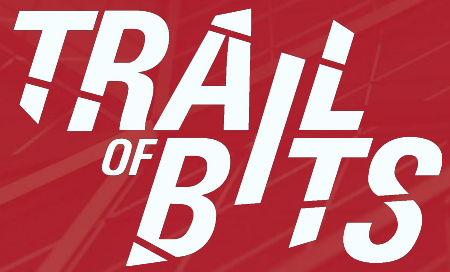


Using Manticore and Symbolic Execution to Find Smart Contracts Bugs

Who Am I?

- **Josselin Feist, josselin@trailofbits.com**

- **Trail of Bits: trailofbits.com**
  - We help organizations build safer software
  - R&D focused: we use the latest program analysis techniques

Goals

- What is Symbolic Execution?
- How can it helps build more secure smart contracts?
- Hands-on with Manticore

Before Starting

- **git clone https://github.com/publications**
- **pip3 install manticore --user**
  - Or docker pull trailofbits/manticore

Automated Smart Contracts Audit

Problem: How to Find Bugs?

- How to test if a smart contract has bugs?

```
contract Simple {
    function f(uint a) payable public {
        if (a == 65) {
            // bug here
        }
    }
}
```

Problem: How to Find Bugs?

- **Manual review: hard, don’t protect for future bugs**
  - Contact a security company
- **Unit tests: cover a small part**
  - Use Truffle

Time consuming, usually low coverage

Finding Bugs With Automated Analysis

- **Static analysis (e.g. Slither)**
  - Look for patterns in the code
  - Fast
  - No user-intervention required

Finding Bugs With Automated Analysis

- **Fuzzing (e.g. Echidna)**
  - Stress the contract through pseudo-random transactions
  - Best effort to explore the behaviors: **testing**
  - Successful technique for ‘classic software’ (e.g. AFL)

Finding Bugs With Automated Analysis

- **Symbolic Execution (e.g. Manticore)**
  - Generate inputs through mathematical representation of the contract
  - Explores all the paths of the contract: **code verification**

Finding Bugs With Automated Analysis

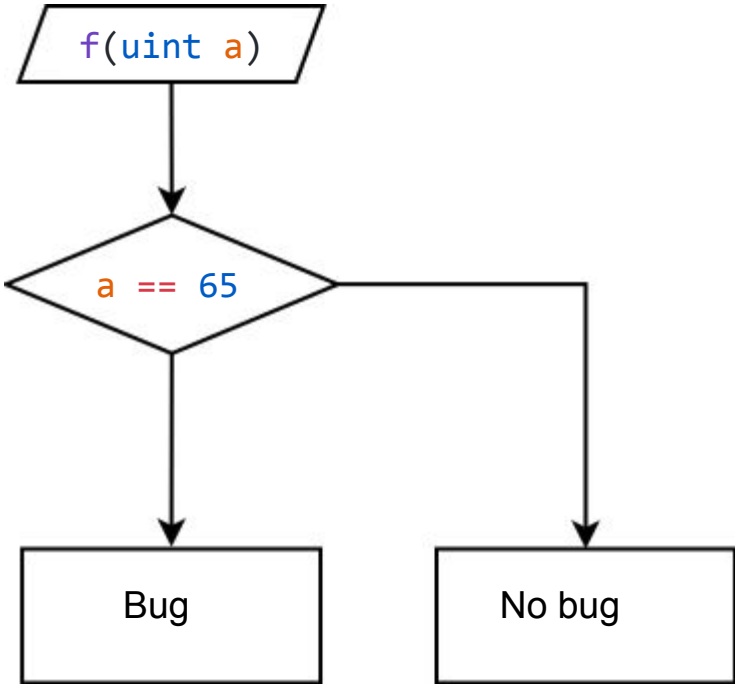
Technique	Tool	Speed	Complexity	Precision
Static Analysis	Slither	second	(no configuration)	+
Fuzzing	Echidna	minutes	+	++
Symbolic Execution	Manticore	hour	++	+++ (Verification)

