Intro to Scribble

Scribble Bootcamp

Dimitar Bounov

Plan

DILIGENCE Fuzzing

Plan

Scribble Overview

DILIGENCE Fuzzing

Plan

Scrible Overview
Function Annotations

DILIGENCE Fuzzing

Plan

Scrible Overview
Function Annotations
Contract Invariants

DILIGENCE Fuzzing

Plan

Scrible Overview
Function Annotations
Contract Invariants
Other Features

DILIGENCE Fuzzing

The Problem

The Problem

We expect certain properties of our code to be true.

The Problem

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

We expect certain properties of our code to be true.

DILIGENCE Fuzzing

The Problem

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

We expect certain properties of our code to be true.

“The balance of to increases after the transfer”

DILIGENCE Fuzzing

Option 1: Comments

```
/// Balance of to increases
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

DILIGENCE Fuzzing

Option 1: Comments

```
/// Balance of to increases
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

Can’t test/check comments

DILIGENCE Fuzzing

Option 2: Add Assertions

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
    uint256 oldBalance = balanceOf(to);
    ...
    assert(balanceOf(to) > oldBalance);
}
```

DILIGENCE Fuzzing

Option 2: Add Assertions

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
    uint256 oldBalance = balanceOf(to);
    ...
    assert(balanceOf(to) > oldBalance);
}
```

Increased runtime cost

DILIGENCE Fuzzing

Option 2: Add Assertions

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
    uint256 oldBalance = balanceOf(to);
    ...
    assert(balanceOf(to) > oldBalance);
}
```

Increased runtime cost

Pollute code/compilcate dev workflow

DILIGENCE Fuzzing

Option 2: Add Assertions

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
    uint256 oldBalance = balanceOf(to);
    ...
    assert(balanceOf(to) > oldBalance);
}
```

Increased runtime cost

Pollute code/compilcate dev workflow

Can we get the best of both worlds?

DILIGENCE Fuzzing

What is Scribble?

DILIGENCE Fuzzing

What is Scribble?

Specification Language For Developers

DILIGENCE Fuzzing

What is Scribble?

```
/// #if_succeeds balanceOf(to) >= amount;
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

Specification Language For Developers

DILIGENCE Fuzzing

What is Scribble?

```
/// #if_succeeds balanceOf(to) >= amount;
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

Specification Language For Developers

Annotations embedded in docstrings

DILIGENCE Fuzzing

What is Scribble?

```
/// #if_succeeds balanceOf(to) >= amount;
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

- Specification Language For Developers
- Annotations embedded in docstrings
- Solidity-like expressions

DILIGENCE Fuzzing

What is Scribble?

```
/// #if_succeeds balanceOf(to) >= amount;
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

Specification Language For Developers

Annotations embedded in docstrings

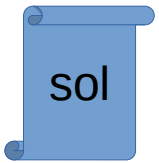
Solidity-like expressions

Can think of them as checkable comments!

DILIGENCE Fuzzing

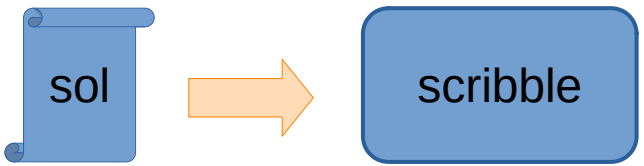
How do we use Scribble?

How do we use Scribble?



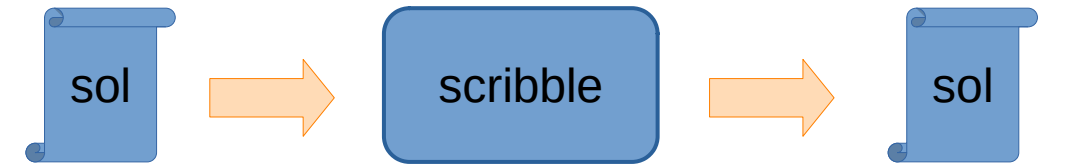
(annotated)

How do we use Scribble?



