Introduction to Smart Contract Analysis with Echidna

Presenters:
- Josselin Feist: josselin@trailofbits.com

Requirements:
- Echidna:
  - docker pull trailofbits/echidna
  - git clone https://github.com/trailofbits/publications/
  - cd "publications/workshops/Automated Smart Contracts Audit - TruffleCon 2018/echidna"
  - docker run -it -v $PWD:/home/echidna/trufflecon trailofbits/echidna
  - cd /home/echidna/trufflecon

All the material of this workshop is available in https://github.com/trailofbits/publications/.
It is recommended to use Echidna at commit dbe1a60a.

The aim of this document is to show how to use Echidna to automatically test smart contracts. The goal of the workshop is to solve the exercices proposed in Section 2. Section 1 introduces how to write a property for Echidna.

Slack: https://empireslacking.herokuapp.com/ #ethereum

1. Testing a property

We will see how to test a smart contract with Echidna. The target is the following smart contract:

```
contract Token{

   mapping(address => uint) public balances;
   function airdrop() public{
       balances[msg.sender] = 1000;
   }

   function consume() public{
       require(balances[msg.sender]>0);
       balances[msg.sender] -= 1;
   }
```

```
    function backdoor() public{
        balances[msg.sender] += 1;
    }
}
```

Figure 1: token.sol

We will make the assumption that this token must have the following properties:

- Anyone can have at maximum 1000 tokens
- The token cannot be transferred (it is not an erc20 token)

Writing a property

Echidna properties are Solidity functions. A property must:

- Have no argument
- Not change the state (i.e. it is a `view` function)
- Return true if it success
- Have its name starting with `echidna_`

Echidna will automatically generate transactions to make the property returning false, or throw an error.

The following property checks that the caller has no more than 1000 tokens:

```
    function echidna_balance_under_1000() public view returns(bool){
        return balances[msg.sender] <= 1000;
    }
```

Figure 2: Property Example

Use inheritance to separate your contract from your properties:

```
contract TestToken is Token{
    function echidna_balance_under_1000() public view returns(bool){
        return balances[msg.sender] <= 1000;
    }
}
```

Figure 3: Property through inheritance

Initiate your contract

Echidna needs a constructor without argument.
If you contract needs a specific initialization, you need to do it in the constructor.

There are two specific addresses in Echidna:

- `0x00a329c0648769a73afac7f9381e08fb43dbea72` which calls the constructor.
- `0x00a329c0648769a73afac7f9381e08fb43dbea70` which calls the other functions.

Running Echidna

Echidna is launched with:

```
$ echidna-test contract.sol
```

If `contract.sol` contains multiple contracts, you can specify the target:

```
$ echidna-test contract.sol MyContract
```

Summary: Testing a property

The following summarizes the run of echidna on our example:

```
contract TestToken is Token{
   constructor() public {}

   function echidna_balance_under_1000() public view returns(bool){
       return balances[msg.sender] <= 1000;
   }
}
```

Figure 4: testtoken.sol

```
$ echidna-test testtoken.sol TestToken
...
  ✗ "echidna_balance_under_1000" failed after 29 tests and 1
shrink.

    | Call sequence: airdrop();
    |                backdoor();

  ✗ 1 failed.
```

Figure 5: Echidna run on the example

Echidna found that the property is violated if `backdoor` is called.

2. Exercises

The two following exercises are meant to be solved using Echidna.

Targeted contract

We will test the following contract:

```
contract Ownership{
    address owner = msg.sender;
    function Owner(){
        owner = msg.sender;
    }
    modifier isOwner(){
        require(owner == msg.sender);
        _;
    }
}

contract Pausable is Ownership{
    bool is_paused;
    modifier ifNotPaused(){
        require(!is_paused);
        _;
    }

    function paused() isOwner public{
        is_paused = true;
    }

    function resume() isOwner public{
        is_paused = false;
    }
}

contract Token is Pausable{
    mapping(address => uint) public balances;
    function transfer(address to, uint value) ifNotPaused public{
        balances[msg.sender] -= value;
        balances[to] += value;
    }
}
```

Figure 6: token.sol

Exercise 1

Add a property to check that `echidna_caller` cannot have more than an initial balance of 10000.

The skeleton for this exercise is:

```
import "token.sol";

contract TestToken is Token {
   address echidna_caller = 0x00a329c0648769a73afac7f9381e08fb43dbea70;

   constructor() public{
       balances[echidna_caller] = 10000;
   }
   // add the property
}
```

Figure 7: exercise1.sol

Once Echidna found the bug, fix the issue, and re-try your property with Echidna.

Exercise 2

Now `paused` is called at deployment, and the ownership is removed.
Add a property to check that the contract cannot be unpaused.

The skeleton for this exercise is:

```
import "token.sol";

contract TestToken is Token {
   address echidna_caller = 0x00a329c0648769a73afac7f9381e08fb43dbea70;
   constructor(){
       paused(); // pause the contract
       owner = 0x0; // lose ownership
   }
   // add the property
}
```

Figure 8: exercise2.sol

Once Echidna found the bug, fix the issue, and re-try your property with Echidna.

Exercise 3 (Bonus)

Consider the following extension of the token:

```
import "token.sol";

contract MintableToken is Token{
   int totalMinted;
   int totalMintable;

   function MintableToken(int _totalMintable){
       totalMintable = _totalMintable;
   }

   function mint(uint value) isOwner(){
       require(int(value) + totalMinted < totalMintable);
       totalMinted += int(value);
       balances[msg.sender] += value;

   }
}
```

Figure 9: bonus.sol

Use the version of `token.sol` containing the fixes of the previous exercices.

Create a scenario, where `echidna_caller` (`0x00a329c0648769a73afac7f9381e08fb43dbea70`) becomes the owner of the contract at construction, and `totalMintable` is set to 10,000. Recall that Echidna needs a constructor without argument.

Add a property to check if `echidna_caller` cannot mint more than 10,000 tokens.

Once Echidna found the bug, fix the issue, and re-try your property with Echidna.