Symbolic Execution

Symbolic Execution in a Nutshell

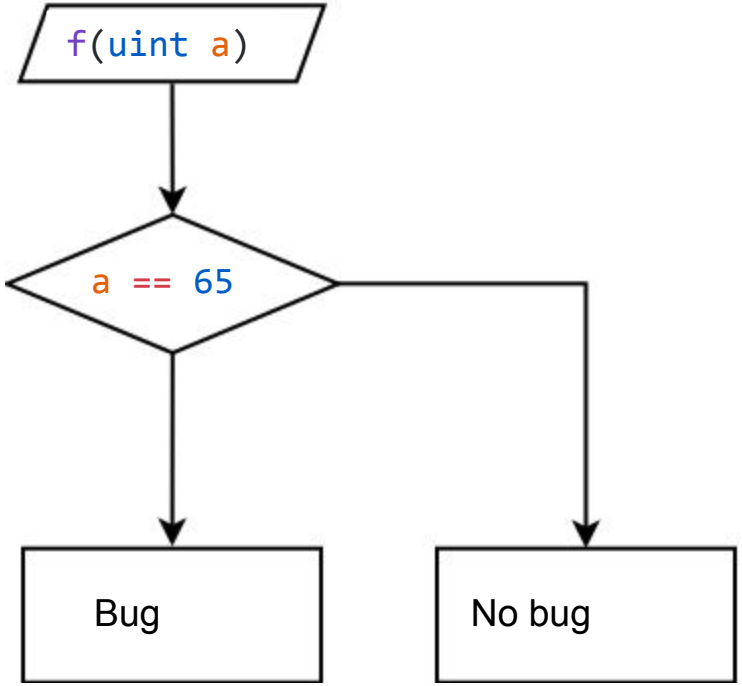
- **Program exploration technique**
- **Execute the program “symbolically”**
  - Represent executions as logical formulas
  - Fork on each condition
- **Use an SMT solver to check the feasibility of a path and generate inputs**

Symbolic Execution Example

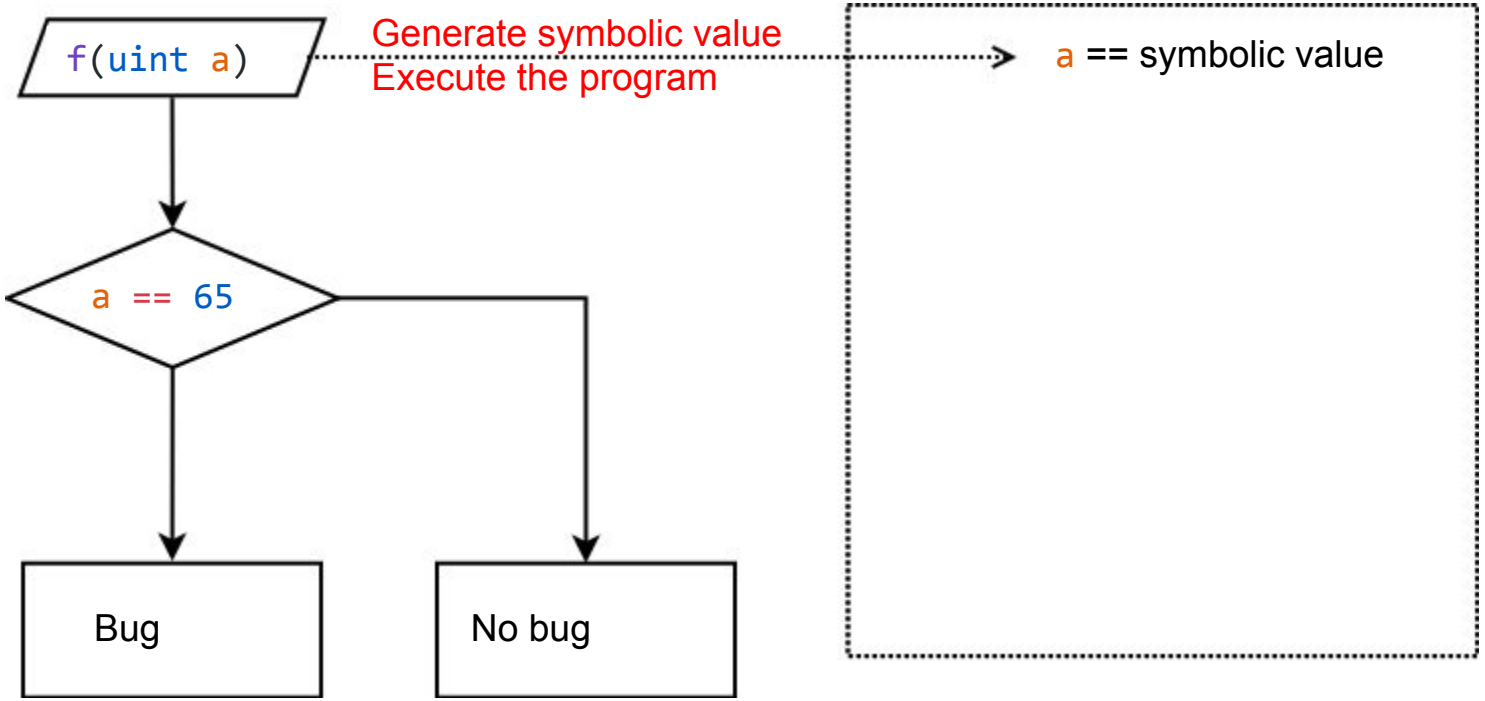



```
contract Simple {
   function f(uint a) payable public {
       // lot of paths and conditions
       if (a == 65) {
           // bug here
       }
   }
}
```