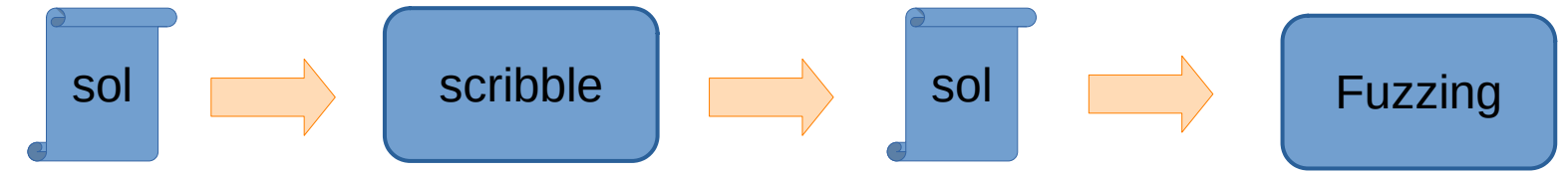
(annotated)

How do we use Scribble?



(annotated) (instrumented)

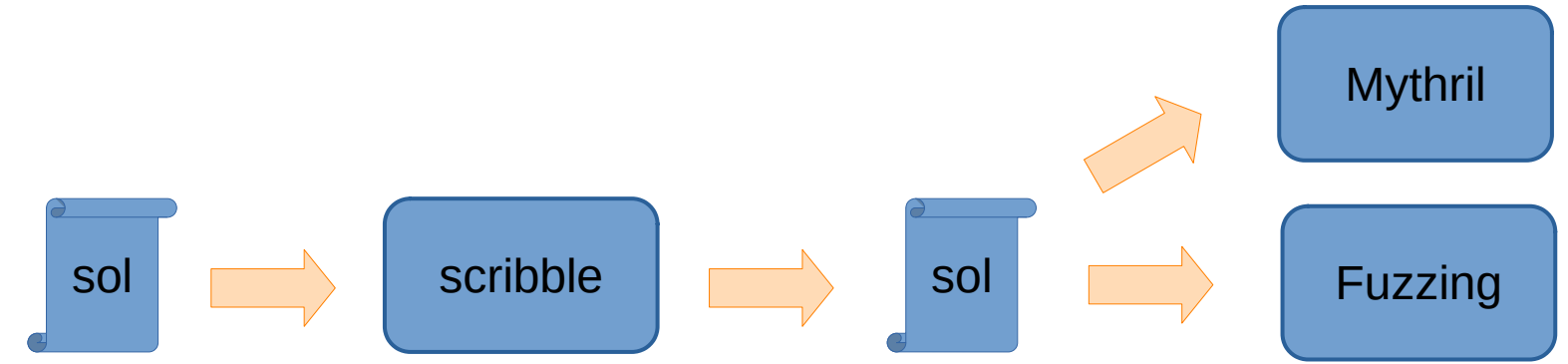
How do we use Scribble?



