Advanced Topics in Malware Analysis
Symbolic Execution

Brendan Saltaformaggio, PhD
Assistant Professor
School of Electrical and Computer Engineering

Symbolic Execution: Introduction

Georgia Tech

1

Learning Objectives

- Employ **symbolic execution** to create constraints
- Evaluate **reasons** for **state explosion**
- Utilize **concolic executions** to simplify complex symbolic expressions

Georgia Tech

2

What is Symbolic Execution?

Symbolic execution (also symbolic analysis) is a means of analyzing a program to determine what inputs cause each part of a program to execute.

Similar to slicing, but instead of concrete values we will use constraints

3

What is Symbolic Execution?

- Symbolic analysis steps through a program (for us: a binary) and performs abstract interpretation of the statements (for us: instructions)
  - Data values are turned into formulas based on how they are used in the program
  - Each execution path is represented via constraints (formulas on the accessed data)
- An interpreter follows the program, assuming symbolic values for inputs rather than obtaining actual inputs as normal execution of the program would
- It builds expressions in terms of those symbols for expressions and variables in the program, and generates constraints in terms of those symbols for the possible outcomes of each conditional branch.
- Excellent Read:
  - Cadar, C., Dunbar, D. and Engler, D. (2008) KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of OSDI'08.*
  - See Also: https://en.wikipedia.org/wiki/Symbolic_execution

4

But… Why?

- Perform analysis of programs WITHOUT inputs
  - Execute a program on symbolic inputs
  - Build constraints on the input values that drive the execution to each path
  - For example, constraints could say “Variable x must be 5 to execute this path”
  - Key Insight: Code can generate its own test cases
- Security Applications:
  - Malware analysis without input and get full code coverage (maybe)!
  - Vulnerability finding --- constraint says “Input ‘ABC’ will cause buffer to overflow”
  - Exploit generation --- constraint says “Input ‘ABC’ will cause RIP to jump”

- King, J. C. (1976). Symbolic execution and program testing. *Communications of the ACM*, *19*(7), 385–394.
- Avgerinos, T., Cha, S. K., Rebert, A., Schwartz, E. J., Woo, M., & Brumley, D. (2014). Automatic exploit generation. *Communications of the ACM*, *57*(2), 74–84

