THE SURVEY OF SYMBOLIC EXECUTION IN SOFTWARE SECURITY

GW Yang, Maple Hu

Introduction

A motivated example

Challenges in symbolic execution

Implement symbolic execution in software security

A short discussion

Overview

Introduction

Testing

- Fuzzing

Symbolic Execution

- Static symbolic execution
- Dynamic/concolic symbolic execution

Some Known Frameworks

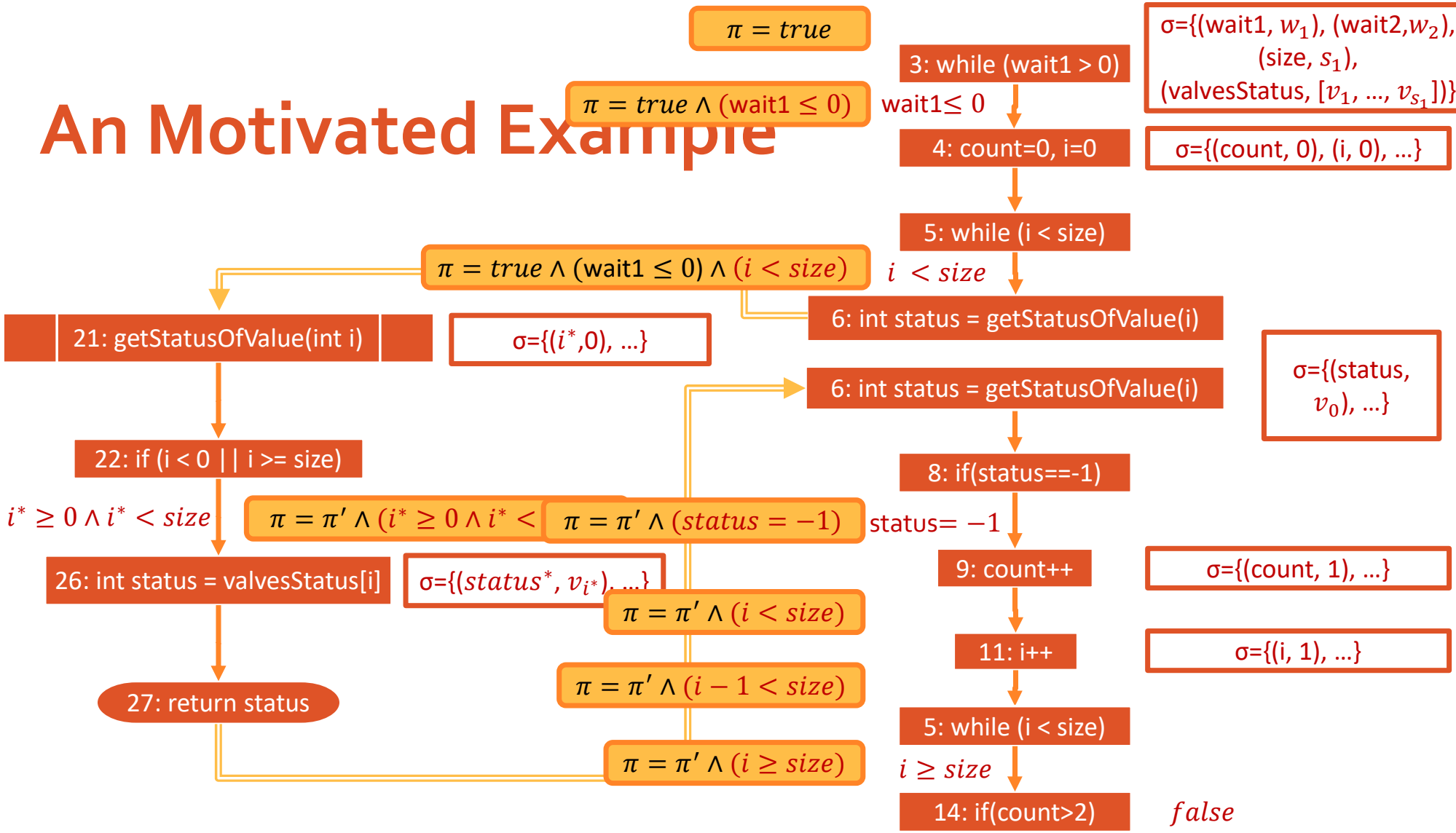
- Klee
- S^2E
- Angr

An Motivated Example

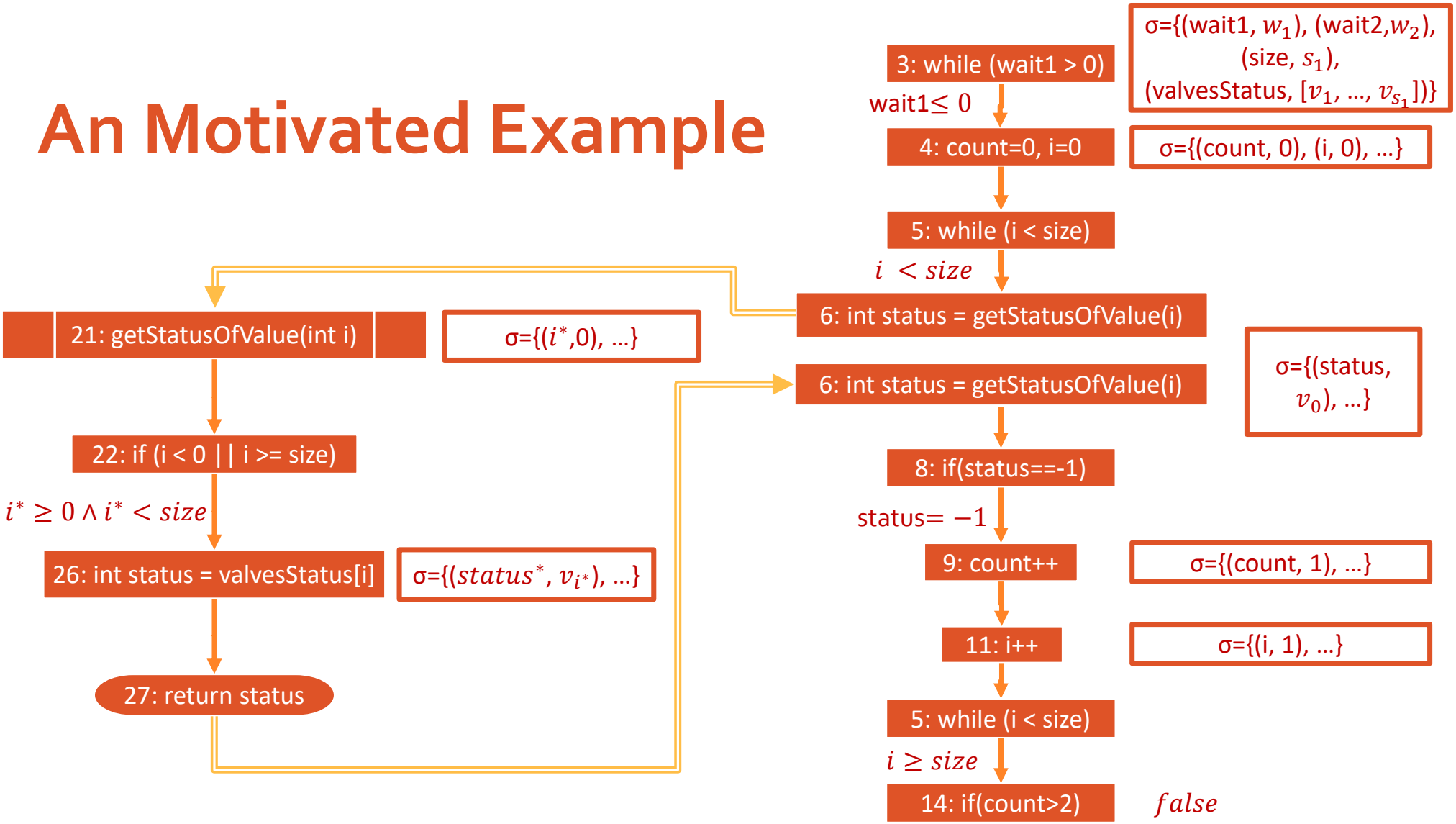
```
1   int checkValves(int wait1, int wait2) {
2       int count, i;
3       while(wait1 > 0) wait1--;
4       count = 0; i = 0;
5       while (i < size) {
6           int status = getStatusOfValue(i);
7
8           if(status == -1) {
9               count++;
10          }
11          i++;
12      }
13
14      if (count > 2) {
15          printf ("ALARM");
16      }
17      while(wait2 > 0) wait2--;
18      return count;
19  }
```

```
21  int getStatusOfValue(int i) {
22      if (i < 0 || i >= size) {
23          printf ("ERROR");
24          exit(EXIT_FAILURE);
25      }
26      int status = valvesStatus[i];
27      return status;
28  }
```

