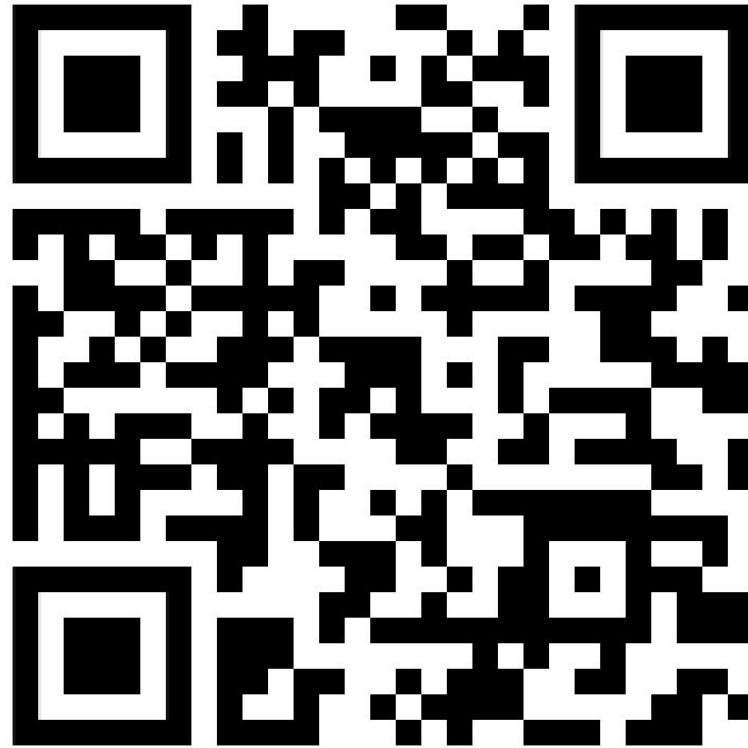


Debugging With Symflower

Using Unit Tests to Find and Fix Bugs





debugging.symflower.com

”

“Program testing can be used to show the presence of bugs, but never to show their absence.”

Dijkstra

Dijkstra might be right, but...



Agenda

1. What is Symflower?
2. Workshop
3. Discussion and Questions



1

What is Symflower?

