Symbolic Execution of Smart Contracts with Manticore

EthCC 2018

Who are we?

- Josselin Feist, josselin@trailofbits.com
- JP Smith, jp@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 Symbolic Execution can help build more secure smart contracts?
- Hands-on with Manticore

Before Starting

- git clone https://github.com/trailofbits/workshops/
  - “Manticore - EthCC 2018”
  - All the files for this workshop
- git clone https://github.com/trailofbits/manticore
  - cd manticore
  - pip2 install --user .

Smart Contract Symbolic Execution

Problems

- How to test the presence of bugs in smart contracts?

```
contract Simple {
    function f(uint a){

        // .. lot of paths and conditions

        if (a == 65) {
            // lead to a bug here
        }
    }
}
```

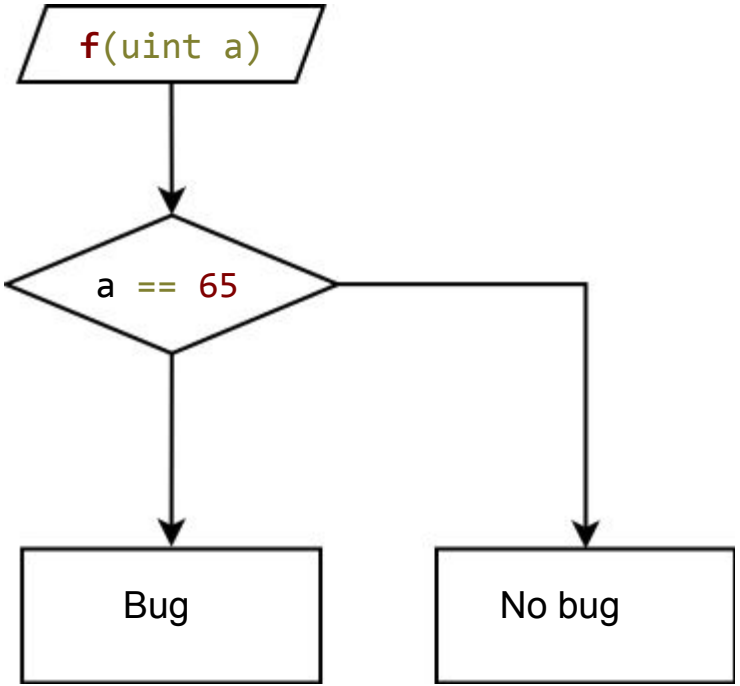
How to Review Code

- Manual review: time-consuming, every modification of the contract may lead to introducing a bug
- Unit tests: only cover a small part of the program's behavior
- Other techniques:
  - Static analysis (e.g. Slither)
  - Fuzzing (e.g. Echidna)
  - **Symbolic Execution** (e.g. Manticore)

Symbolic Execution in a Nutshell

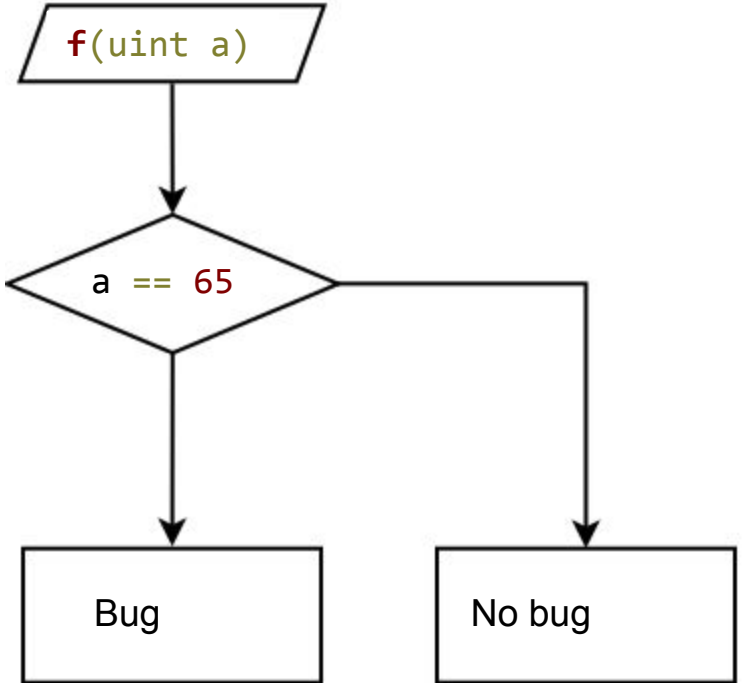
- Program exploration technique
- Execute the program “symbolically” = Represent executions as logical formulas
- Use an SMT solver to check the feasibility of a path and generate inputs

