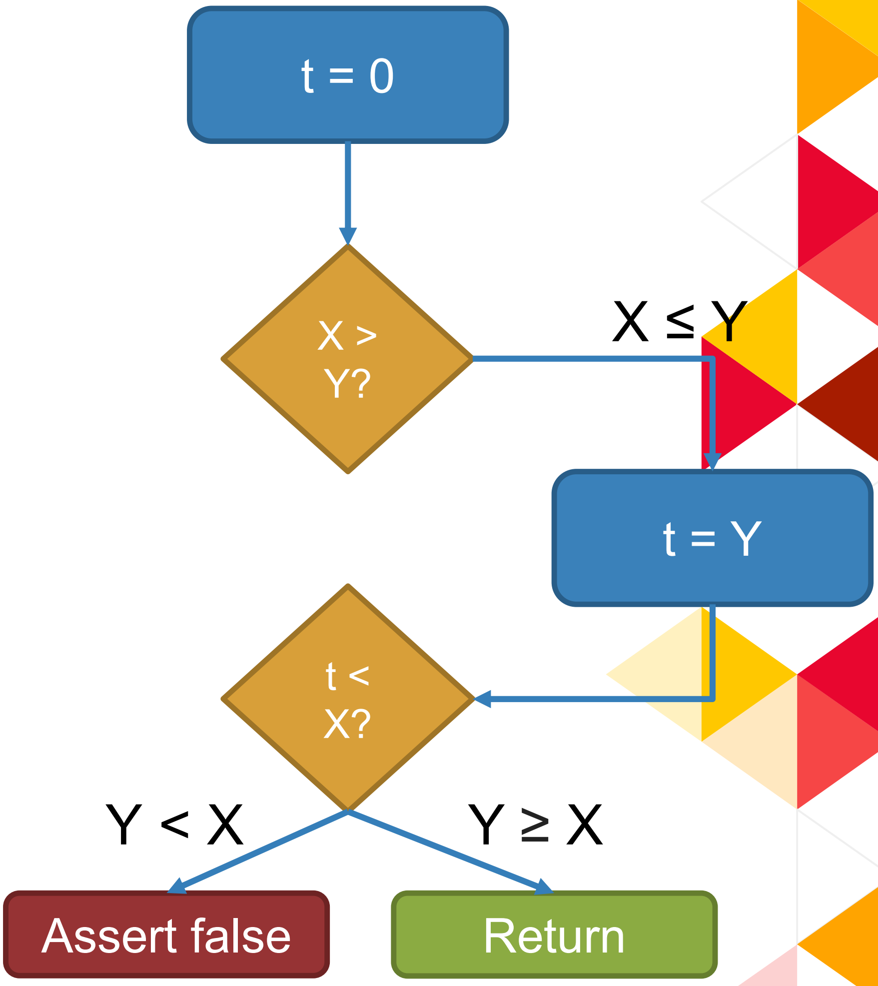




SimGr
(*simulation manager*)
Manages which states are active



t = 0
X > Y?
X > Y
t = X
t < X?
t < X
X ≥ X
Assert false
Return



t = 0
X > Y?
X ≤ Y
t = Y
t < X?
Y < X
Y ≥ X
Assert false
Return

Searching...

Method 1: Search for an instruction address

Perhaps we want to find how to reach this address.

```
804867a:   sub    $0xc,%esp
804867d:   push   $0x8048760
8048682:   call   8048400
<backdoor@plt>
8048687:   add    $0x10,%esp
```

Method 2: Search for anything else!

Perhaps we want to find when the variable ‘success’ is equal to true.

Any arbitrary function that determines if we have reached a state we want would work.

But sea

2^n

Each node is a **condition**!