History Excursion

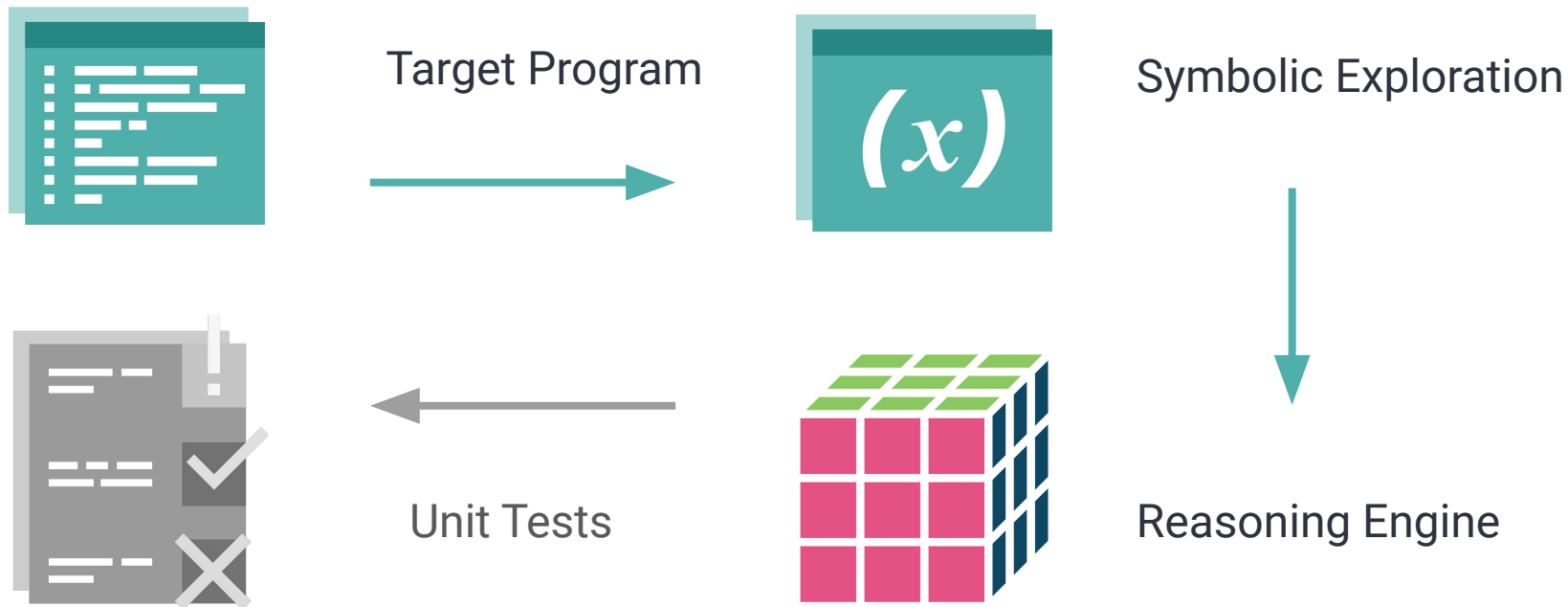


“Software Development Productivity Tool”

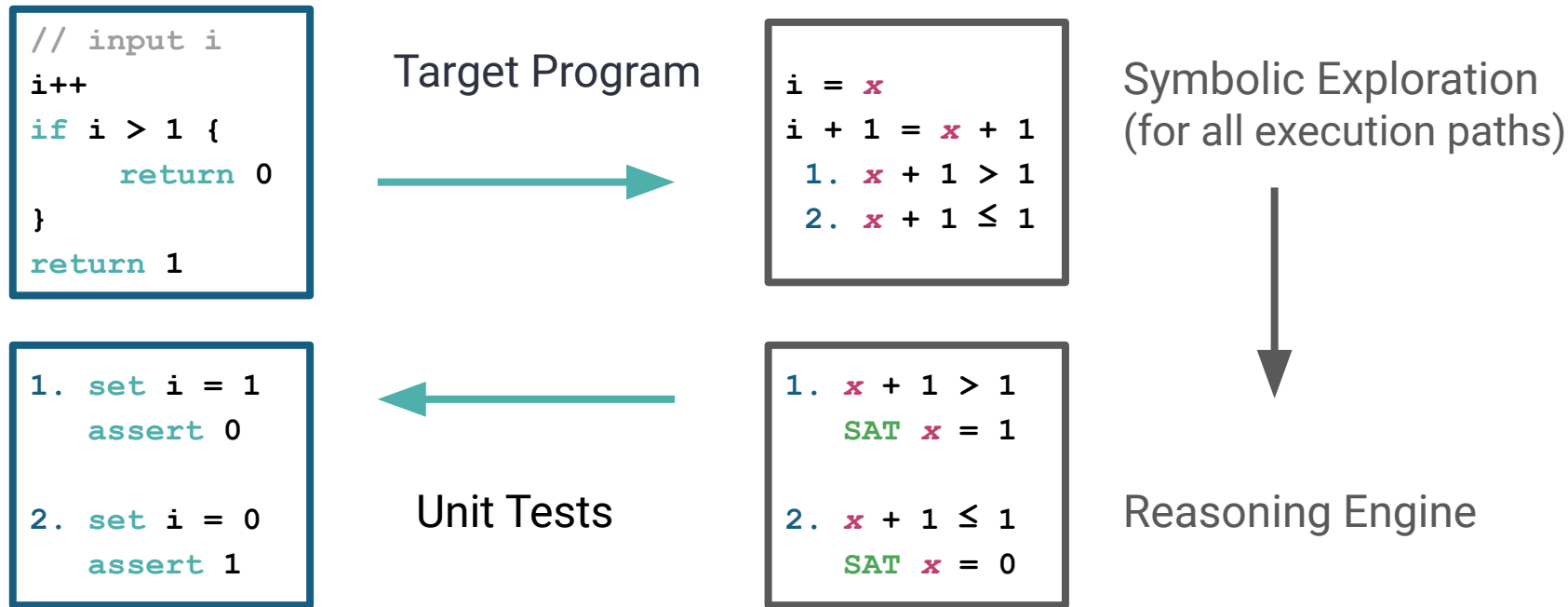
Generate Unit Tests to find Bugs + Security Issues