Symbolic Execution Example

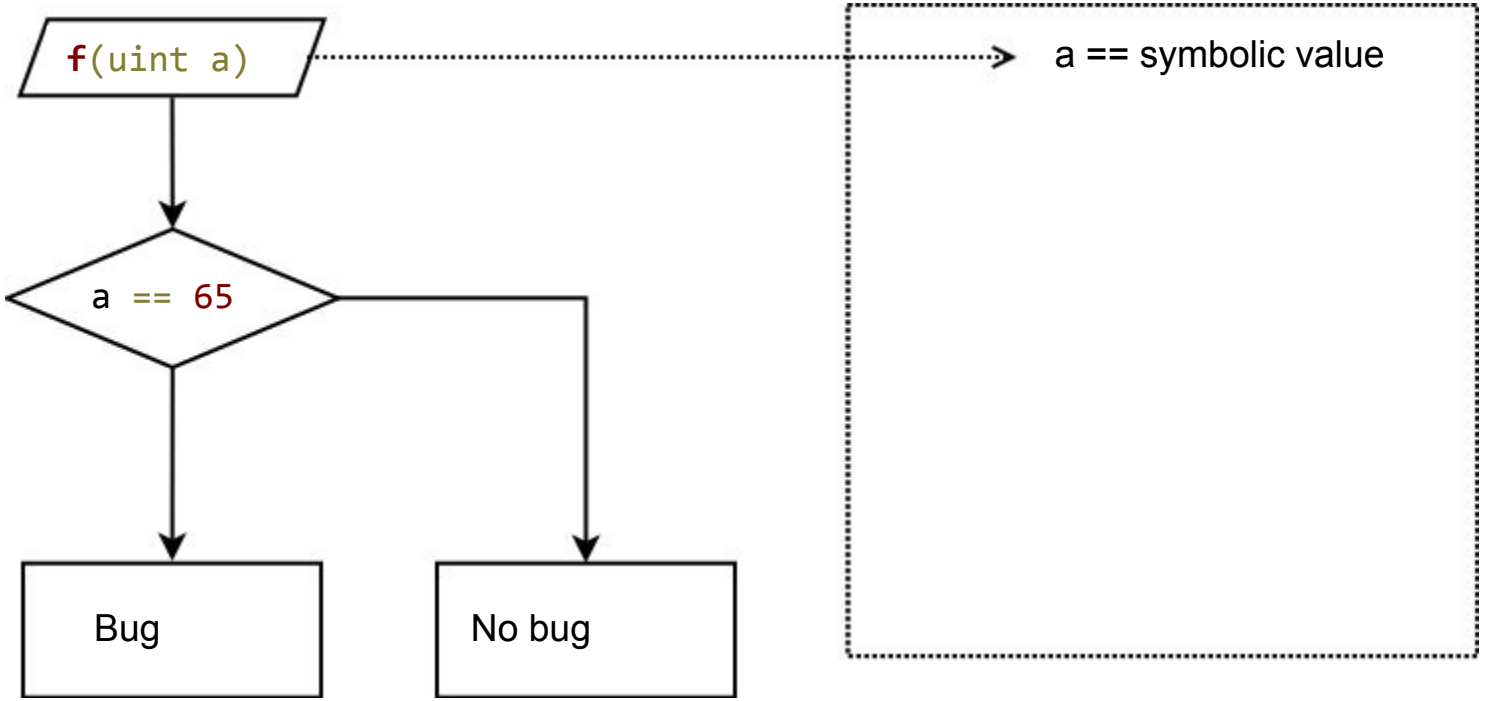



```
contract Simple {
    function f(uint a){

        // .. lot of paths and conditions

        if (a == 65) {
            // lead to a bug here
        }
    }
}
```