An Motivated Example



An Motivated Example



Challenges

Path Explosion

- scalability
- accuracy

Memory Model

- linear address space, fixed concrete size
- forking model

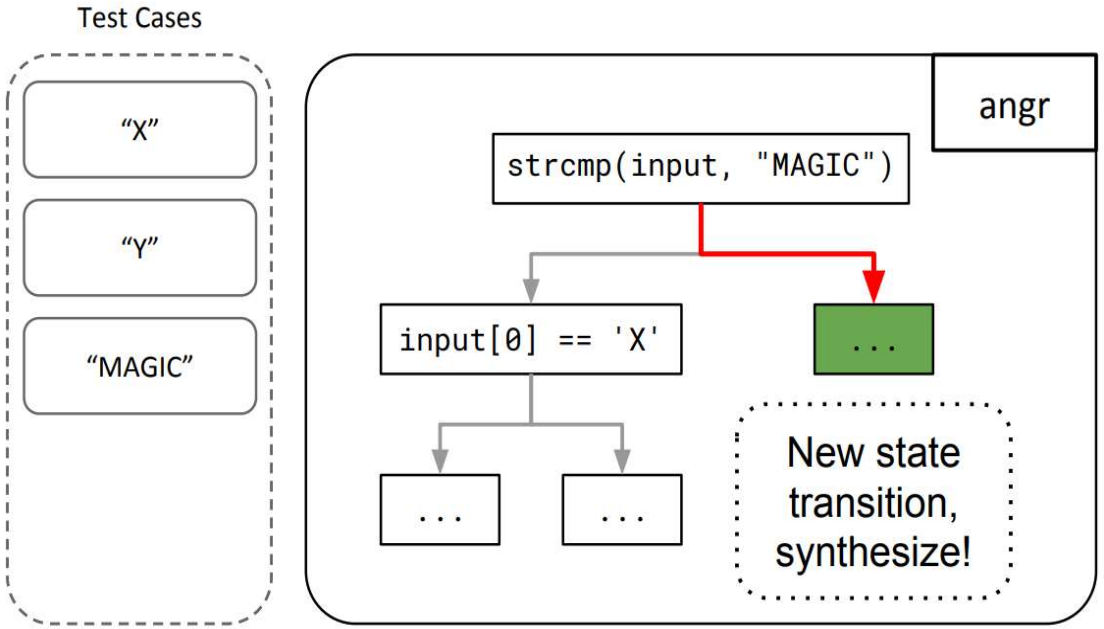
Constraint Solving

- SMT: NP-hard problem

Path Explosion

- pruning to avoid exploring parts of the program that have no effect on the result
- cooperate with other analysis, symbolic execute only in necessary part

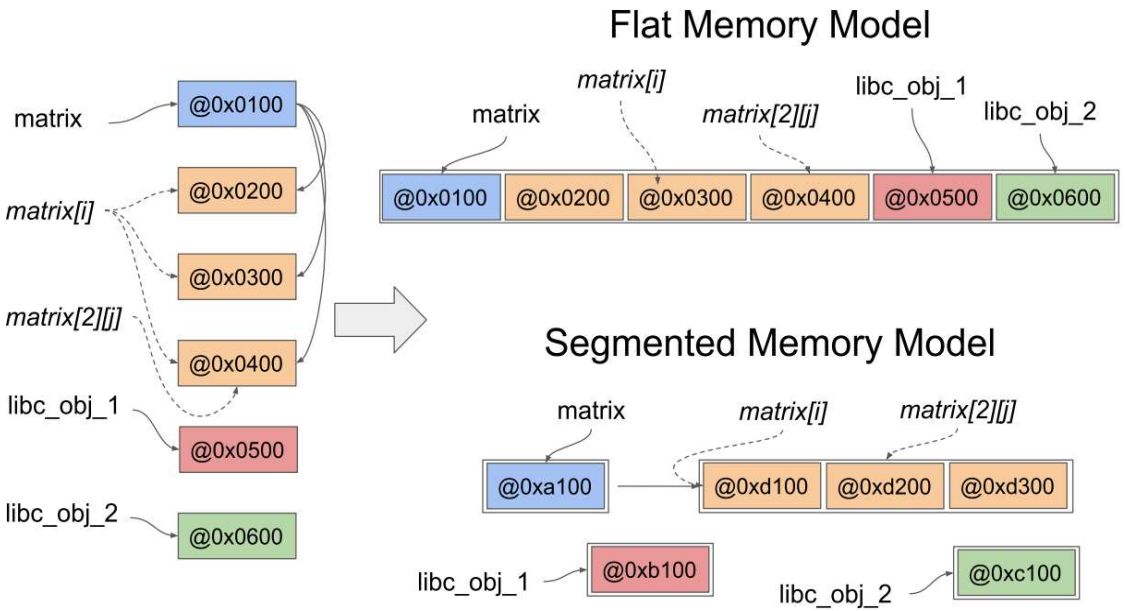
Driller



Memory Model

- linear address space with fixed concrete size:
  - represent the size of memory objects with symbols
- forking model:
  - Segment-based memory model

