Property Checking with Scribble and MythX

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 with MythX
-

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 mythx-cli:

```

Copy pip3install-Umythx-cli

```

Next you will need a MythX account. If you don't already have one, you can make a MythX account here .

Once you have an account you will need an API Key to authenticate to the mythx service. You can generate one by logging into the MythX dashboard , clicking on the "API Key"** menu entry and filling out the password field. (detailed instructions here). After you get your key, you can set it as theMYTHX_API_KEY environment variable:

```

Copy exportMYTHX_API_KEY=

```

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 below code 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 we will move to the terminal, and use theMythX CLI to start fuzzing and symbolic exploration.

In the directory with the annotated contract, run:

```

Copy

# The mythx analyze command can use Scribble to interpret the annotations, and send the instrumented code to the MythX API for analysis

mythxanalyze--scribbleToken.sol

```

```

Copy Found 1 job(s). Submit? [y/N]: y [####################################] 100%

```

☕ Relax for 2 minutes as the MythX robots do their work!

After about 2 minutes we get the following results:

```

Copy Report for flattened.sol https://dashboard.mythx.io/#/console/analyses/1de3dc57-7db3-4d69-9e10-0df745753676

| Line | SWC Title | Severity | Short Description |
| --- | --- | --- | --- |
| 40 | (SWC-110) Assert Violation | Low | A user-provided assertion failed. |
| 41 | (SWC-110) Assert Violation | Low | An assertion violation was triggered. |

```

Under the hood

The command you just ran actually did several steps. Firstmythx-cli invokedscribble to generate aninstrumented version of the contract. That is, a version of the original contract with the mostly the same behavior, except it also converted the property you had in the docstring into a concrete check at the end oftransfer . For more information on how instrumentation works check outthis section. If you want tosee the instrumented code, you can run

```

Copy scribbleToken.sol

```

Next Steps

ReadSpecification Language Docs to learn more about the specification language

Check out therepository of examples

Previous Quantification Next Property Checking with Scribble and Mythril Last updated2 years ago

Was this helpful?