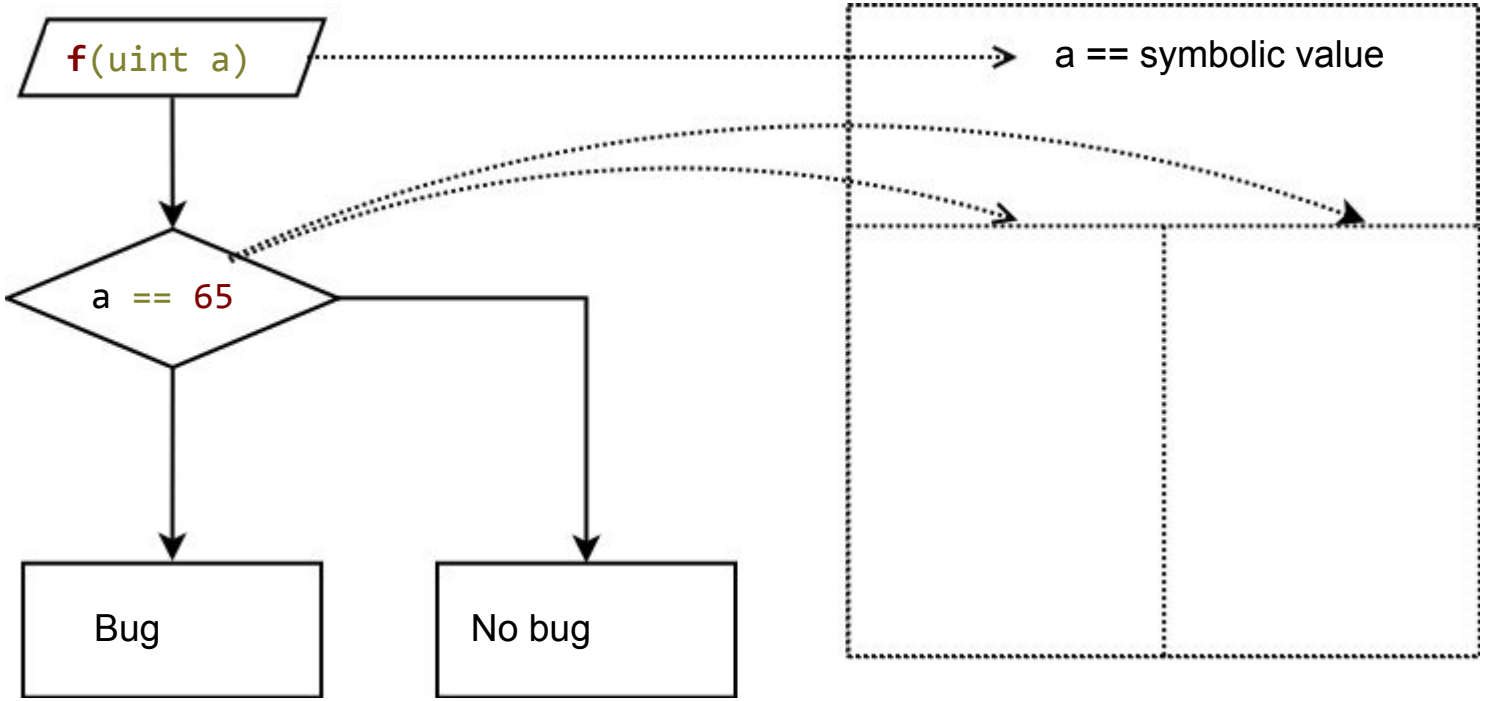
Symbolic Execution Example



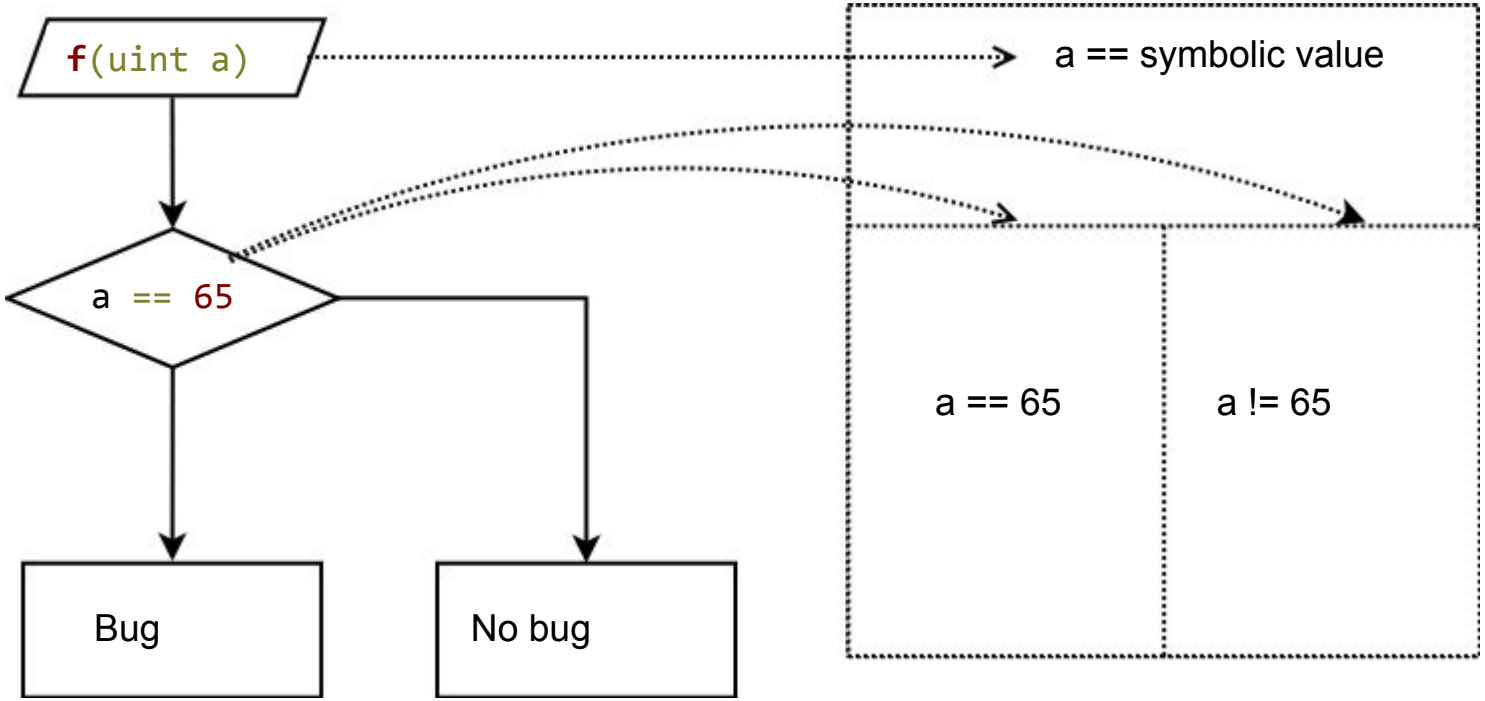
Symbolic Execution Example



Symbolic Execution Example



Symbolic Execution Example



Symbolic Execution

- Explore the program automatically
- Allow to find unexpected paths

Manticore: Automatic Analysis

Manticore - EVM

- A symbolic execution engine for EVM
- All possible contract paths are explored
- Supports multiple contracts and transactions
- API for generic instrumentation

Manticore: Simple Example

```
$ cat simple.sol
pragma solidity^0.4.20;
contract Simple {
    function f(uint a) payable public{
        if (a == 65) {
            revert();
        }
    }
}
```

Manticore Run Example

```
$ manticore simple.sol
2018-02-28 17:06:21,650: [25981] m.main:INFO: Beginning analysis
2018-02-28 17:06:21,803: [25981] m.ethereum:INFO: Starting symbolic transaction: 1
2018-02-28 17:06:22,098: [25981] m.ethereum:INFO: Generated testcase No. 0 - REVERT
2018-02-28 17:06:23,185: [25981] m.ethereum:INFO: Generated testcase No. 1 - REVERT
2018-02-28 17:06:24,206: [25981] m.ethereum:INFO: Finished symbolic transaction: 1 |
Code Coverage: 100% | Terminated States: 3 | Alive States: 1
2018-02-28 17:06:24,213: [32058] m.ethereum:INFO: Generated testcase No. 2 - STOP
2018-02-28 17:06:25,269: [25981] m.ethereum:INFO: Results in /examples/mcore_zua0Yl
```

