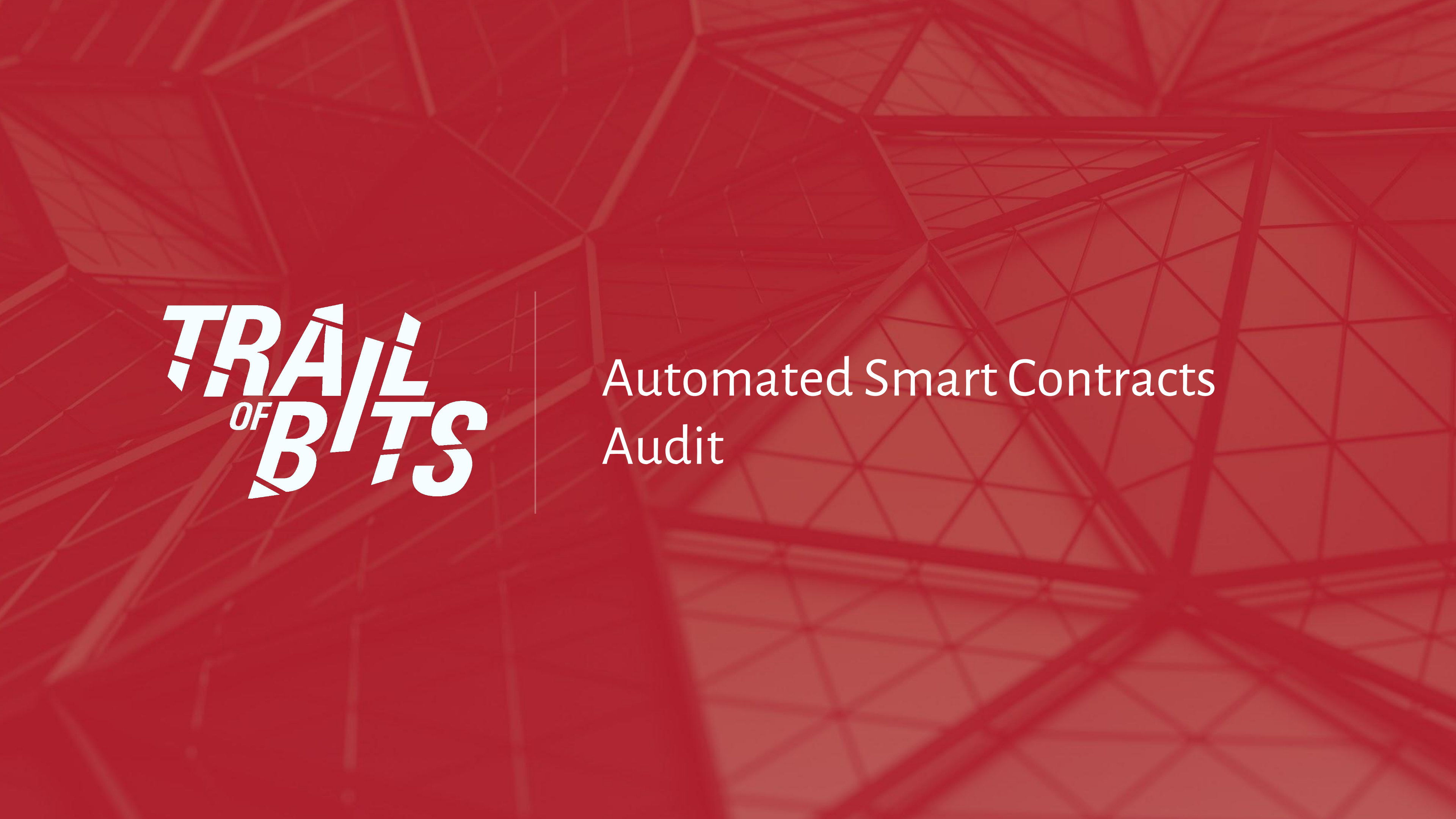


TRAIL OF BITS

Automated Smart Contracts Audit

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 fuzzing?
- What is Symbolic Execution?
- How can they help build more secure smart contracts?
- Hands-on with Echidna and Manticore

Before Starting

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

Automated Smart Contracts Audit

Problem: How to Find Bugs?

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

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

Problem: How to Find Bugs?

- **Manual review: time-consuming, every modification of the contract may may introduce a bug**
  - Contact a security company
- **Unit tests: cover a small part of the program's behavior**
  - Use Truffle

⇨ **Time consuming, usually low coverage**

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

- **Static analysis (e.g. Slither)**
    - Analyze possible behaviors without executing the program
    - Variety of techniques: pattern matching to formal verification
    - Out of scope today

Echidna: Property Based Testing

Echidna

- **Property based testing**
  - You write a property, Echidna tries to break it

Echidna: Example

```
// Anyone can have at maximum 1000 tokens
// The tokens cannot be transferred (not ERC20)

mapping(address => uint) public balances;

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

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

Echidna: Example

```
// Anyone can have at maximum 1000 tokens
// The tokens cannot be transferred (not ERC20)

mapping(address => uint) public balances;