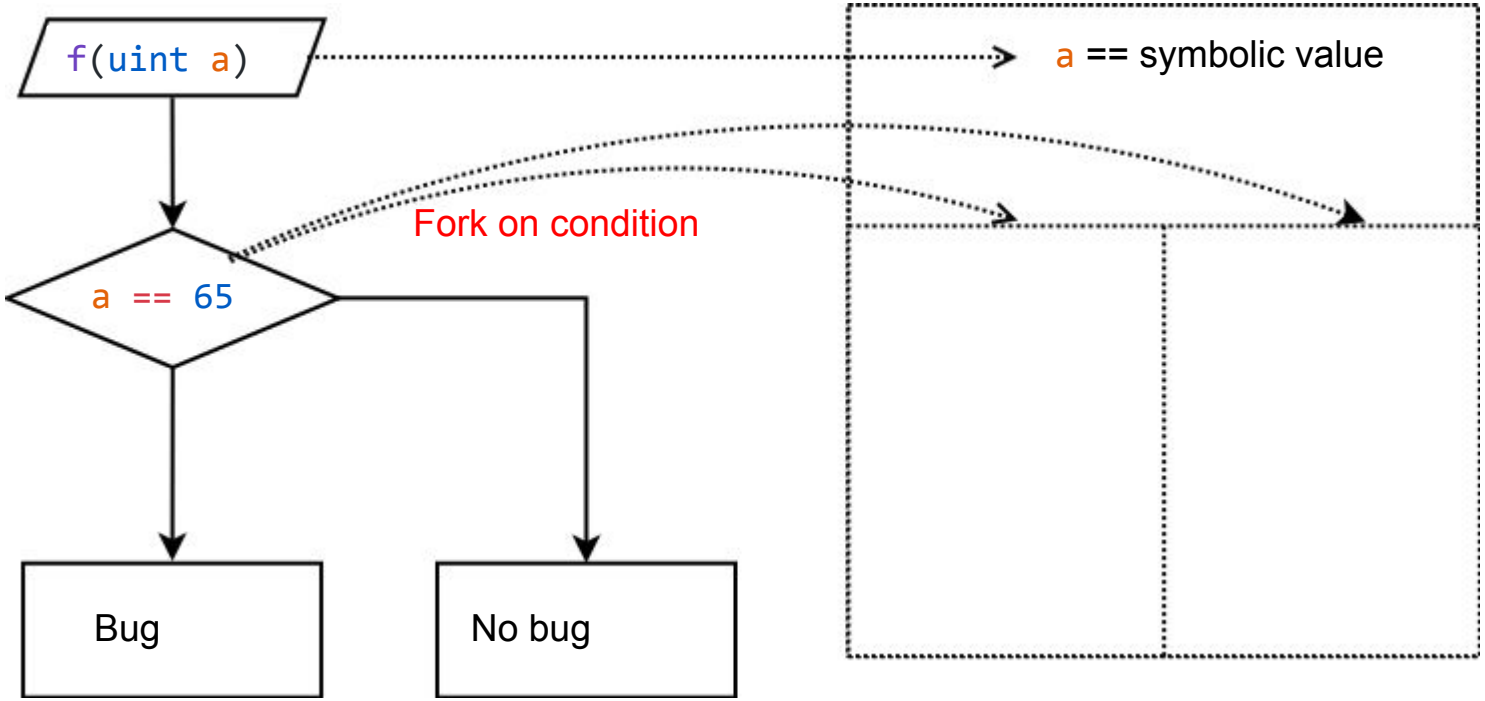
Symbolic Execution Example



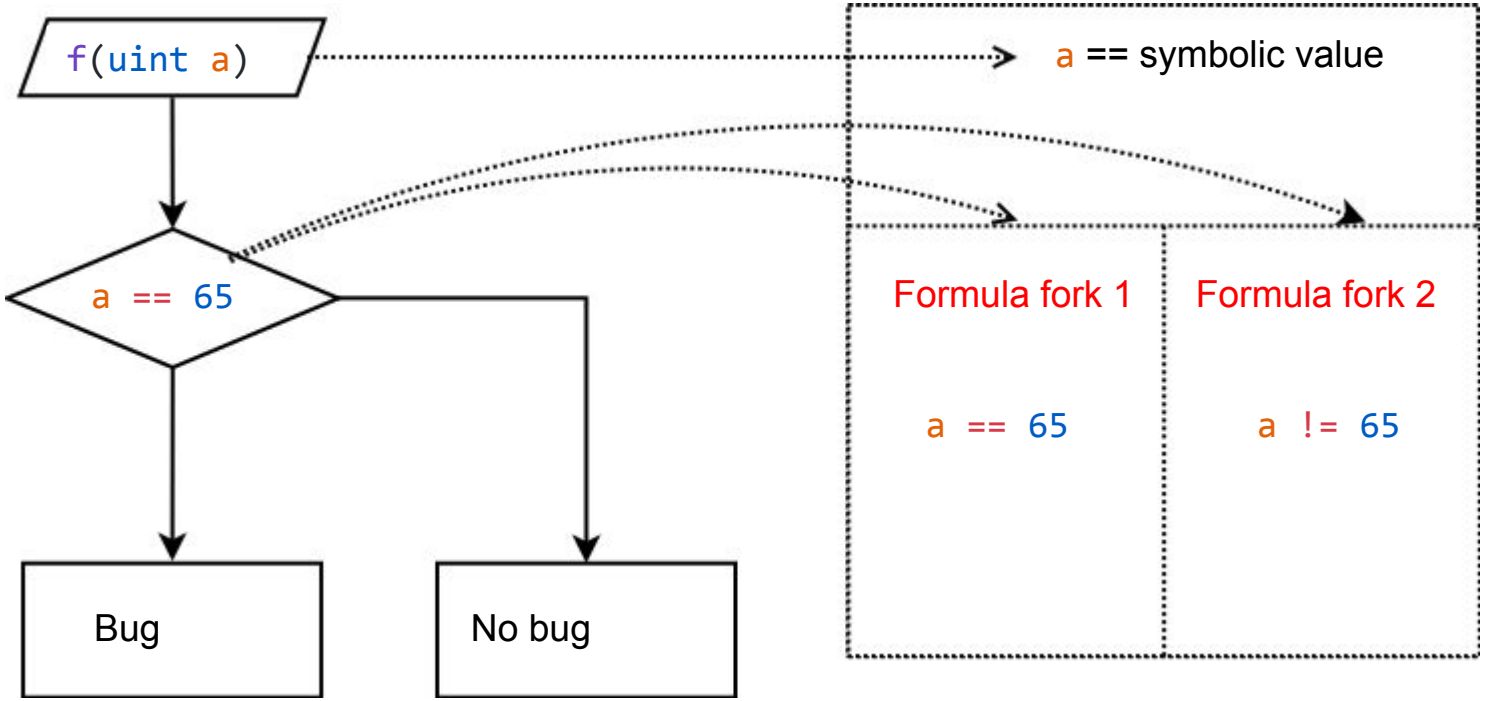
Symbolic Execution Example



Symbolic Execution Example



Symbolic Execution Example



Symbolic Execution in a Nutshell

- Explore the program automatically
- Allow to find unexpected paths
- Possibility to add arbitrary conditions

Manticore

Manticore

- **A symbolic execution engine supporting EVM**
- **Builtin detectors for classic issues**
  - Selfdestruct, External Call, Reentrancy, Delegatecall, ...