Manticore Run Example

```
$ manticore simple.sol
2018-02-28 17:06:21,650: [25981] m.main:INFO: Beginning analysis
2018-02-28 17:06:21,803: [25981] m.ethereum:INFO: Starting symbolic transaction: 1
2018-02-28 17:06:22,098: [25981] m.ethereum:INFO: Generated testcase No. 0 - REVERT
2018-02-28 17:06:23,185: [25981] m.ethereum:INFO: Generated testcase No. 1 - REVERT
2018-02-28 17:06:24,206: [25981] m.ethereum:INFO: Finished symbolic transaction: 1 |
Code Coverage: 100% | Terminated States: 3 | Alive States: 1
2018-02-28 17:06:24,213: [32058] m.ethereum:INFO: Generated testcase No. 2 - STOP
2018-02-28 17:06:25,269: [25981] m.ethereum:INFO: Results in /examples/mcore_zua0Yl
```

# of paths explored

Manticore Run Example

```
$ manticore simple.sol
2018-02-28 17:06:21,650: [25981] m.main:INFO: Beginning analysis
2018-02-28 17:06:21,803: [25981] m.ethereum:INFO: Starting symbolic transaction: 1
2018-02-28 17:06:22,098: [25981] m.ethereum:INFO: Generated testcase No. 0 - REVERT
2018-02-28 17:06:23,185: [25981] m.ethereum:INFO: Generated testcase No. 1 - REVERT
2018-02-28 17:06:24,206: [25981] m.ethereum:INFO: Finished symbolic transaction: 1 |
Code Coverage: 100% | Terminated States: 3 | Alive States: 1
2018-02-28 17:06:24,213: [32058] m.ethereum:INFO: Generated testcase No. 2 - STOP
2018-02-28 17:06:25,269: [25981] m.ethereum:INFO: Results in /examples/mcore_zua0Yl
```

Exploration information

Manticore Run Example

```
$ manticore simple.sol
2018-02-28 17:06:21,650: [25981] m.main:INFO: Beginning analysis
2018-02-28 17:06:21,803: [25981] m.ethereum:INFO: Starting symbolic transaction: 1
2018-02-28 17:06:22,098: [25981] m.ethereum:INFO: Generated testcase No. 0 - REVERT
2018-02-28 17:06:23,185: [25981] m.ethereum:INFO: Generated testcase No. 1 - REVERT
2018-02-28 17:06:24,206: [25981] m.ethereum:INFO: Finished symbolic transaction: 1 |
Code Coverage: 100% | Terminated States: 3 | Alive States: 1
2018-02-28 17:06:24,213: [32058] m.ethereum:INFO: Generated testcase No. 2 - STOP
2018-02-28 17:06:25,269: [25981] m.ethereum:INFO: Results in /examples/mcore_zua0Yl
```

Output directory

Manticore Output Example

```
$ ls mcore_lDfo6m/
```

global_Simple.init_asm	test_00000000.constraints	test_00000001.tx
global_Simple_init.bytecode	test_00000000.logs	test_00000002.constraints
global_Simple.init_visited	test_00000000.pkl	test_00000002.logs
global_Simple.runtime_asm	test_00000000.summary	test_00000002.pkl
global_Simple_runtime.bytecode	test_00000000.tx	test_00000002.summary
global_Simple.runtime_visited	test_00000001.constraints	test_00000002.tx
global_Simple.sol	test_00000001.logs	visited.txt
global.summary	test_00000001.pkl	
state_0000000d.pkl	test_00000001.summary	

Manticore Output Example

```
$ ls mcore_lDfo6m/
```

global_Simple.init_asm	test_00000000.constraints	**test_00000001.tx**
global_Simple_init.bytecode	test_00000000.logs	test_00000002.constraints
global_Simple.init_visited	test_00000000.pkl	test_00000002.logs
global_Simple.runtime_asm	test_00000000.summary	test_00000002.pkl
global_Simple_runtime.bytecode	**test_00000000.tx**	test_00000002.summary
global_Simple.runtime_visited	test_00000001.constraints	**test_00000002.tx**
global_Simple.sol	test_00000001.logs	visited.txt
global.summary	test_00000001.pkl	
state_0000000d.pkl	test_00000001.summary	