5

Advanced Topics in Malware Analysis

Symbolic Execution

Brendan Saltaformaggio, PhD

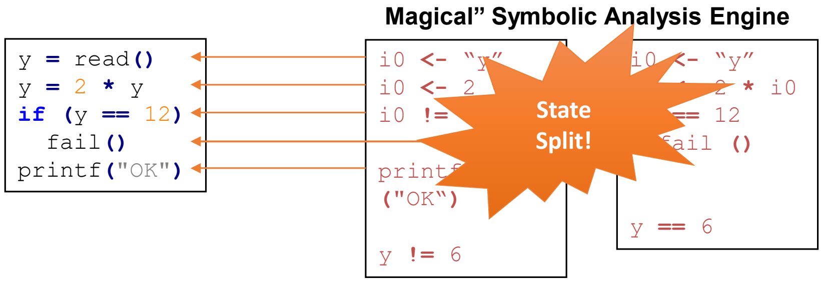
Assistant Professor

School of Electrical and Computer Engineering

Symbolic Execution: Example

6

Symbolic Execution Example



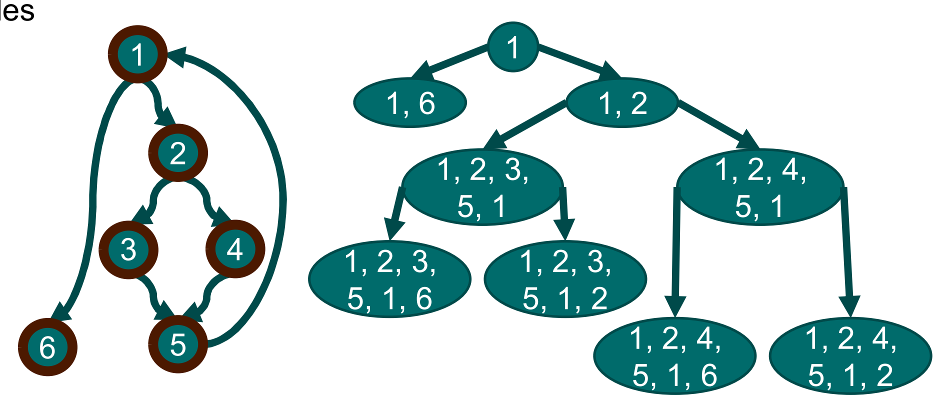
- What input will cause the program to execute the fail function?
- The symbolic analysis engine will step through the program and create constraints on the values
- Constraints can then be queried to answer questions

Georgia Tech

7

State Explosion

- At each decision point the number of tracked states (possible program paths) doubles



Georgia Tech

8

Example: Vortex Wargame

```
#include <...>
void print(unsigned char *buf, int len); // print state (for debugging)

#define e(); if(((unsigned int)ptr & 0xff000000)==0xca000000) { win(); }

int main() {
  unsigned char buf[512];
  unsigned char *ptr = buf + (sizeof(buf)/2);
  unsigned int x;

  while((x = getchar()) != EOF) {
    switch(x) {
      case '\n': print(buf, sizeof(buf)); continue; break;
      case '\\': ptr--; break;
      default: e(); if(ptr > buf + sizeof(buf)) continue; ptr++[0] = x;
} } }
```

Source: http://www.overthewire.org/wargames/

Georgia Tech

9

Example: Vortex Wargame

```
#include <...>
void print(unsigned char *buf, int len); // print state (for debugging)

#define e(); if(((unsigned int)ptr & 0xff000000)==0xca000000) { win(); }

int main() {
  unsigned char buf[512];
  unsigned char *ptr = buf + (sizeof(buf)/2);
  unsigned int x;

  while((x = getchar()) != EOF) {
    switch(x) {
      case '\n': print(buf, sizeof(buf)); continue; break;
      case '\\': ptr--; break;
      default: e(); if(ptr > buf + sizeof(buf)) continue; ptr++[0] = x;
} } }
```

NK2

Source: http://www.overthewire.org/wargames/

Georgia Tech

10

Slide 10

NK2 I added boxes to make the code more noticeable in addition to color but it is up to you if you want to keep it.

Nil Korkmaz, 1/27/2020

Example: Vortex Wargame

```
#include <...>
void print(unsigned char *buf, int len); // print state (for debugging)

#define e(); if(((unsigned int)ptr & 0xff000000)==0xca000000) { win(); }

int main() {
  unsigned char buf[512];
  unsigned char *ptr = buf + (sizeof(buf)/2);
  unsigned int x;

  while((x = getchar()) != EOF) {
    switch(x) {
      case '\n': print(buf, sizeof(buf)); continue; break;
      case '\\': ptr--; break;
      default: e(); if(ptr > buf + sizeof(buf)) continue; ptr++[0] = x;
} } }
```

Source: http://www.overthewire.org/wargames/

Georgia Tech

11

Example: Vortex Wargame

```
#include <...>
void print(unsigned char *buf, int len); // print state (for debugging)

#define e(); if(((unsigned int)ptr & 0xff000000)==0xca000000) { win(); }

int main() {
  unsigned char buf[512];
  unsigned char *ptr = buf + (sizeof(buf)/2);
  unsigned int x;

  while((x = getchar()) != EOF) {
    switch(x) {
      case '\n': print(buf, sizeof(buf)); continue; break;
      case '\\': ptr--; break;
      default: e(); if(ptr > buf + sizeof(buf)) continue; ptr++[0] = x;
} } }
```