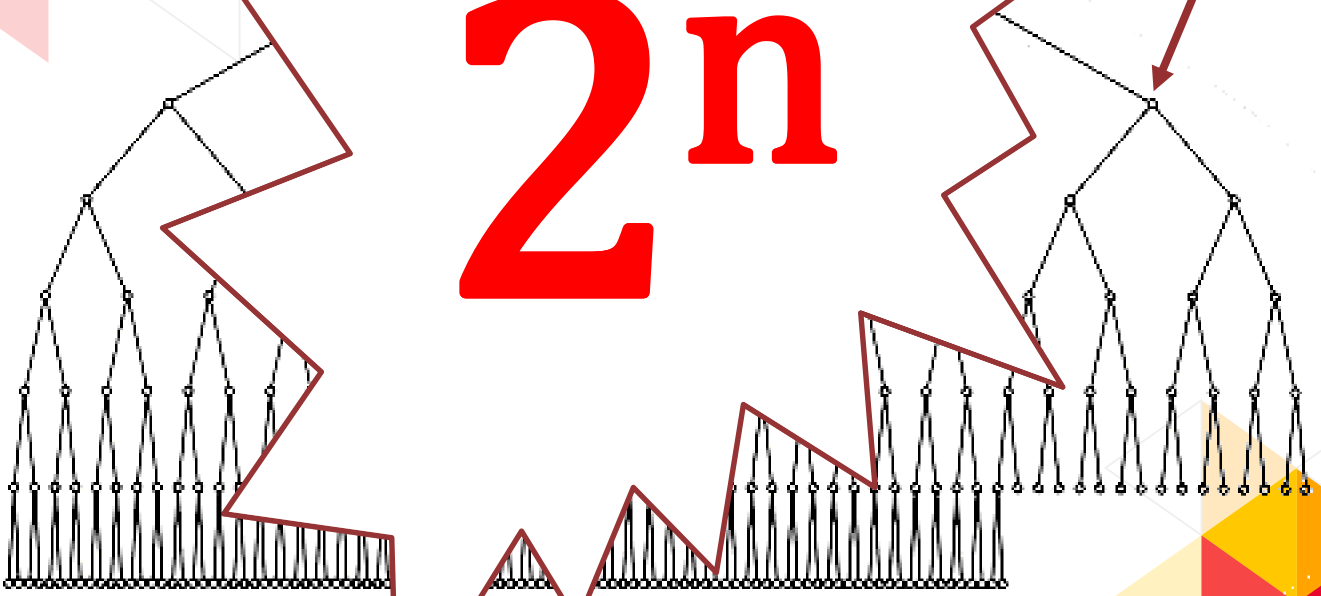


Image source: http icodeguru.com/vc/10book/books/book3/chap6.htm

State Space Explosion

```
For I in range(len(input)):
        if string[i] != ‘Z’:
                return False

Return True
```