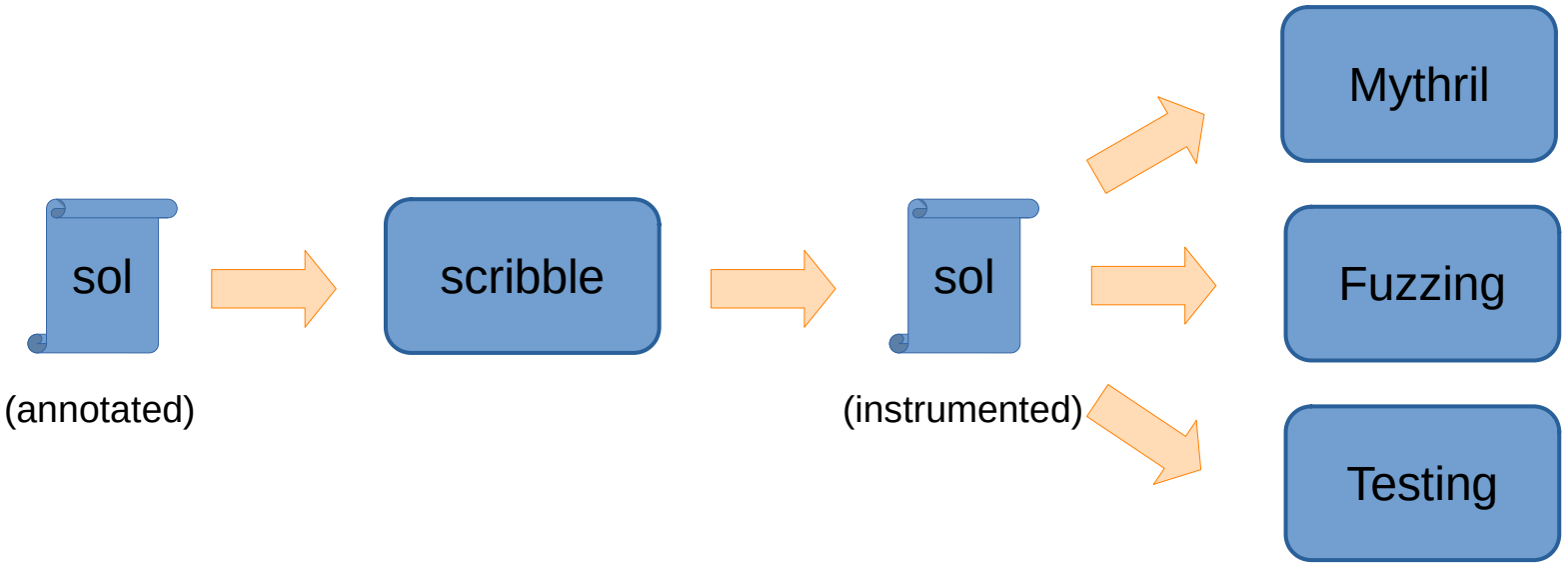
(annotated)

(instrumented)

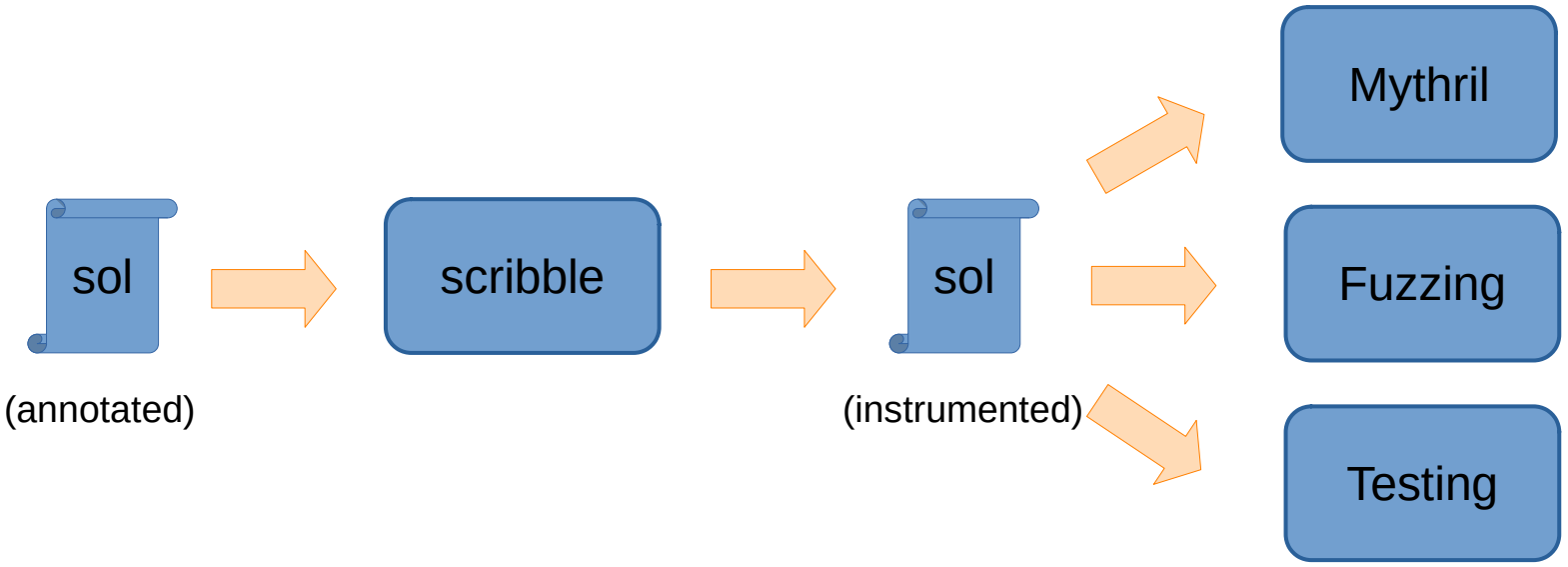
How do we use Scribble?