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

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

- **Property: balances(msg.sender) <= 1000**

Echidna: How To Use it

- Write the property in Solidity:

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

Echidna: How To Use it

- Let Echidna check the property

Echidna: Example

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

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

  ✗ 1 failed.
```

Echidna: Example

- Discover a hidden function:

```
// ...

function backdoor() public {

    balances[msg.sender] += 1;
}

//  ...
```

Echidna: Exercises

Exercises

- Open workshops/Automated Smart Contracts Audit - TruffleCon 2018/echidna/exercises.pdf (git clone https://github.com/trailofbits/publications/)

Token

```
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;
   }
}
```

Echidna: Exercises Solutions

Exercise 1

```
contract TestToken is Token {

   address echidna_caller = 0x00a329c0648769a73afac7f9381e08fb43dbea70;


   constructor() public {
       balances[echidna_caller] = 10000;
   }

   // add the property

}
```

Exercise 1

```
contract TestToken is Token {

   address echidna_caller = 0x00a329c0648769a73afac7f9381e08fb43dbea70;


   constructor() public {
       balances[echidna_caller] = 10000;
   }

   function echidna_test_balance() view public returns(bool) {
       return balances[echidna_caller] <= 10000;
   }
}
```

Exercise 1

```
$ echidna-test exercise1_solution.sol TestToken
...

  ✗ "echidna_test_balance" failed after 9 tests and 15 shrinks.

    │ Call sequence: transfer(0,10001);

  ✗ 1 failed.
```

Exercise 2

```
contract TestToken is Token {
   address echidna_caller = 0x00a329c0648769a73afac7f9381e08fb43dbea70;

   constructor() {
       paused();
       owner = 0x0; // lose ownership
   }

   // add the property
}
```

Exercise 2

```
contract TestToken is Token {
   address echidna_caller = 0x00a329c0648769a73afac7f9381e08fb43dbea70;

   constructor() {
       paused();
       owner = 0x0; // lose ownership
   }

   function echidna_no_transfer() view returns(bool) {
       return is_paused == true;
   }
}
```

Exercise 2

```
$ echidna-test exercise2_solution.sol TestToken
...

  ✗ "echidna_no_transfer" failed after 44 tests and 2 shrinks.

    │ Call sequence: Owner();
    │                resume();

  ✗ 1 failed.
```

Exercise Bonus

```
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;

   }
}
```

Exercise Bonus

```
contract Test is MintableToken {

   address echidna_caller = 0x00a329c0648769a73afac7f9381e08fb43dbea70;
   function Test() MintableToken(10000) {
       owner = echidna_caller;
   }

   function echidna_test_balance() view public returns(bool) {
       return balances[msg.sender] <= 10000;
   }

}
```

Exercise Bonus

```
$ echidna-test bonus_solution.sol TestToken
...

  ✗ "echidna_test_balance" failed after 100 tests and 129
shrinks.

    │ Call sequence:
mint(578960446186580977117854925043439539266349923328202820197287
92003956564819968);