Source: http://www.overthewire.org/wargames/

Georgia Tech

12

Example: Vortex Wargame

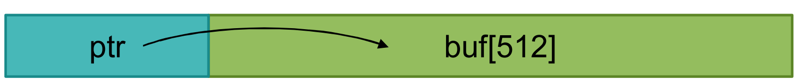
```c
#include <...>
void print(unsigned char *buf, int len); // print state (for debugging)

#define e(); if(((unsigned int)ptr & 0xff000000)==0xca000000) { win(); }

int main() {
  unsigned char buf[512];
  unsigned char *ptr = buf + (sizeof(buf)/2);
  unsigned int x;

  while((x = getchar()) != EOF) {
    switch(x) {
      case '\n': print(buf, sizeof(buf)); continue; break;
      case '\\': ptr--; break;
      default: e(); if(ptr > buf + sizeof(buf)) continue; ptr++[0] = x;
} } }
```

Source: http://www.overthewire.org/wargames/

Georgia Tech

13

Example: Vortex Wargame




```c
switch (input) {          // read char from user
case '\n': debug()        // print debug information
case '\': ptr--;          // decrement ptr
default:
    if (ptr & 0xff000000 == 0xca000000) win();
    if (ptr < buf[len]) ptr++[0] = input;
}
```

Source: http://www.overthewire.org/wargames/

- 3 decision points for each character in the input!
- Problem size: 3^n states being tracked for input size n!!

Georgia Tech

14

Reasons for State Explosion

- Too much input/output data
  - Can we limit symbolic analysis to only the data we care about?
  - <10 symbolic bytes
    - E.g., an address, offset, or pointer
  - 20-80 symbolic bytes
    - E.g., shellcode, ROP chain
  - >200 symbolic bytes
    - E.g., shellcode plus data, long ROP chains, or complete data structures

Georgia Tech

15

Reasons for State Explosion (Cont'd)

- Too much included state
  - Can we limit symbolic state that gets tracked?
  - How to choose what state to prune?
- Too much executed code
  - Tracking every operation inflates the constraints, can we divide and conquer?

Georgia Tech

16

Advanced Topics in Malware Analysis

Symbolic Execution

Brendan Saltaformaggio, PhD

Assistant Professor

School of Electrical and Computer Engineering

Concolic Execution

Georgia Tech

17

Concolic Execution to the Rescue!

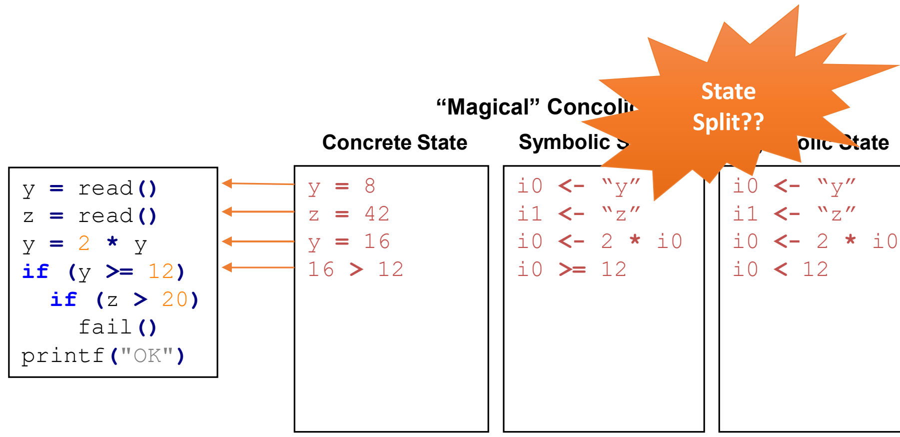
- Combine concrete and symbolic execution:
- Concrete + Symbolic = Concolic
- Use concrete execution with a concrete input to guide symbolic execution
- Selectively replace symbolic variables with concrete values
- Concrete values help simplify complex and unmanageable symbolic expressions