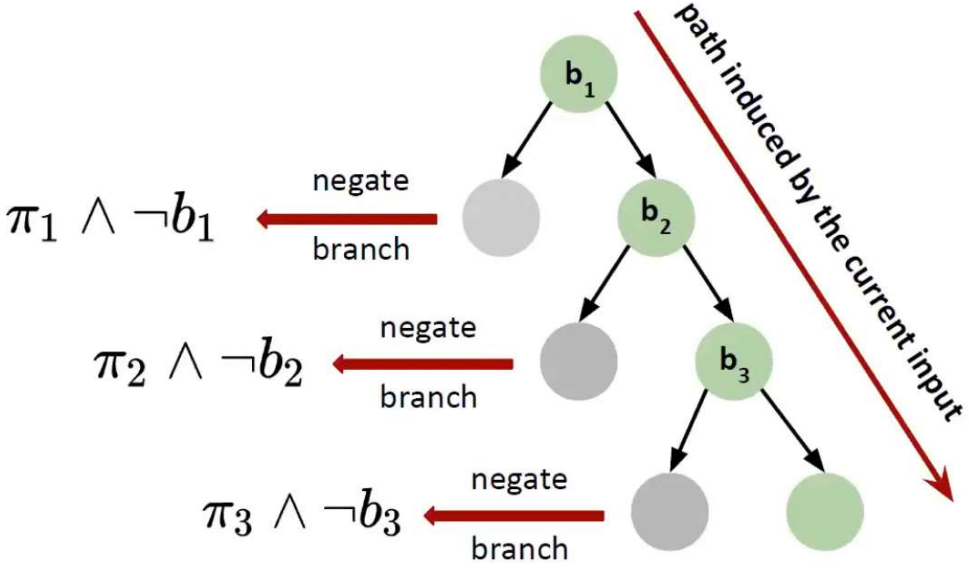
Segmented Memory Model



Constraint Solving

- modify constraints according to the properties of the current program
- use some dynamic tricks to bypass the constraints solving procedure

Fuzzing symbolic expressions



Implementation

Testing and Vulnerability detection

Reproduction/Execution synthesis

Automatic vulnerability exploitation (AEG)

Malware detection

Testing and Vulnerability detection

```
1   if(x * x * x > 0){
2       if(x > 0 && y == 0){
3           // bug
4       }
5   }
```

- Concolic execution
  - Assertions
  - Directed testing
- Identifying symbolic variables
  - Tainted analysis
- The natural complement of SE is Fuzzing.
- Prefect symbolic execution is impossible.

Reproduction/Execution synthesis

- Crash reports
  - crash points
  - call stacks
  - call sequences
  - complete execution traces
- SE guided by crash reports
- Trade-off in trace information collection
  - performance
  - user privacy

```
1  void process(char* source, char* dest) {
2      int out = 0;
3      int in = 0;
4      int srcLength = strlen(source);
5      while(in < srcLength){
6          if (source[in] >= 'a' && source[in] <= 'z'){
7              dest[out] = uppercase(source[in]);
8          }else if(source[in] == '\\'){
9              dest[out] = replaceescape(source[in+1]);
10             out++;
11         }else{
12             dest[out] = source[in];
13         }
14         out++; in++;
15     }
16     printresult(dest, out);
17 }
18 int main(int argc, char* argv[]) {
19     if(argc != 2)
20         exit(0);
21     if(strlen(argv[1]) >= 256)
22         exit(0);
23     char* outputstr = malloc(256);
24     process(argv[1], outputstr);
25 }
```

Automatic vulnerability exploitation (AEG)

- Vulnerable or exploitable?
- End to end analysis
  - Source code analysis
  - Binary analysis
- Realworld is more complex
  - Security mitigations
    DEP, ASLR
  - Host environment
  - Even hard by hand

1. provide a input that can reach the vulnerable point
2. Run instrumented binary with the input, record the program trace and memory layout.
3. exploit it and verify it.

Malware detection

Trigger-based behaviors

Sandbox is not sufficient

Discrete inputs

- system time
- local files
- networking
- ...

SE is an assistant.

Pros:

- Symbolic execution is realistic in realworld
- Symbolic execution is the most direct method

Cons:

- Most of symbolic semantics have to be hand drafted
- Most of symbolic execution engines for specific domains are not available.

A short discussion

ANY QUESTIONS?