Property Checking with Scribble and Mythril

Intro

In this tutorial we will learn how to:

- start using scribble
- add a custom property to an ERC20 token contract
- look for violations of our custom property withmythril
-

Without further ado...

Dependencies

Make sure that you have the following installed:

- python3 and pip3
- nodejs version 12.x and npm.
-

Setup

To get setup you first need to install scribble:

```

Copy npminstall-geth-scribble

```

Next install mythril:

```

Copy pip3install-Umythril

```

Now we are ready to begin!

Background

Before we begin, we should clarify what we mean by 'custom properties'. Custom properties in general can be any predicate you have in mind about your code. For example, a custom property can be:

- Only the current owner of the contract may change theowner
- state variable
- My contract will allow trades only after theinitialize()
- function was called
- The number of minted tokens only increases
- etc..
-

In order to be able to reason about these propertiesformally however, we need to translate their English statements into a formal language that automated tools understand. We will show how to do this using theScribble spec language. Note that the language choice imposes certain limitations on what can be expressed. We are actively working on adding more expressive power to Scribble.

Our example

For this tutorial we will consider a simple ERC20 token, and annotate it with a custom property expressed in the Scribble language. Take the code below and save it asToken.sol in a directory of your choice:

Token.sol

```

Copy pragma solidity ^0.6.0;

contract ERC20 { uint256 private _totalSupply; mapping (address => uint256) private _balances; mapping (address => mapping (address => uint256)) private _allowances;

constructor() public { _totalSupply = 1000000; _balances[msg.sender] = 1000000; }

function totalSupply() external view returns (uint256) { return _totalSupply; }

function balanceOf(address _owner) external view returns (uint256) { return _balances[_owner]; }

function allowance(address _owner, address _spender) external view returns (uint256) { return _allowances[_owner][_spender]; }

function transfer(address _to, uint256 _value) external returns (bool) { address from = msg.sender; require(_value <= _balances[from]);

uint256 newBalanceFrom = _balances[from] - _value; uint256 newBalanceTo = _balances[_to] + _value; _balances[from] = newBalanceFrom; _balances[_to] = newBalanceTo;

emit Transfer(msg.sender, _to, _value); return true; }

function approve(address _spender, uint256 _value) external returns (bool) { address owner = msg.sender; _allowances[owner][_spender] = _value; emit Approval(owner, _spender, _value); return true; }

function transferFrom(address _from, address _to, uint256 _value) external returns (bool) { uint256 allowed = _allowances[_from][msg.sender]; require(_value <= allowed); require(_value <= _balances[_from]); _balances[_from] -= _value; _balances[_to] += _value; _allowances[_from][msg.sender] -= _value; emit
```

Transfer(_from, _to, _value); return true; }

event Transfer(address indexed _from, address indexed _to, uint256 _value); event Approval(address indexed _owner, address indexed _spender, uint256 _value); }

```

Custom Properties

Lets consider thetransfer function above.

There are several properties that could be specified over the transfer function:

1. If the transfer function succeeds then the recipient had sufficient balance at the start
2. If the transfer succeeds then the sender will have_value subtracted from it's balance
3. If the transfer succeeds then the receiver will have_value added to it's balance
4. If the transfer succeeds then the sum of the balances between the sender and receiver remains he same
5.

For this example we'll go with property #4;

> Try to write annotations for the other two properties after you've finished this getting started guide!

Formalizing the property

First things first, we need to dissect what the property actually entails. There are two main parts:if the transfer succeeds andthe sum of the sender and receiver balances is the same before and after the transaction .

if the transfer succeeds This specifies when you expect something to hold. In the scribble specification language we useif_succeeds function annotations to check properties after successful termination of a function. Check out thespecification docs for detailed information on theif_succeeds keyword.

the sum of the sender and receiver balances is the same before and after the transaction This is the boolean expression that we expect to hold. Note that this property introduces a statement that relates the initial and final state of the contract. That is, we want to know that the sum of the balances (_balances[sender] + _balances[receiver] ) doesn't change relative to the state before the transaction. To allow you to express such properties theold() is used. Check out thespecification docs for more information onold .

Using this keyword we can formulate the following expression:

```

Copy old(_balances[_to]) + old(_balances[msg.sender]) == _balances[_to] + _balances[msg.sender]

```

Now we can combine the two to get to our specification:

```

Copy if_succeeds {:msg "Transfer does not modify the sum of balances" } old(_balances[_to]) + old(_balances[msg.sender]) == _balances[_to] + _balances[msg.sender];

```

Adding the annotation

if_succeeds annotations are function specifications, which in the Scribble specification language are placed above the function definitions using a docstring comment.

