Manticore Training

Presenter: Josselin Feist, josselin@trailofbits.com

Requirements:

- Basic Python knowledge
- Manticore:
    - pip install --user manticore
    - git clone https://github.com/publications
    - cd "publications/workshops/Using Manticore and Symbolic Execution to Find Smart Contracts Bugs - Devcon 4"

Manticore 0.2.2 is recommended.

The two following exercises are meant to be solved using the features detailed in the Manticore API document.

Each exercise corresponds to automatically find a vulnerability with a Manticore script. Once the vulnerability is found, Manticore can be applied to a fixed version of the contract, to demonstrate that the bug is effectively fixed.

The solution presented during the workshop will be based on the proposed scenario of each exercise, but other solutions are possible. Each scenario is composed of three steps: the initialization, the exploration and the check of properties.

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

Example: Incorrect token transfer

This scenario is given as an example. You can follow its structure to solve the following exercises.

my_token.py uses manticore to find for an attacker to generate tokens during a transfer on Token (my_token.sol).

Proposed scenario

Initialization:

- Create one user account
- Create the contract account

Exploration:

- Call `balances` on the user account
- Call transfer with symbolic destination and value
- Call `balances` on the user account

Property:
- Check if the user can have more token after the transfer than before.

Unprotected function

Use Manticore to find an input allowing an attacker to steal ethers from UnprotectedWallet. Propose a fix of the contract, and test your fix using your Manticore script.

Proposed scenario
Initialization:
- Create two accounts, one for the creator (with 10 ether), one for the attacker (with 0 ether)
- Create the contract account

Exploration:
- Creator calls deposit() with 1 ether
- The attacker calls two functions (explore any function, using raw transactions)

Property:
- An attack is found if on one of the path, the attacker's balance has 1 ether.

```
contract UnprotectedWallet{
   address public owner;


   modifier onlyowner {
       require(msg.sender==owner);
       _;
   }

   constructor() public {
       owner = msg.sender;
   }
   function changeOwner(address _newOwner) public {
      owner = _newOwner;
   }


   function deposit() payable public {
   }


   function withdraw() onlyowner public {
       msg.sender.transfer(this.balance);
   }
}
```

Figure 1: unprotected.sol

Hints:
The balance of an account can be accessed through:

```
state.platform.get_balance(account.address)
```

Integer overflow

Use Manticore to find if an overflow is possible in Overflow.add. Propose a fix of the contract, and test your fix using your Manticore script.

Proposed scenario

Initialization
- Create one user account
- Create the contract account

Exploration
- Call two times add with two symbolic values
- Call sellerBalance()

Property:
- Check if it is possible for the value returned by sellerBalance() to be lower than the first input.

```
pragma solidity^0.4.24;
contract Overflow {
    uint public sellerBalance=0;

    function add(uint value) public returns (bool){
        sellerBalance += value; // complicated math, possible overflow
    }
}
```

Figure 2: overflow.sol

Hints:
The value returned by the last transaction can be accessed through:

```
state.platform.transactions[-1].return_data
```

The data returned needs to be deserialized:

```
data = ABI.deserialize("uint", data)
```

To add a constraint a > b on two unsigned integers use:

```
state.constrain(Operators.UGT(a, b))
```