How do we use Scribble?



How do we use Scribble?



Scribble integrates with many workflows

DILIGENCE Fuzzing

Plan

Scrible Overview
Function Annotations
Contract Invariants
Other Features

DILIGENCE Fuzzing

Function Annotations

Function Annotations

```
/// #if_succeeds balanceOf(to) >= amount;
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

DILIGENCE Fuzzing

Function Annotations

```
/// #if_succeeds balanceOf(to) >= amount;
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

if_succeeds P; checks that P holds upon successful termination of function

DILIGENCE Fuzzing

Function Annotations

```
/// #if_succeeds balanceOf(to) >= amount;
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

if_succeeds P; checks that P holds upon successful termination of function

P can be any valid pure solidity expression

DILIGENCE Fuzzing

Function Annotations

```
/// #if_succeeds balanceOf(to) >= amount;
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

if_succeeds P; checks that P holds upon successful termination of function

P can be any valid pure solidity expression

P can include some scribble extensions

DILIGENCE Fuzzing

Function Annotations

```
/// #if_succeeds balanceOf(to) >= amount;
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

if_succeeds P; checks that P holds upon successful termination of function

P can be any valid pure solidity expression

P can include some scribble extensions

What if we want to say exactly how much to's balance increases?

DILIGENCE Fuzzing

Function Annotations

Function Annotations

```
/// #if_succeeds
///     old(balanceOf(to)) + amount == balanceOf(to);
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

DILIGENCE Fuzzing

Function Annotations

```
/// #if_succeeds
///     old(balanceOf(to)) + amount == balanceOf(to);
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

Use old() to refer to state before call

DILIGENCE Fuzzing

Function Annotations

```
/// #if_succeeds
///     old(balanceOf(to)) + amount == balanceOf(to);
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

Use old() to refer to state before call

In old(E), E must have a value type

DILIGENCE Fuzzing

Function Instrumentation

Function Instrumentation

```
/// #if_succeeds
///     old(balanceOf(to)) + amount == balanceOf(to);
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

DILIGENCE Fuzzing

Function Instrumentation

```
/// #if_succeeds
///     old(balanceOf(to)) + amount == balanceOf(to);
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) virtual override public returns (bool RET_0) {
    vars0 memory _v;
    _v.old_0 = balanceOf(to);

    RET_0 = _original_Foo_transferFrom(from, to, amount);

    if (!((_v.old_0 + amount) == balanceOf(to))) {
        assert(false);
    }
}

function _original_Foo_transferFrom(
    address from,
    address to,
    uint256 amount
) private returns (bool) {
```

DILIGENCE Fuzzing

Function Instrumentation

