DeepState: Symbolic Unit Testing for C and C++

by Peter Goodman and Alex Groce

Presented by Rylan O’Connell and Jacques Becker
Agenda

1. Motivation
2. Under the Hood
3. Demo
4. Critiques
5. Questions?
Unit Testing

TEST(Wallet, testBalanceZero) {
    Wallet wallet(0);
    EXPECT_EQ(false, wallet.Withdraw(50));
}

TEST(Wallet, testBalanceNormal) {
    Wallet wallet(100);
    EXPECT_EQ(true, wallet.Withdraw(50));
}

TEST(Wallet, testBalanceBig) {
    Wallet wallet(50);
    wallet.Deposit(4294967296);
    EXPECT_EQ(true, wallet.Withdraw(4294967296));
}
Testing Strategies

Unit Testing

(+): Widespread adoption
(-): Non exhaustive
(+): Easy/fast to write

Binary Analysis

(-): Specialized knowledge
(+): Guaranteed correctness
(-): Expensive to develop

DeepState aims to package powerful binary analysis tools into an accessible framework.
Symbolic Unit Testing

```cpp
#include <deepstate/DeepState.hpp>

TEST(Wallet, testBalance) {
    Symbolic<uint32_t> x;
    ASSUME_NE(x, 0);
    Wallet wallet(50);
    wallet.Deposit(x);
    EXPECT_EQ(true, wallet.Withdraw(x));
}
```

```cpp
class Wallet {
    private:
        uint32_t balance;
    
    public:
        Wallet(uint32_t start) : balance(start) {} 

        void Deposit(uint32_t amount) {
            balance += amount;
        }

        bool Withdraw(uint32_t amount) {
            if (amount <= balance) {
                balance -= amount;
                return true;
            } else {
                return false;
            }
        }

        void CheckBalance() {
            return balance;
        }
};
```
Agenda

1. Motivation
2. Under the Hood
3. Demo
4. Critiques
5. Questions?
Modular Backend Support

angr
• Industry standard for SE
• (Relatively) fast execution
• Core of MechanicalPhish

Manticore
• Just happy to be included

Semi-modular support for existing SE engines
Path Explosion

- Complexity increases **exponentially** with branches
- **Unbounded** state forking on symbolic loop bounds
- Main limiting factor with “real world” programs
OneOf

• In case of large branching statement, randomly selects one branch to execute
• Creates multiple versions of the program, each with some options omitted
• Enables more effective fuzzing
Pumping

Given a loop with symbolic bounds

```
for(int i = 0; i < SYMBOLIC_ARG; ++i)
```

1. Concretize the symbolic argument
2. Take maximum* concrete value
3. Continue as normal
Pumping (Continued)

Pros

(+) Performance boost
(+) Analyzes “real” programs
(+) Novel approach

Cons

(-) Chance of unsound analysis
(-) Must specify behavior
(-) Novel approach
Demo
Agenda

1. Motivation
2. Under the Hood
3. Demo
4. Critiques
5. Questions?
Critiques

(+): Easily approachable
(+): Bridging the gap between academia and industry
(-): Developers are expected to specify pumping behavior*
(-): Never discuss DeepState’s fuzzing backends
(-): Capabilities of OneOf are never fully explored
“In some ways, developers are the adversary of symbolic execution engines: they continually find ways to write hard to analyze code.”

-Goodman & Groce
Am I out of touch?
No, it’s the developers who are wrong.