Transactions information

Manticore Output Example

```
$ cat test_00000001.tx
```
Transactions Nr. 0
Type: Create
From: 0xd30a286ec6737b8b2a6a7b5fbb5d75b895f62956
To: 0x1bfa530d5d685155e98cd7d9dd23f7b6a801cfef
Value: 0
Data: 6060604052341...5a0029
Return_data: 6060604052600436106103..3fe81d8ff9d1568084695a0029

Transactions Nr. 1
Type: Call
From: 0xd51f25e60490392aa9eb72624f93de30ccd111f3
To: 0x1bfa530d5d685155e98cd7d9dd23f7b6a801cfef
Value: 1 (*)
Data: b3de648b0000..0000000000 (*)
Return_data:

Function call:
f(65) -> REVERT (*)
return: ()

Manticore Output Example

```
$ cat test_00000001.tx
Transactions Nr. 0
Type: Create
From: 0xd30a286ec6737b8b2a6a7b5fbb5d75b895f62956
To: 0x1bfa530d5d685155e98cd7d9dd23f7b6a801cfef
Value: 0
Data: 6060604052341...5a0029
Return_data: 606060405260043610603..3fe81d8ff9d1568084695a0029

Transactions Nr. 1
Type: Call
From: 0xd51f25e60490392aa9eb72624f93de30ccd111f3
To: 0x1bfa530d5d685155e98cd7d9dd23f7b6a801cfef
Value: 1 (*)
Data: b3de648b0000..0000000000 (*)
Return_data:

Function call:
f(65) -> REVERT (*)
return: ()
```

Tx #0: constructor

Manticore Output Example

```
$ cat test_00000001.tx
Transactions Nr. 0
Type: Create
From: 0xd30a286ec6737b8b2a6a7b5fbb5d75b895f62956
To: 0x1bfa530d5d685155e98cd7d9dd23f7b6a801cfef
Value: 0
Data: 6060604052341...5a0029
Return_data: 6060604052600436106103..3fe81d8ff9d1568084695a0029
```

```
$ cat simple.sol
 pragma solidity^0.4.20;
 contract Simple {
     function f(uint a) payable public{
         if (a == 65) {
             revert();
         }
     }
 }
```

Transactions Nr. 1
Type: Call
From: 0xd51f25e60490392aa9eb72624f93de30ccd111f3
To: 0x1bfa530d5d685155e98cd7d9dd23f7b6a801cfef
Value: 1 (*)
Data: b3de648b0000..0000000000 (*)
Return_data:

Function call:
f(65) -> REVERT (*)
return: ()

Tx #1: f(65) -> leads to revert the transaction

Manticore: Assisted Analysis

Finding Smart Contract Vulnerabilities

- “Classic” vulnerabilities
  - Integer overflow/underflow/..
- Logic vulnerabilities/errors in the design
- What is a vulnerability in a contract?
  - It depends on the contract purpose!
- A user ends with more ethers than invested, is it a bug?
  - Yes, if the contract is a paid service
  - No, if the contract is a lottery

Finding Smart Contract Vulnerabilities

- Solution: assisted analysis == benefit from users' knowledge
- Manticore: full python API to script

Manticore Example

- Find all the paths leading `f()` to crash

```
contract Simple {
    function f(uint a) payable public{
        if (a == 65) {
            revert();
        }
    }
}
```

```
# simple.py
from manticore.ethereum import ManticoreEVM

m = ManticoreEVM() # initiate the blockchain
source_code = '''
pragma solidity^0.4.20;
contract Simple {
    function f(uint a) payable public {
        if (a == 65) {
            revert();
        }
    }
}
'''

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

contract_account.f(m.SValue, caller=user_account) # Call f(a), with a symbolic value

## Check if an execution ends with a REVERT or INVALID
for state in m.terminated_states:
    last_tx = state.platform.transactions[-1]
    if last_tx.result in ['REVERT','INVALID']:
        print "Error found in f() execution (see %s)"%m.workspace
        m.generate_testcase(state, 'BugFound')
```

```python
# simple.py
from manticore.ethereum import ManticoreEVM

m = ManticoreEVM() # initiate the blockchain
```

