Echidna Training

Presenter: Josselin Feist, josselin@trailofbits.com

Requirements:

- git clone https://github.com/trailofbits/trufflecon-2019

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 exercises proposed in Section 2. Section 1 introduces how to write a property for Echidna.

Need help? Slack: https://empireslacking.herokuapp.com/ #echidna

Installation

Echidna can be installed through docker or using the pre-compiled binary.

Echidna through docker

```
docker pull trailofbits/eth-security-toolbox
docker run -it -v "$PWD":/home/trufflecon trailofbits/eth-security-toolbox
```

The last command runs eth-security-toolbox in a docker that has access to your current directory. You can change the files from your host, and run the tools on the files from the docker

Inside docker, run :

```
solc-select 0.5.3
cd /home/trufflecon/
```

Binary

https://github.com/crytic/echidna/releases/tag/1.0.0.0

solc 0.5.3 is recommended.

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!💥
  Call sequence:
    airdrop()
    backdoor()
```

Figure 5: Echidna run on the example

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

Exercise 1 : Access control

We are going to see how to test the correct access control of a contract.

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

Property

We will simulate a state where the contract is paused, and the owernship set to 0x0. Our goal is to determine if anyone can unpause the contract.

The skeleton for this exercise is:

```
import "token.sol";

contract TestToken is Token {

   constructor(){
       paused(); // pause the contract
       owner = 0x0; // lose ownership
   }
   // add the property
}
```

Figure 8: template.sol

Write a property to ensure that no one can transfer tokens.

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

Exercise 2 - Arithmetic (Bonus)

Property

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

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: template.sol

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