Siam ahhhh

Set constraints to AVOID the path **(the same way we search!)**

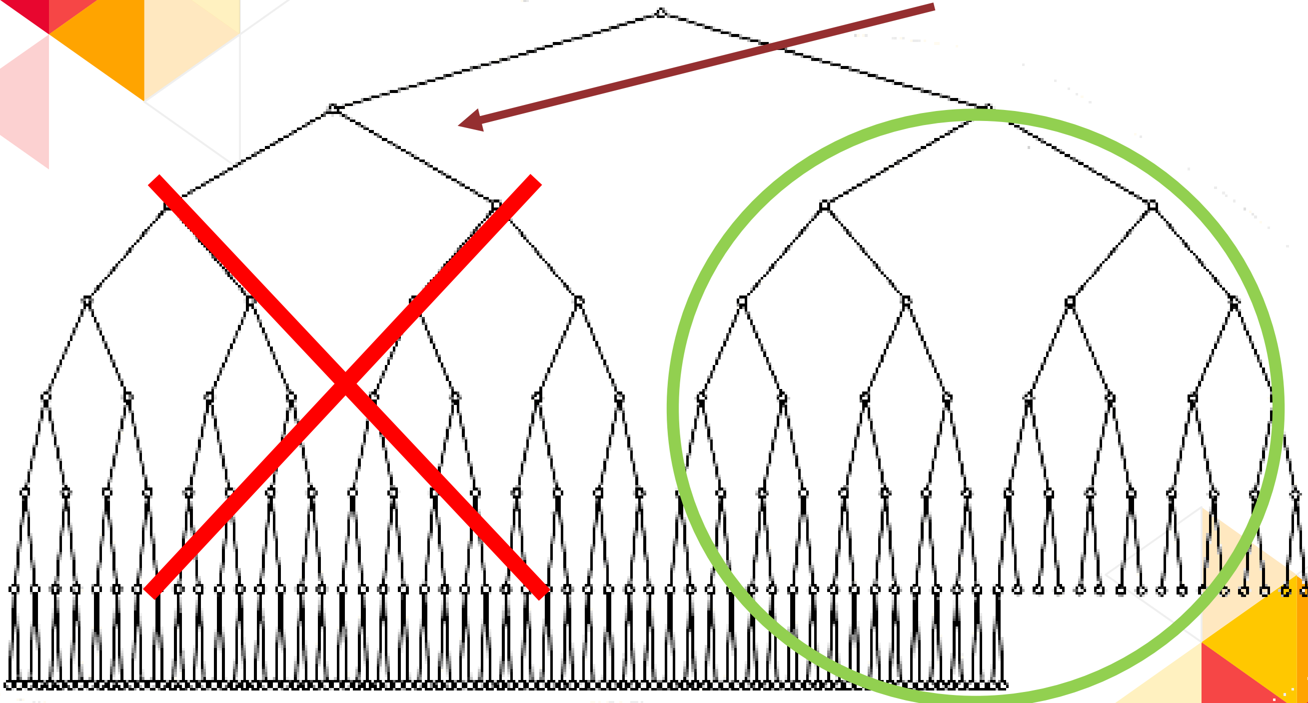


Image source: http://www.icodeguru.com/vc/10book/books/book3/chap6.htm

Algorithm Part 1

- Disassemble with IDA to find out where to go (and not go)
- Load the binary as a project
  - *angr.Project(<file_path>)*
- Initialize the state
  - *proj.factory.entry_state()*
- Set up the simulation manager
  - *proj.factory.simgr(state)*

Algorithm Part 2

- Explore!
  - simgr.explore(find=find, avoid=avoid)
- Get the first satisfiable state in simgr.found
- Get the input from stdin
  - found.posix.dumps(0)

Angr debugging tips

- import logging
- logging.getLogger('angr').setLevel('DEBUG')

HANDS ON

Challenge 2:

When You Can’t Reverse ☹

Challenge 3:

Declaring Your Own Symbolic Variables

More Manual Labor

- scanf(“%u %u”, &i, &j)
  - Takes in 2 integers that are size 32 bits, store one in i, store the other in j
- We want to exploit our knowledge of the program as much as possible

Bitvectors

- Claripy is an interface provided by angr that abstracts away the underlying SMT solver
- So our Int(‘X’) now becomes:
  - claripy.BVS(‘X’, 32)
  - ‘X’ is the name
  - 32 is the size (in bits)

Bitvectors

- claripy.BVS('pass', 4)

A	B	C	D

Where A, B, C, D are symbolic variables that are constrained to 0 or 1

Other BitVectors

Definitions:

- A *concrete* bitvector: a bitvector that can take on exactly 1 value.
- (Example: { λ: λ = 1 })
- claripy.BVV(val, size_in_bits)

- A *symbolic* bitvector: a bitvector that can take on more than 1 value.
- (Example: { λ: λ > 10 })

Other BitVectors

- An *unsatisfiable* bitvector: a bitvector that cannot take on any values.
- (Example: { λ: λ = 10, λ ≠ 10 })
- An *unconstrained* bitvector: a bitvector that can take on any value, within the bounds of its size.
  - Represented as Unconstrained states in Angr

Accessing Memory

- We need to associate the bitvector with the memory region
  - state.mem[state.regs.ebp - 0xc].int = password0
- Where password0 is a claripy.BVS

Gotta Skip Scanf

- Angr does not handle multiarg scanf
  - That means **we got to skip it**!

- Easiest way: start after the call to scanf
  - HINT: use *blank_state(<addr>)*

Getting the Answer

- Get the backend SMT solver instance
  - *found.se*
- Evaluate the symbolic expression
  - *solver.eval(<BVS>)*

Challenge 4:

Declaring Your Own Methods With angr.SimProcedure

Remember This?

```
For I in range(len(input)):
        if string[i] != ‘Z’:
                return False

Return True
```

This binary has something similar!

Angr.SimProcedure

```
class SimProcName(angr.SimProcedure):
  def run(self, param1, param2):
          # self.state is accessible
          # do stuff
          return claripy.BVV(1)
```

Loading Memory

- state.memory.load(addr, length)

ITE in claripy

- claripy.If(a==b, claripy.BVV(1,32), claripy.BVV(0,32))

Hooking symbols

- sym = “function_name”
- proj.hook_symbol(sym, SimProcName())

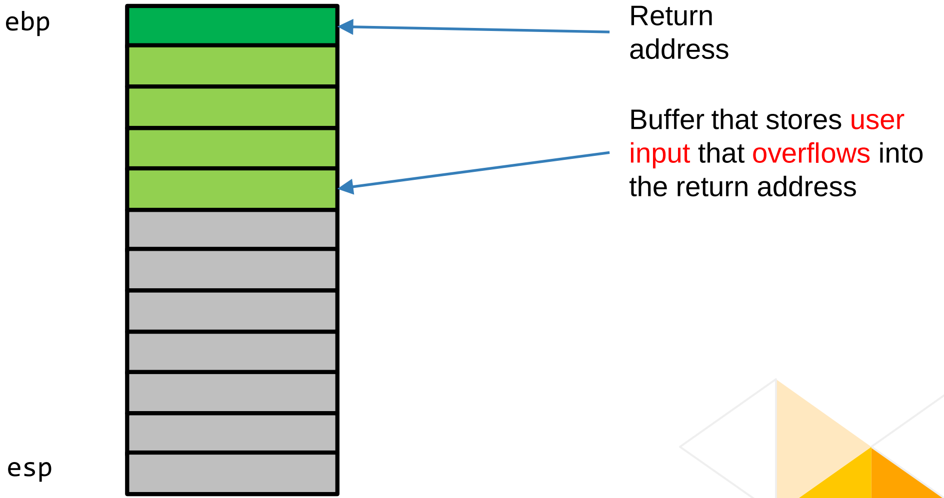
Challenge 5:

EIP Control!

When is something exploitable?

- Control flow?
- How do we know we have control over the control flow?

To determine if we can find a buffer overflow that would lead to an arbitrary jump, we could search for a situation where the return address of a function is unconstrained:



But there’s an easier way…

Search for a situation where the instruction pointer (ip) is symbolic:

`eip`

Commercial Break

Chill for 15 minutes!


CYBER
CYBER

Automatic Exploit Generation

Theory

- From previously:
  - Full EIP control (unconstrained!)
- What we need:
  - Another unconstrained buffer

Algorithm

- Look for all unconstrained symbolic bitvectors
  - Add constraints such that
    - EIP points to buffer
    - Buffer is shellcode
  - If satisfiable:
    - Print concretized values

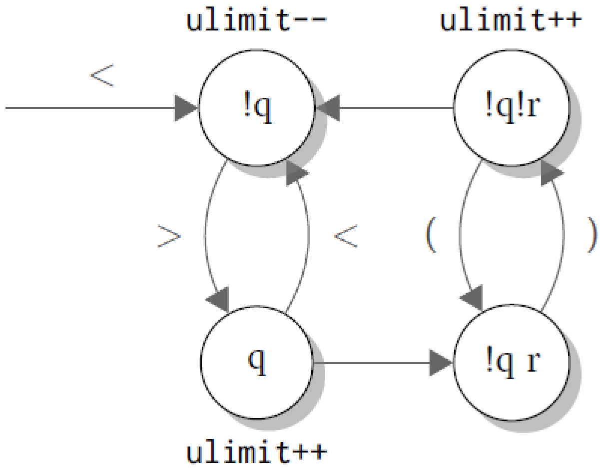
For More Information

https://github.com/ChrisTheCoolHut/Zeratool/

Bad News

State Space Explosion

```c
#define BUFFERSIZE 200
#define TRUE 1
#define FALSE 0
int copy_it (char *input, unsigned int length) {
    char c, localbuf[BUFFERSIZE];
    unsigned int upperlimit = BUFFERSIZE - 10;
    unsigned int quotation = roundquote = FALSE;
    unsigned int inputIndex = outputIndex = 0;
    while (inputIndex < length) {
        c = input[inputIndex++];
        if ((c == '<') && (!quotation)) {
            quotation = TRUE; upperlimit--;
        }
        if ((c == '>') && (quotation)) {
            quotation = FALSE; upperlimit++;
        }
        if ((c == '(') && (!quotation) && !roundquote) {
            roundquote = TRUE; upperlimit--; // decrementation was missing in bug
        }
        if ((c == ')') && (!quotation) && roundquote) {
            roundquote = FALSE; upperlimit++;
        }
        // If there is sufficient space in the buffer, write the character.
        if (outputIndex < upperlimit) {
            localbuf[outputIndex] = c;
            outputIndex++;
        }
    }
    if (roundquote) {
        localbuf[outputIndex] = ')'; outputIndex++; }
    if (quotation) {
        localbuf[outputIndex] = '>'; outputIndex++; }
}
```



- **201** loop iterations to trigger bug
- **10** different paths
- $5^{201} \approx 2^{664}$ paths to prove the bug
- Naively testing for absence of the bug would mean we need to test all possible input strings

SAT Is NP-Hard

- Even if you have the constraints, solving for the variables is NP-Hard
  - If we have **too many symbolic variables/constraints**, computationally too slow!