Symbolic Execution (Recap)

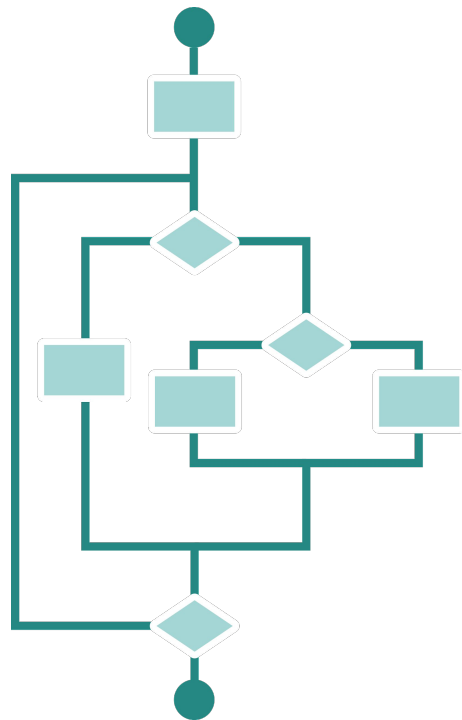


Symbolic Execution (Recap)



- [Overview](#)
- [Symbolic Execution](#)
- [Other Methods](#)

[Original Paper](#) by James C. King



JKU (Linz) Institute for Formal Models and Verification



Former Students / Founders

- DI Evelyn Haslinger, BSc
- DI Markus Zimmermann, BSc
- Simon Bauer, BSc

Former Lecturers

- Univ.-Prof.ⁱⁿ Dr.ⁱⁿ Martina Seidl
- Univ.-Prof. Dr. Armin Biere

The Company

Smart Unit Test
templates, automated
test case generation,
problem detection,
assisted debugging,
high code coverage,
real-time feedback...



- founded in 2018
- 10x employees
- several awards

symflower.com



2 Workshop

Debugging with Unit Tests



Follow along yourself!

go to get.symflower.com



FREE for Linux, macOS and Windows



$$h : X \rightarrow Y$$

Security, Cryptography, Compression, Efficient Algorithms
Checksums, Error Correction, Fast Databases

Our Hash Function

```
func BabyHash(in int) int {  
    div := 3  
    for i := 0; i < 3; i++ {  
        in = in / div  
        div = div + in  
    }  
    return div  
}
```


Examples



The Story

→ GitHub or r/cybersecurity



Bug Report

```
func BabyHash(in int) int {  
    div := 3  
    for i := 0; i < 3; i++ {  
        in = in / div  
        div = div + in  
    }  
    return div  
}
```



-17

Crashes

Debugging Plan

1. Isolate Problem
2. Simple **Reproducer**
3. **Debug** Problem
4. **Fix** (Watch for **Regressions**)



Simplified Version of **TRAFFIC** Principle

Generate Unit Tests

symflower

- freeze behavior
- edge cases

```
func TestSymflowerBabyHash1(t *testing.T) {  
    in := -9  
    babyHash(in)  
}  
  
func TestSymflowerBabyHash2(t *testing.T) {  
    in := 0  
    actual := babyHash(in)  
    expected := 3  
    assert.Equal(t, expected, actual)  
}
```

```
func BabyHash(in int) int {  
    div := 3  
    for i := 0; i < 3; i++ {  
        in = in / div  
        div = div + in  
    }  
    return div  
}
```

$in = -9$

$in_0 = -9/3 = -3$

$div_0 = 3-3 = 0$



Division by Zero

Debugging -17

```
func BabyHash(in int) int {  
    div := 3  
    for i := 0; i < 3; i++ {  
        in = in / div  
        div = div + in  
    }  
    return div  
}
```

$$\text{in} = -17$$

$$\text{in}_0 = -17/3 = -5$$

$$\text{div}_0 = 3-5 = -2$$

$$\text{in}_1 = -5/-2 = 2$$

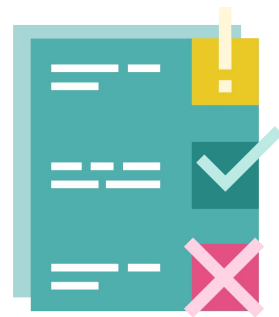
$$\text{div}_1 = -2+2 = 0$$

Potential Fix

```
func BabyHash(in int) int {  
    div := 3  
    for i := 0; i < 3; i++ {  
        in = div / in  
        div = div + in  
    }  
    return div  
}
```