Initiate the blockchain

```python
# simple.py
from manticore.ethereum import ManticoreEVM

m = ManticoreEVM() # initiate the blockchain
source_code = '''
pragma solidity^0.4.20;
contract Simple {
    function f(uint a) payable public {
        if (a == 65) {
            revert();
        }
    }
}
'''

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

Initiate the accounts

```
# simple.py
from manticore.ethereum import ManticoreEVM

m = ManticoreEVM() # initiate the blockchain
source_code = '''
pragma solidity^0.4.20;
contract Simple {
    function f(uint a) payable public {
        if (a == 65) {
            revert();
        }
    }
}
'''

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

contract_account.f(m.SValue, caller=user_account) # Call f(a), with a symbolic value
```

Call f() with a symbolic input

```
# simple.py
from manticore.ethereum import ManticoreEVM

m = ManticoreEVM() # initiate the blockchain
source_code = '''
pragma solidity^0.4.20;
contract Simple {
    function f(uint a) payable public {
        if (a == 65) {
            revert();
        }
    }
}
'''

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

contract_account.f(m.SValue, caller=user_account) # Call f(a), with a symbolic value

## Check if an execution ends with a REVERT or INVALID
for state in m.terminated_states:
    last_tx = state.platform.transactions[-1]
    if last_tx.result in ['REVERT','INVALID']:
        print "Error found in f() execution (see %s)"%m.workspace
        m.generate_testcase(state, 'BugFound')
```

Find if a path fails

Manticore

```
$ python simple.py
Error found in f() execution (see path/mcore_pkIhCq)

$ cat mcore_pkIhCq/BugFound_00000000.tx
Transactions Nr. 0
Type: Create [...]

Transactions Nr. 1
Type: Call
From: 0xd30a286ec6737b8b2a6a7b5fbb5d75b895f62956
To: 0x1bfa530d5d685155e98cd7d9dd23f7b6a801cfef
Value: 0
Data: b3de648b0000000000000000000000000000000000000000000000000000000000000041 (*)
Return_data:

Function call:
f(65) -> REVERT (*)
return: ()
```

Invalid path found!

Manticore

- Auditors: Automatically find vulnerabilities
- Developers: Enhanced unit-tests

Exercise 1

Can an Attacker Steal the Contract's Balance?

```
pragma solidity ^0.4.20;
contract UnprotectedWallet{
    address public owner;

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

    function UnprotectedWallet() public {
        owner = msg.sender;
    }

    function changeOwner(address _newOwner)  public {
       owner = _newOwner;
    }

    function deposit() payable public { }

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

Exercise 1: Solution

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

m = ManticoreEVM() # initiate the blockchain
source_code = ''' pragma solidity ^0.4.20; ...'''

# 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)

print "Creator account: 0x%x (%d)"%(creator_account, creator_account)
print "Attacker account: 0x%x (%d)"%(attacker_account, attacker_account)
```

Initialisation

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

m = ManticoreEVM() # initiate the blockchain
source_code = ''' pragma solidity ^0.4.20; ...'''

# 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)

print "Creator account: 0x%x (%d)"%(creator_account, creator_account)
print "Attacker account: 0x%x (%d)"%(attacker_account, attacker_account)

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

Deposit of 1 ether

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

m = ManticoreEVM() # initiate the blockchain
source_code = ''' pragma solidity ^0.4.20; ...'''

# 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)

print "Creator account: 0x%x (%d)"%(creator_account, creator_account)
print "Attacker account: 0x%x (%d)"%(attacker_account, attacker_account)

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

```
# 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)
```

Two transactions from the attacker

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

m = ManticoreEVM() # initiate the blockchain
source_code = ''' pragma solidity ^0.4.20; ...'''

# 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)

print "Creator account: 0x%x (%d)"%(creator_account, creator_account)
print "Attacker account: 0x%x (%d)"%(attacker_account, attacker_account)

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

```
# 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)
```

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

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

    if solver.check(state.constraints):
        print "Attacker can steal the ether! see
%s"%m.workspace
        m.generate_testcase(state, 'WalletHack')
```