- **Python API for generic instrumentation**
  - Today's goal

Manticore: Command Line

```
contract Suicidal {
    function backdoor() {
        selfdestruct(msg.sender);
    }
}
```

Manticore: Command Line

```
$ manticore examples/suicidal.sol --detect-selfdestruct

m.main:INFO: Beginning analysis
m.ethereum:INFO: Starting symbolic create contract
m.ethereum:INFO: Starting symbolic transaction: 0
m.ethereum:WARNING: Reachable SELFDESTRUCT
m.ethereum:INFO: 0 alive states, 4 terminated states
m.ethereum:INFO: Starting symbolic transaction: 1
m.ethereum:INFO: Generated testcase No. 0 - RETURN
m.ethereum:INFO: Generated testcase No. 1 - REVERT
m.ethereum:INFO: Generated testcase No. 2 - SELFDESTRUCT
m.ethereum:INFO: Generated testcase No. 3 - REVERT
m.ethereum:INFO: Results in /home/manticore/mcore_9pqdsgtc
```

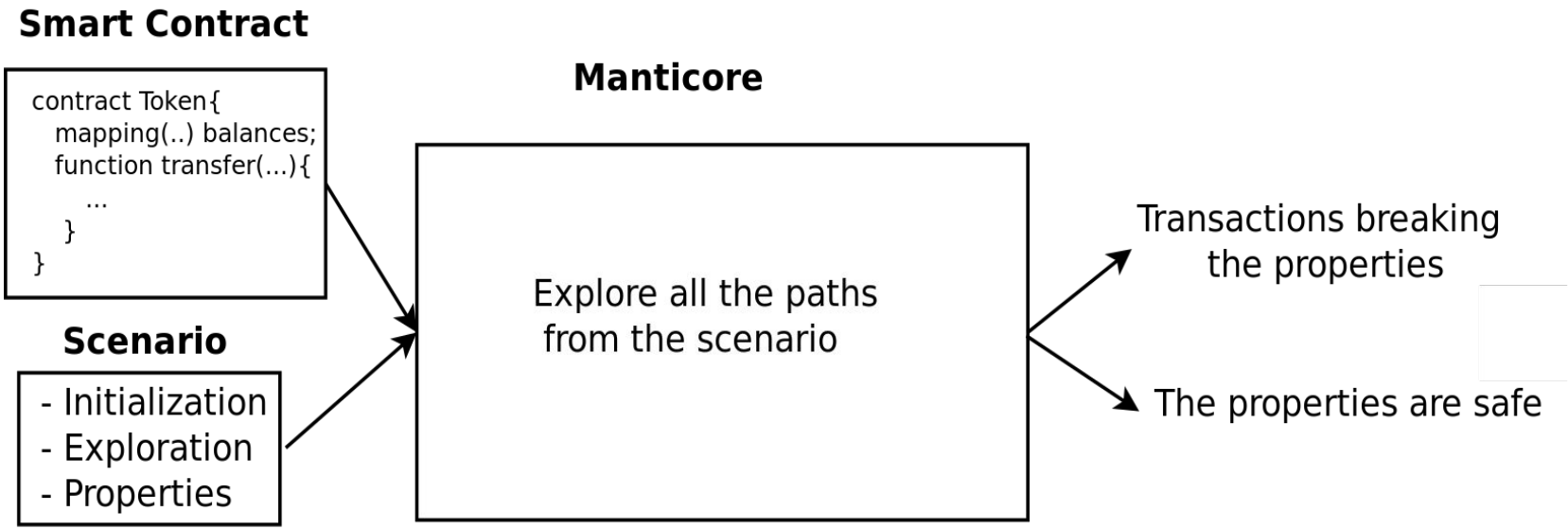
Manticore: Command Line

```
$ cat mcore_9pqdsgtc/test_00000002.tx

Transactions Nr. 0
...
Function call:
Constructor() -> RETURN

Transactions Nr. 1
..
Function call:
backdoor() -> SELFDESTRUCT (*)
```