Georgia Tech

18

Concolic Execution Example

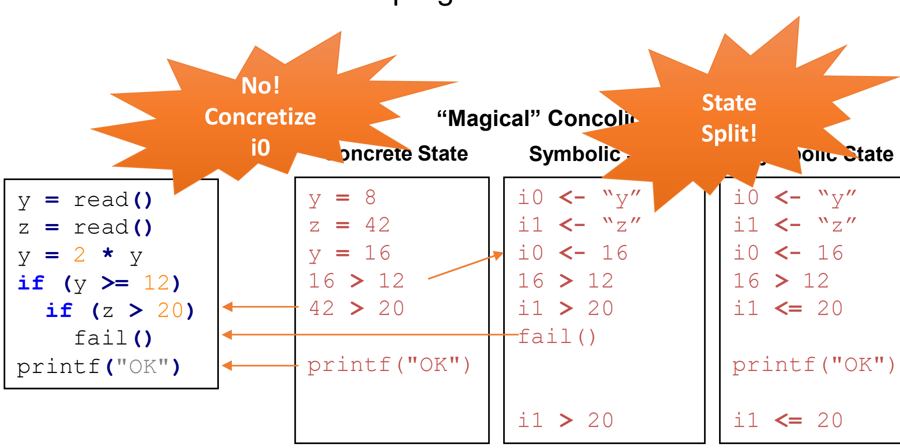
- What value of **z** will cause the program to execute the fail function?



19

Concolic Execution Example

- What value of **z** will cause the program to execute the fail function?



20

Rewind: Carefully Consider Code Coverage!

- What value of **z** will cause the program to execute the fail function?

State Split??

"Magical" Concolic

```
y = read()
z = read()
y = 2 * y
if (y >= 12)
  if (z > 20)
    fail()
printf("OK")
```

Concrete State

```
y = 2
z = 42
y = 4
4 < 12
```

Symbolic

```
i0 <- "y"
i1 <- "z"
i0 <- 2 * i0
i0 >= 12
```

olic State

```
i0 <- "y"
i1 <- "z"
i0 <- 2 * i0
i0 < 12
```

Georgia Tech

21

Rewind: Carefully Consider Code Coverage!

- What value of **z** will cause the program to execute the fail function?

No! Concretize i0

"Magical" Concolic Analysis Engine

```
y = read()
z = read()
y = 2 * y
if (y >= 12)
  if (z > 20)
    fail()
printf("OK")
```

oncrete State

```
y = 2
z = 42
y = 4
4 < 12


printf("OK")
```

Symbolic State

```
i0 <- "y"
i1 <- "z"
i0 <- 4
4 < 12


printf("OK")

i1 = ????
```

Georgia Tech

22

Advanced Topics in Malware Analysis
Symbolic Execution

Brendan Saltaformaggio, PhD
Assistant Professor
School of Electrical and Computer Engineering

Symbolic Execution Tools

Georgia Tech

23

Symbolic Execution Tools

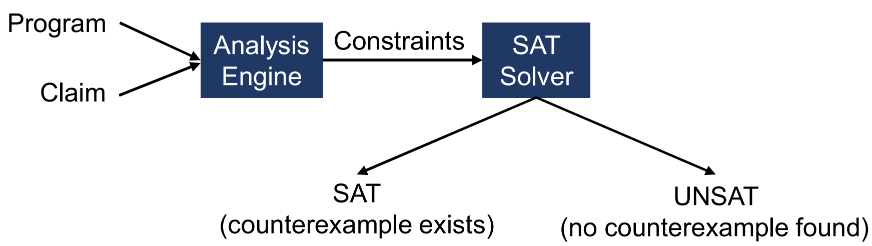
- **Angr**
  - Solving CTF problems, exploit generation, etc.
  - We use Angr in my lab often
  - http://angr.io/
- **S2E: Selective Symbolic Execution**
  - Automatic testing of binary code
  - http://dslab.epfl.ch/proj/s2e
- **Triton**
  - Leverages execution traces from many platforms
  - https://triton.quarkslab.com
- **KLEE**
  - Bug finding in source code
  - KLEE was very widely used until Angr
  - http://ccadar.github.io/klee/
- **FuzzBALL**
  - PoC exploits for given vulnerability conditions
  - http://bitblaze.cs.berkeley.edu/fuzzball.html
- Many more!