Attacker's balance >1 wei

Exercice 1: Solution

```
$python unprotectedWallet.py
Creator account: 0xd30a286ec6737b8b2a6a7b5fbb5d75b895f62956
(1204823582282099840624073347142121973792277670230)
Attacker account: 0x1bfa530d5d685155e98cd7d9dd23f7b6a801cfef
(1597251719794392811754872930582220176691446292 31)
Attacker can steal the ether! see /path/mcore_XXXX

$cat /path/mcore_XXXX/WalletHack_XXXX.tx

[..]

From: 0x1bfa530d5d685155e98cd7d9dd23f7b6a801cfef
changeOwner(1597251719794392811754872930582220176691446292 31L)

[..]

From: 0x1bfa530d5d685155e98cd7d9dd23f7b6a801cfef
withdraw()
```

```
pragma solidity ^0.4.20;
contract UnprotectedWallet{
    address public owner;

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

    function UnprotectedWallet() public {
        owner = msg.sender;
    }

    function changeOwner(address _newOwner)  public {
      owner = _newOwner;
    }

    function deposit() payable public { }

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

Exercise 2

Is an Integer Overflow Possible?

```
pragma solidity^0.4.20;
contract Overflow {
    uint public sellerBalance=0;

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

Exercise 2: Solution

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

m = ManticoreEVM() # initiate the blockchain
source_code = '''
pragma solidity^0.4.20;
contract Overflow {
    uint public sellerBalance=0;

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

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

Initialisation

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

m = ManticoreEVM() # initiate the blockchain
source_code = '''
pragma solidity^0.4.20;
contract Overflow {
    uint public sellerBalance=0;

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

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

#First add won't overflow uint256 representation
contract_account.add(m.SValue, caller=user_account)
#Potential overflow
contract_account.add(m.SValue, caller=user_account)
contract_account.sellerBalance(caller=user_account)
```

Call add() two times
Call sellerBalance()

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

m = ManticoreEVM() # initiate the blockchain
source_code = '''
pragma solidity^0.4.20;
contract Overflow {
    uint public sellerBalance=0;

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

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

#First add won't overflow uint256 representation
contract_account.add(m.SValue, caller=user_account)
#Potential overflow
contract_account.add(m.SValue, caller=user_account)
contract_account.sellerBalance(caller=user_account)
```

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

    # last_return is the data returned
    last_return = state.platform.last_return_data
    # First input (first call to add)
    input0 = state.input_symbols[0]

    # retrieve last_return and input0 in a similar format
    last_return = Operators.CONCAT(256, *last_return)
    # starts at 4 to skip function id
    input0 = Operators.CONCAT(256, *input0[4:36])
```

Retrieve the last_return and the input in a similar format

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

m = ManticoreEVM() # initiate the blockchain
source_code = '''
pragma solidity^0.4.20;
contract Overflow {
    uint public sellerBalance=0;

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


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

#First add won't overflow uint256 representation
contract_account.add(m.SValue, caller=user_account)
#Potential overflow
contract_account.add(m.SValue, caller=user_account)
contract_account.sellerBalance(caller=user_account)
```

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

    # last_return is the data returned
    last_return = state.platform.last_return_data
    # First input (first call to add)
    input0 = state.input_symbols[0]

    # retrieve last_return and input0 in a similar format
    last_return = Operators.CONCAT(256, *last_return)
    # starts at 4 to skip function id
    input0 = Operators.CONCAT(256, *input0[4:36])

    state.constrain(input0 > last_return)

    if solver.check(state.constraints):
        print "Overflow found, see %s"%m.workspace
        m.generate_testcase(state, 'OverflowFound')
```

Add constraint input0 > sellerBalance

Exercice 2: Solution

```
$python overflow.py
Overflow found! see /path/mcore_XXXX


$cat /path/mcore_XXXX/OverflowFound_XXXX.tx

[..]
add(75988587087560630658257033252266325728079589706855245644198287161206004945024L)
[..]
add(434220334657312264981048270166765350409365066187129137809812790934787714049 28L)
[..]
sellerBalance() -> RETURN
return: 36185313159756617327908752602549529157461116599275953857219822467716467 10016L

Overflow found!
```

Conclusions

Conclusions

- Symbolic execution is a great tool to find bugs
  - We use it on our internal audits, found deeply hidden bugs
- Manticore can be integrated into your development process!
  - More complete exploration than classic unit tests
  - Deepstate integration coming soon

Manticore Github

https://github.com/trailofbits/manticore

trailofbits/manticore

manticore - Dynamic binary analysis tool

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