```
/// #if_succeeds
///     old(balanceOf(to)) + amount == balanceOf(to);
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

Inserted a wrapper function with all the instrumentation

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) virtual override public returns (bool RET_0) {
    vars0 memory _v;
    _v.old_0 = balanceOf(to);

    RET_0 = _original_Foo_transferFrom(from, to, amount);

    if (!((_v.old_0 + amount) == balanceOf(to))) {
        assert(false);
    }
}

function _original_Foo_transferFrom(
    address from,
    address to,
    uint256 amount
) private returns (bool) {
```

DILIGENCE Fuzzing

Function Instrumentation

```
/// #if_succeeds
///     old(balanceOf(to)) + amount == balanceOf(to);
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

Original function renamed and private

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) virtual override public returns (bool RET_0) {
    vars0 memory _v;
    _v.old_0 = balanceOf(to);

    RET_0 = _original_Foo_transferFrom(from, to, amount);

    if (!((_v.old_0 + amount) == balanceOf(to))) {
        assert(false);
    }
}

function _original_Foo_transferFrom(
    address from,
    address to,
    uint256 amount
) private returns (bool) {
```

DILIGENCE Fuzzing

Function Instrumentation

```
/// #if_succeeds
///     old(balanceOf(to)) + amount == balanceOf(to);
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

Original function renamed and private

Invoked in wrapper

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) virtual override public returns (bool RET_0) {
    vars0 memory _v;
    _v.old_0 = balanceOf(to);

    RET_0 = _original_Foo_transferFrom(from, to, amount);

    if (!((_v.old_0 + amount) == balanceOf(to))) {
        assert(false);
    }
}

function _original_Foo_transferFrom(
    address from,
    address to,
    uint256 amount
) private returns (bool) {
```

DILIGENCE Fuzzing

Function Instrumentation

```
/// #if_succeeds
///     old(balanceOf(to)) + amount == balanceOf(to);
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

Old state computed before call

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) virtual override public returns (bool RET_0) {
    vars0 memory _v;
    _v.old_0 = balanceOf(to)

    RET_0 = _original_Foo_transferFrom(from, to, amount);

    if (!((_v.old_0 + amount) == balanceOf(to))) {
        assert(false);
    }
}

function _original_Foo_transferFrom(
    address from,
    address to,
    uint256 amount
) private returns (bool) {
```

DILIGENCE Fuzzing

Function Instrumentation

```
/// #if_succeeds
///     old(balanceOf(to)) + amount == balanceOf(to);
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool) {
```

Old state computed before call

Property checked after call

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) virtual override public returns (bool RET_0) {
    vars0 memory _v;
    _v.old_0 = balanceOf(to)

    RET_0 = _original_Foo_transferFrom(from, to, amount);

    if (!((_v.old_0 + amount) == balanceOf(to))) {
        assert(false);
    }
}

function _original_Foo_transferFrom(
    address from,
    address to,
    uint256 amount
) private returns (bool) {
```

DILIGENCE Fuzzing

Demo

DILIGENCE Fuzzing

Function Instrumentation

Function Instrumentation

if_succeeds also allowed on contracts

DILIGENCE Fuzzing

Function Instrumentation

```
/// #if_succeeds counter > old(counter);
contract Foo {
    uint counter;

    function a() public {
        ...
    }

    function b() public {
        ...
    }
}
```

if_succeeds also allowed on contracts

DILIGENCE Fuzzing

Function Instrumentation

```
/// #if_succeeds counter > old(counter);
contract Foo {
    uint counter;

    function a() public {
        ...
    }

    function b() public {
        ...
    }
}
```

if_succeeds also allowed on contracts

Same as adding the same if_succeeds to every public/external function

Function Instrumentation

```
contract Foo {
    uint counter;
    /// #if_succeeds counter > old(counter);
    function a() public {
        ...
    }
    /// #if_succeeds counter > old(counter);
    function b() public {
        ...
    }
}
```