Georgia Tech

24

Enabling Technique: SAT Solving

- In computational complexity theory, **satisfiability** (SAT) is the problem of determining if the variables of a given Boolean formula can be assigned in such a way as to make the formula evaluate to TRUE
- **Main Idea:** Given a program and a claim, use a SAT Solver to find whether there exists an execution that violates the claim (a counterexample)
  - First problem shown to be NP-complete (1971)



25

SAT Solving in Symbolic Execution

- For each path:
  1. Convert the set of constraints into a Boolean formula (by ANDing all the variable constraints together)
  2. Check path condition satisfiability (SAT Problem), explore only feasible paths
  3. When execution path diverges, fork, adding constraints on symbolic values
  4. When we terminate (or crash), use a constraint solver to generate concrete input

- All symbolic analysis engines put time-limits on constraint solving
  - Often 10 hours...

26

SAT Solving Example

Program

```
int x;
int y=8,z=0,w=0;
if (x)
  z = y - 1;
else
  w = y + 1;
assert (z == 7 || w == 9)
```

Can the assert statement fail?

Constraints

```
y = 8 &&
z = x ? y - 1 : 0 &&
w = x ? 0 :y + 1 &&
z != 7 &&
w != 9
```

SAT

No counterexample
assertion always holds!

27

SAT Solving Example 2

Program

```
int x;
int y=8,z=0,w=0;
if (x)
  z = y - 1;
else
  w = y + 1;
assert (z == 5 || w == 9)
```

Can the assert statement fail?

Constraints

```
y = 8 &&
z = x ? y - 1 : 0 &&
w = x ? 0 :y + 1 &&
z != 5 &&
w != 9
```

UNSAT

counterexample:

```
y = 8, x = 1, w = 0, z = 7
```

28

An Example in z3

- SAT Solving has become very common in many applications
- Z3 is a high-performance theorem prover developed at Microsoft Research
- Z3 supports real and integer arithmetic, fixed-size bit-vectors, extensional arrays, uninterpreted functions, and quantifiers

```
#!/usr/bin/env python
# Copyri
from z3

x = Rea
y = Rea
s = Sol
s.add(x + y > 5, x > 1, y > 1)
print(s.check())
print(s.model())

m = s.model()
for d in m.decls():
  print "%s = %s" % (d.name(),
m[d])
```

```
~/z3/examples/python$ ./example.py
sat
[y = 4, x = 2]
y = 4
x = 2
```

29

Mini Symbolic Execution Engine In Z3

- Z3 can find the input that would trigger a crash
- Fork at every predicate and explore each side

```
~/z3/examples/python$ ./example.py
Fork - Line 6
[3216] assume (2*x == y)
[3217] assume ¬(2*x == y)
[3217] exit
Fork - Line 7
[3216] assume (y == x + 10)
[3218] assume ¬(y == x + 10)
[3218] exit
[3216] Traceback (most recent call last):
  File "./test_me.py", line 11, in <module>
    test_me(x, y)
  File "./test_me.py", line 7, in test_me
    assert False
AssertionError: x = 10, y = 20
[3216] exit
```

```
1.  #!/usr/bin/env python
2.  from mc import *
3.
4.  def test_me(x, y):
5.    z = 2 * x
6.    if z == y:
7.      if y == x + 10:
8.        assert False
9.
10. x = BitVec("x", 32)
11. y = BitVec("y", 32)
12. test_me(x, y)
```

Source: http://kqueue.org/blog/2015/05/26/mini-mc/

30

Lesson Summary

- Employ Symbolic Execution to create constraints
- Evaluate reasons for state explosion
- Utilize concolic executions to simplify complex symbolic expressions