```

Copy pragma solidity ^0.6.0;

contract ERC20 { mapping (address => uint256) private _balances; /// #if_succeeds {:msg "Transfer does not modify the sum of balances"} old(_balances[_to]) + old(_balances[msg.sender]) == _balances[_to] + _balances[msg.sender]; function transfer(address _to, uint256 _value) public returns (bool) { // ... } }

```

> Analysis

To see if the property holds, you can run mythril from the command line as follows:

```

Copy mythanalyzeToken.instrumented.sol-t1--execution-timeout60

```

The-t option tells mythril to explore only 1 transaction deep, and--execution-timeout specifies a timeout of 60 sec as the name implies. This should be sufficient, however if for some reason you don't see the expected failures, try a 2 minute timeout.

In about a minute mythril should return with two violations - an assertion violation and auser assertion violation:

```

Copy ==== Exception State ==== SWC ID: 110 Severity: Medium Contract: ERC20 Function name: transfer(address,uint256) PC address: 2014 Estimated Gas Usage: 18755 - 64531 A user-provided assertion failed. A user-provided assertion failed with the message '0: Transfer does not modify the sum of balances'

In file: Token.instrumented.sol:40

AssertionFailed("0: Transfer does not modify the sum of balances")

Initial State:

Account: [CREATOR], balance: 0x40008290082002429, nonce:0, storage:{} Account: [ATTACKER], balance: 0x0, nonce:0, storage:{} Account: [SOMEGUY], balance: 0x0, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], calldata: , value: 0x0 Caller: [CREATOR], function: transfer(address,uint256), txdata: 0xa9059cbb000000000000000000000000affeaffeaffeaffeaffeaffeaffeaffeaffe0000000000000000000000000000000000000000000000000000000000004f00, value: 0x0

==== Exception State ==== SWC ID: 110 Severity: Medium Contract: ERC20 Function name: transfer(address,uint256) PC address: 2021 Estimated Gas Usage: 19521 - 65553 An assertion violation was triggered. It is possible to trigger an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

In file: Token.instrumented.sol:41

assert(false)

Initial State:

Account: [CREATOR], balance: 0x10820020411009202, nonce:0, storage:{} Account: [ATTACKER], balance: 0x0, nonce:0, storage:{} Account: [SOMEGUY], balance: 0x0, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], calldata: , value: 0x0 Caller: [CREATOR], function: transfer(address,uint256), txdata: 0xa9059cbb000000000000000000000000affeaffeaffeaffeaffeaffeaffeaffeaffe0000000000000000000000000000000000000000000000000000000000080000, value: 0x0

```

And voila! We have found a counterexample to our property! Note that the first failure directly incudes the user-readable label from the assertion, which makes it easy to identify which property was violated if you have multiple:

```

Copy A user-provided assertion failed with the message '0: Transfer does not modify the sum of balances'

```

If you were to decode the transaction data:

```

Copy txdata: 0xa9059cbb000000000000000000000000affeaffeaffeaffeaffeaffeaffeaffeaffe0000000000000000000000000000000000000000000000000000000000004f00

```

You would find that the recipient of the transfer is0xaffeaffeaffeaffeaffeaffeaffeaffeaffe which is the address of theCREATOR account in mythril. That however is also the same account that sent the transaction:

```

Copy Caller: [CREATOR], function: transfer(address,uint256),....

```

So the failure occurs when the sender and receiver oftransfer are the same. We will leave it up to the reader to look at the code and figure out what goes wrong in this case.

Previous Property Checking with Scribble and MythXNext In-place testing with Scribble Last updated2 years ago

On this page * Intro * Dependencies * Setup * Background * Our example * Custom Properties * Analysis
```