div = 0 ✓

in = 0 ✗



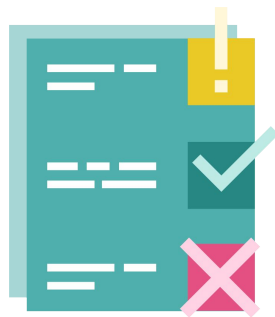
Tests catch Regression

```
func TestSymflowerBabyHash1(t *testing.T) {  
    in := -9  
    babyHash(in)  
}
```

```
func TestSymflowerBabyHash2(t *testing.T) {  
    in := 0  
    actual := babyHash(in)  
    expected := 3  
    assert.Equal(t, expected, actual)  
}
```

div = 0 ✓

in = 0 ✗



Correct Fix

```
func BabyHash(in int) int {  
    div := 3  
    for i := 0; i < 3; i++ {  
        if (div == 0) {  
            div = 3  
        }  
        in = in / div  
        div = div + in  
    }  
    return div  
}
```

div = 0 ✓

in = 0 ✓



Generate Unit Tests

symflower

- `div = 0` ✓
- one iteration

```
func TestSymflowerBabyHash1(t *testing.T) {  
    in := -15  
    actual := babyHash(in)  
    expected := 3  
    assert.Equal(t, expected, actual)  
}  
  
func TestSymflowerBabyHash2(t *testing.T) {  
    in := 0  
    actual := babyHash(in)  
    expected := 3  
    assert.Equal(t, expected, actual)  
}
```

WHEN YOU FIXED ONE BUG

**AND THE BUG TRACKER
STILL HAS TEN MORE**

No one writes bug-free code!

⇒ Workflows and tools to **enhance debugging**

Unit Tests give **repeatable, simple checks** and **prevent regressions** (+ catch bugs before production)

3

Discussion



Questions?



hello@symflower.com



twitter.com/symflower

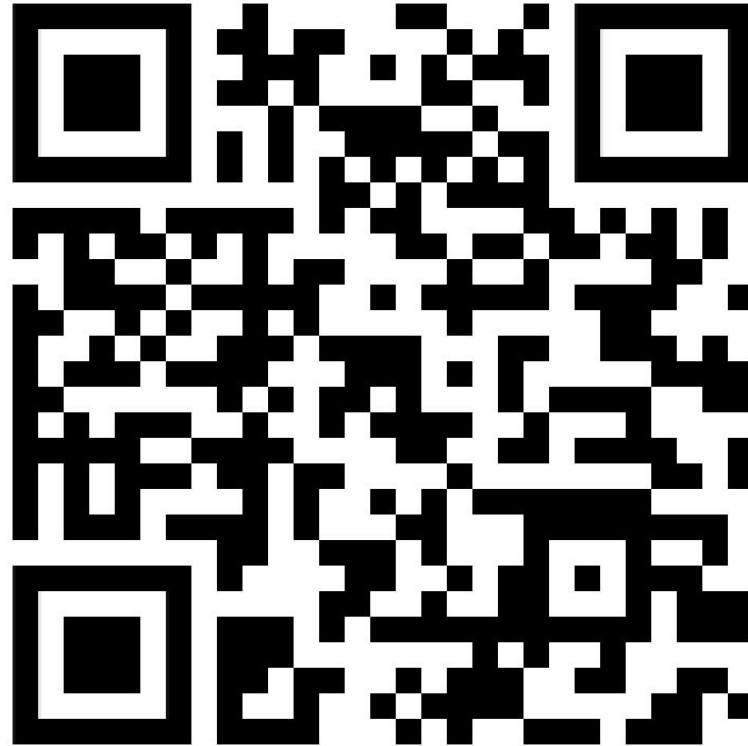


symflower
AUTOMATING QUALITY ASSURANCE



symflower
AUTOMATING QUALITY ASSURANCE

Simon Bauer
simon.bauer@symflower.com



debugging.symflower.com