if_succeeds also allowed on contracts

Same as adding the same if_succeeds to every public/external function

DILIGENCE Fuzzing

Function Instrumentation

```
contract Foo {
    uint counter;
    /// #if_succeeds counter > old(counter);
    function a() public {
        ...
    }
    /// #if_succeeds counter > old(counter);
    function b() public {
        ...
    }
}
```

if_succeeds also allowed on contracts

Same as adding the same if_succeeds to every public/external function

if_succeeds added to subcontract functions too

DILIGENCE Fuzzing

Plan

Scrible Overview
Function Annotations
Contract Invariants
Other Features

DILIGENCE Fuzzing

Contract Invariants

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;
```

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;
```

Specified with #invariant keyword

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;
```

Specified with #invariant keyword

unchecked_sum is a scribble builtin

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;
```

Invariants over the lifetime of a contract

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;
```

Invariants over the lifetime of a contract

Can talk about contract state variables

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;
```

- Invariants over the lifetime of a contract
- Can talk about contract state variables
- Cannot use ‘old’

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;
```

Is this invariant true at **every** point?

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;
```

Is this invariant true at **every** point?

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual returns (bool) {
    ...
    balances[from] -= amount;
    balances[to] += amount;
```

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;
```

Is this invariant true at **every** point?

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual returns (bool) {
    ...
    balances[from] -= amount;
    balances[to] += amount;
```

Invariant temporarily broken here

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;
```

Is this invariant true at **every** point?

So when are these invariants true?

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual returns (bool) {
    ...
    balances[from] -= amount;
    balances[to] += amount;
```

Invariant temporarily broken here

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;
```

Invariants true at “observable points” in contract lifetime

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;
```

Invariants true at “observable points” in contract lifetime

An “observable point” is any point before we enter/after we exit the contract

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;
```

Invariants true at “observable points” in contract lifetime

An “observable point” is any point before we enter/after we exit the contract

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;
```

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;
```

So how do we check invariant holds at all observable points?

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;
```

So how do we check invariant holds at all observable points?

1. Check invariant holds after constructor

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;

    constructor() {
        check_invs();
    }
```

So how do we check invariant holds at all observable points?

1. Check invariant holds after constructor

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;

    constructor() {
        check_invs();
    }
```

So how do we check invariant holds at all observable points?

1. Check invariant holds after constructor

2. Check invariant holds before exiting external call

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;

    constructor() {
        check_invs();
    }

    function transferFrom(
        address from,
        address to,
        uint256 amount
    ) public virtual returns (bool) {



        ...
        check_invs();
        return true;
```

So how do we check invariant holds at all observable points?

1. Check invariant holds after constructor

2. Check invariant holds before exiting external call

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;

    constructor() {
        check_invs();
    }

    function transferFrom(
        address from,
        address to,
        uint256 amount
    ) public virtual returns (bool) {



        ...
        check_invs();
        return true;
```

So how do we check invariant holds at all observable points?

1. Check invariant holds after constructor
2. Check invariant holds before exiting external call
3. Check invariant holds before making an external call

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;

    constructor() {
        check_invs();
    }

    function transferFrom(
        address from,
        address to,
        uint256 amount
    ) public virtual returns (bool) {
        ...
        check_invs();
        someListenerContract.transferCallback();
        ...
        check_invs();
        return true;
```

So how do we check invariant holds at all observable points?

1. Check invariant holds after constructor
2. Check invariant holds before exiting external call
3. Check invariant holds before making an external call

DILIGENCE Fuzzing

Contract Invariants

```
/// #invariant unchecked_sum(balances) == totalSupply;
contract ERC20 {
    mapping(address => uint) balances;
    uint totalSupply;

    constructor() {
        check_invs();
    }

    function transferFrom(
        address from,
        address to,
        uint256 amount
    ) public virtual returns (bool) {
        ...
        check_invs();
        someListenerContract.transferCallback();
        ...
        check_invs();
        return true;
```

So how do we check invariant holds at all observable points?