Manticore: Python API

- **Python API to express arbitrary properties**
- **Scenario = 3 steps:**
  - Initialization: what contracts, how many users?
  - Exploration: what functions to explore, what is symbolic
  - Properties to check: what should happen/what should not happen

Manticore: Python API



Manticore: Python API

- Find if someone can steal tokens

```
function transfer(address to, uint val){
       if(balances[msg.sender] >= balances[to]){
           balances[msg.sender] -= val;
           balances[to] += val;
       }
   }
```

Manticore: Python API

Steps:

1. Initialization: Deploy contract
2. Exploration: Call transfer with symbolic values
3. Property: sender's balance does not increase

Manticore: Python API

```
from manticore.ethereum import ManticoreEVM, ABI
from manticore.core.smtlib import Operators

# Initialization
m = ManticoreEVM()
with open('my_token.sol') as f:
   source_code = f.read()
user_account = m.create_account(balance=1000)
contract_account = m.solidity_create_contract(source_code, owner=user_account, balance=0)
```

Initialization:
Create an user account and deploy the contract

Manticore: Python API

```
from manticore.ethereum import ManticoreEVM, ABI
from manticore.core.smtlib import Operators

# Initialization
m = ManticoreEVM()
with open('my_token.sol') as f:
   source_code = f.read()
user_account = m.create_account(balance=1000)
contract_account = m.solidity_create_contract(source_code, owner=user_account, balance=0)
```

```
# Exploration
contract_account.balances(user_account)

symbolic_val = m.make_symbolic_value()
symbolic_to = m.make_symbolic_value()
contract_account.transfer(symbolic_to, symbolic_val)

contract_account.balances(user_account)
```

Exploration:

- Collect the balance
- Call transfer with symbolic values
- Collect the new balance

Manticore: Python API

```
# Check of properties
bug_found = False
for state in m.running_states:

   balance_before = state.platform.transactions[1].return_data
   balance_before = ABI.deserialize("uint", balance_before)

   balance_after = state.platform.transactions[-1].return_data
   balance_after = ABI.deserialize("uint", balance_after)

   state.constrain(Operators.UGT(balance_after, balance_before))

   if state.is_feasible():
       print("Bug found! see {}".format(m.workspace))
       m.generate_testcase(state, 'Bug')
       bug_found = True

if not bug_found:
   print('No bug were found')
```

Bug found if:

balance_after(sender) > balance_before(sender)

Bug found!

```
$ cat mcore_.../Bug_00000000.tx



balances(..) -> 100

transfer(...,20430840703553386272388160528996790065041473555354846411818661786570194
945)

balances(..)
->1157716583966126420372985968481589110632049431920852091930457653461265594450 91
```

Bug found!

```
function transfer(address to, uint val){
        if(balances[msg.sender] >= balances[to]){
            balances[msg.sender] -= val;
            balances[to] += val;
        }
    }
```

Manticore: Exercise 1

Can you steal ethers?