The Real World™ is hard

- Files
- Static Libraries
- MANY MANY unemulated functions

Understanding Tradeoffs

Soundness vs Completeness

Recent Developments

Symbolic Execution in the Real World™

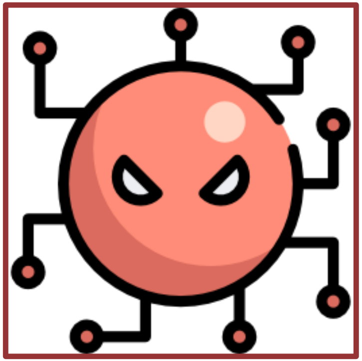
Verifying Program Correctness and Speed

- Intellitest (from Microsoft) uses concolic execution to generate inputs to test programs
- https://docs.microsoft.com/en-us/visualstudio/test/intellitest-manual/input-generation

Automatic Exploit Generation

Malware Analysis

- Explore all possible paths of the malware



Plugin coming soon!

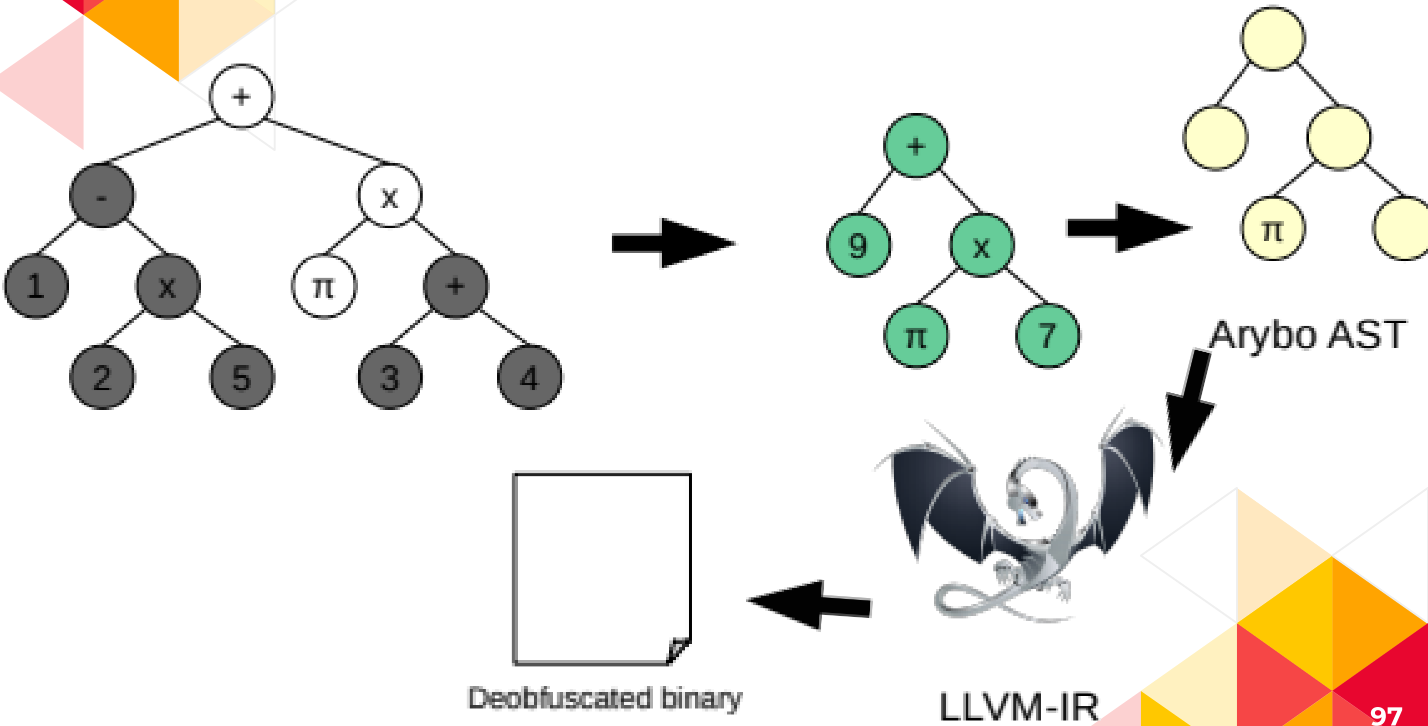
Deobfuscating VM-based binaries

TRITON

Dynamic Binary Analysis

Triton is a dynamic binary analysis (DBA) framework. It provides internal components like a Dynamic Symbolic Execution (DSE) engine, a Taint Engine, AST representations of the x86 and the x86-64 instructions set semantics, SMT simplification passes, an SMT Solver Interface and, the last but not least, Python bindings. Based on these components, you are able to build program analysis tools, automate reverse engineering and perform software verification.

High Level Idea



Conclusion

If you need help...

- Drop me an email!
  - wongwaituck (at) gmail.com

- Join the angr slack
  - http://angr.io/invite/

Finito