1. Check invariant holds after constructor
2. Check invariant holds before exiting external call
3. Check invariant holds before making an external call

DILIGENCE Fuzzing

Demo

DILIGENCE Fuzzing

Plan

Scrible Overview
Function Annotations
Contract Invariants
Other Features

DILIGENCE Fuzzing

Let bindings

```
/// #if_succeeds
///     old(balanceOf(from)) > amount &&
///     old(balanceOf(from)) - amount == balanceOf(from);
function transferFrom(
    address from,
    address to,
    uint256 amount
) public returns (bool) {

}
```

DILIGENCE Fuzzing

Let bindings

```
/// #if_succeeds
///     old(balanceOf(from)) > amount &&
///     old(balanceOf(from)) - amount == balanceOf(from);
function transferFrom(
    address from,
    address to,
    uint256 amount
) public returns (bool) {

}
```

Same expression computed twice

DILIGENCE Fuzzing

Let bindings

```
/// #if_succeeds
///     old(balanceOf(from)) > amount &&
///     old(balanceOf(from)) - amount == balanceOf(from);
function transferFrom(
    address from,
    address to,
    uint256 amount
) public returns (bool) {

}
```

Same expression computed twice

Can reduce computation using a let-binding

DILIGENCE Fuzzing

Let bindings

```
/// #if_succeeds
///     let oldBalance := old(balanceOf(from)) in
///         oldBalance > amount &&
///         oldBalance - amount == balanceOf(from);
function transferFrom(
    address from,
    address to,
    uint256 amount
) public returns (bool) {
```

DILIGENCE Fuzzing

Let bindings

```
/// #if_succeeds
///     let oldBalance := old(balanceOf(from)) in
///         oldBalance > amount &&
///         oldBalance - amount == balanceOf(from);
function transferFrom(
    address from,
    address to,
    uint256 amount
) public returns (bool) {
```

DILIGENCE Fuzzing

Let bindings

```
/// #if_succeeds
///     let oldBalance := old(balanceOf(from)) in
///         oldBalance > amount &&
///         oldBalance - amount == balanceOf(from);
function transferFrom(
    address from,
    address to,
    uint256 amount
) public returns (bool) {
```

DILIGENCE Fuzzing

Let bindings

```
/// #if_succeeds
///     let oldBalance := old(balanceOf(from)) in
///         oldBalance > amount &&
///         oldBalance - amount == balanceOf(from);
function transferFrom(
    address from,
    address to,
    uint256 amount
) public returns (bool) {
```

Let bindings are immutable
(not variables)

DILIGENCE Fuzzing

Implication

Implication

You can express implication using ==>

Implication

```
/// #if_succeeds
///     amount > 0 ==> balanceOf(to) > old(balanceOf(to))
function transferFrom(
    address from,
    address to,
    uint256 amount
) public returns (bool) {
```

You can express implication using ==>

DILIGENCE Fuzzing

Summary

Summary

- Scribble overview

DILIGENCE Fuzzing

Summary

- Scribble overview
- Function annotations (#if_succeeds)

Summary

- Scribble overview
- Function annotations (#if_succeeds)
- Contract invariants (#invariant)

Summary

- Scribble overview
- Function annotations (#if_succeeds)
- Contract invariants (#invariant)
- Other language features (sum, let, old, ==>)

Homework

Questions

Course Material:
https://github.com/ConsenSys/secureum-diligence-bootcamp/

Other:
Scribble: https://github.com/consensys/scribble
Scribble Docs: https://docs.scribble.codes/
Discord Channel: #carex-diligence-scribble-nov22
Instructors: dimo#1001, wuestholz#3558

DILIGENCE Fuzzing