- https://github.com/publications
- publications/workshops/Using Manticore and Symbolic Execution to Find Smart Contracts Bugs - Devcon 4
- Open manticore_api.pdf
- Open exercises.pdf

Can you steal ethers?

```
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);
    }
}
```

Manticore: Exercise 1 Solution

Solution

```
from manticore.ethereum import ManticoreEVM
from manticore.core.smtlib import Operators, solver

m = ManticoreEVM()
with open('unprotected.sol') as f:
    source_code = f.read()

# Generate the accounts. Creator has 10 ethers; attacker 0
creator_account = m.create_account(balance=10*10**18)
attacker_account = m.create_account(balance=0)
contract_account = m.solidity_create_contract(source_code, owner=creator_account)
```

Solution

```
# Deposit 1 ether, from the creator
contract_account.deposit(caller=creator_account, value=10**18)
```

Solution

```
# Two raw transactions from the attacker
symbolic_data = m.make_symbolic_buffer(320)
m.transaction(caller=attacker_account,
              address=contract_account,
              data=symbolic_data,
              value=0)
symbolic_data = m.make_symbolic_buffer(320)
m.transaction(caller=attacker_account,
              address=contract_account,
              data=symbolic_data,
              value=0)
```

Solution

```
for state in m.running_states:
    # Check if the attacker can ends with some ether


    balance = state.platform.get_balance(attacker_account.address)
    state.constrain(balance > 1)


    if state.is_feasible():
        print("Attacker can steal the ether! see {}".format(m.workspace)
        m.generate_testcase(state, 'WalletHack')
```

Solution

```
$ cat mcore_.../WalletHack_00000000.tx

…

deposit() -> STOP

…

changeOwner(113295552048775031759123758081492326321685290549 2) -> STOP (*)

…

withdraw() -> STOP (*)
```

Manticore: Exercise 2

Is an Integer Overflow Possible?

```
contract Overflow {
    uint public sellerBalance = 0;

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

- **There are many ways to check it**
  - The one proposed is not the simplest, but it will allow you to get familiar with Manticore!

Manticore: Exercise 2 Solution

Solution

```
from manticore.ethereum import ManticoreEVM, ABI
from manticore.core.smtlib import Operators

m = ManticoreEVM() # initiate the blockchain
with open('overflow.sol') as f:
   source_code = f.read()

# Generate the accounts
user_account = m.create_account(balance=1000)
contract_account = m.solidity_create_contract(source_code, owner=user_account,
balance=0)
```

Solution

```
# First add won't overflow uint256 representation
value_0 = m.make_symbolic_value()
contract_account.add(value_0)
# Potential overflow
value_1 = m.make_symbolic_value()
contract_account.add(value_1)
contract_account.sellerBalance()
```

Solution

```
for state in m.running_states:
   # Check if input0 > sellerBalance

   # last_return is the data returned
   last_return = state.platform.transactions[-1].return_data
   last_return = ABI.deserialize("uint", last_return)

   state.constrain(Operators.UGT(value_0, last_return))

   if state.is_feasible():
       print("Overflow found! see {}".format(m.workspace))
       m.generate_testcase(state, 'OverflowFound')
```

Solution

```
$ cat mcore_.../OverflowFound_00000000.tx

...

add(606613267268583294395704282859755566477516074631091675046538409410595688611 85)
-> RETURN (*)

add(696720803593263343806332913725397222283339363697467491096097938909489738547 21)
-> RETURN (*)

sellerBalance() -> RETURN

return:
145413178488684683966327346498273710228155591672153525748060508240954130759 70 (*)
```

Workshop Summary

Manticore: Summary

- Manticore will verify your code
- You can verify high-level and low-level properties
- Manticore will help to trust your code

Workshop Summary

- **Our tools will help you building safer smart contracts**
  - Manticore: https://github.com/trailofbits/manticore/
  - Slither: https://github.com/trailofbits/slither/
  - Echidna: https://github.com/trailofbits/echidna/
  - Etheno: https://github.com/trailofbits/etheno
- **If you need help: https://empireslacking.herokuapp.com/**
  - #ethereum
- **We pay bounties!**