  ✗ 1 failed.
```

Echidna: Summary

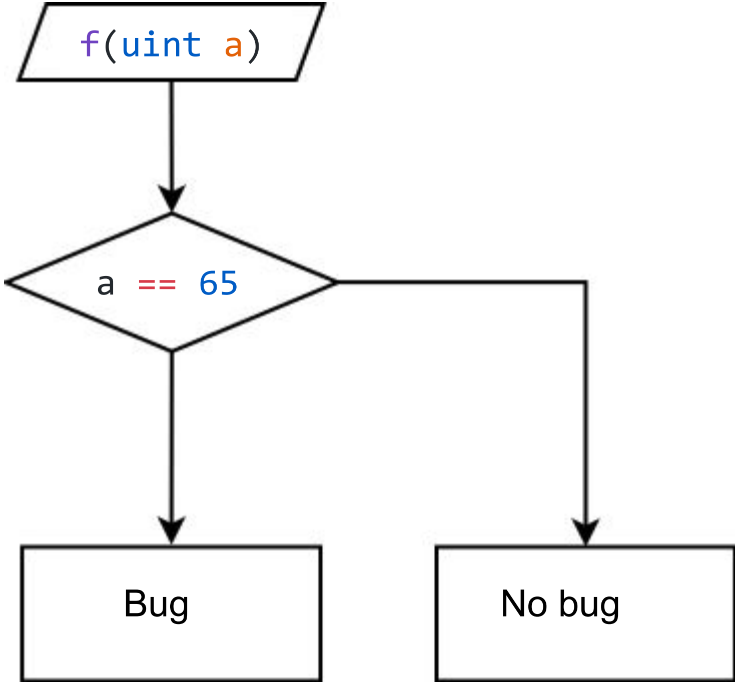
- Echidna will automatically test your code
- No complex setup, properties written in Solidity
- You can integrate Echidna into your development process!

Symbolic Execution

Symbolic Execution in a Nutshell

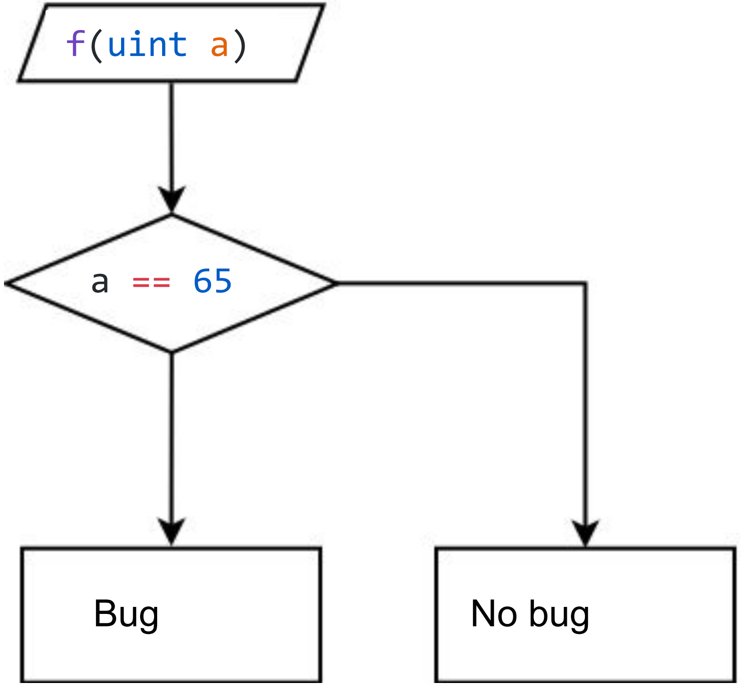
- **Program exploration technique**
- **Execute the program “symbolically”**
  - Represent executions as logical formulas
- **Use an SMT solver to check the feasibility of a path and generate inputs**

Symbolic Execution Example

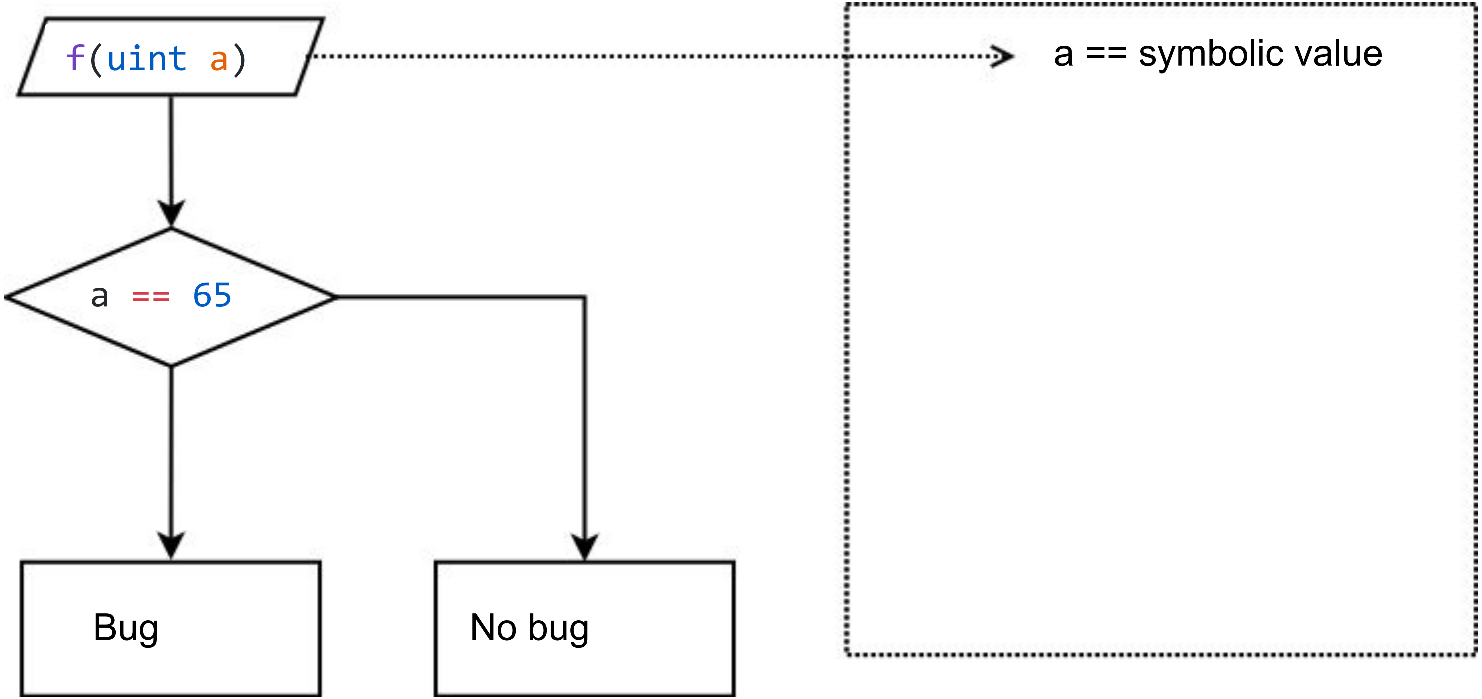


```
contract Simple {
   function f(uint a) payable public {
       // lot of paths and conditions
       if (a == 65) {
           // bug here
       }
   }
}
```

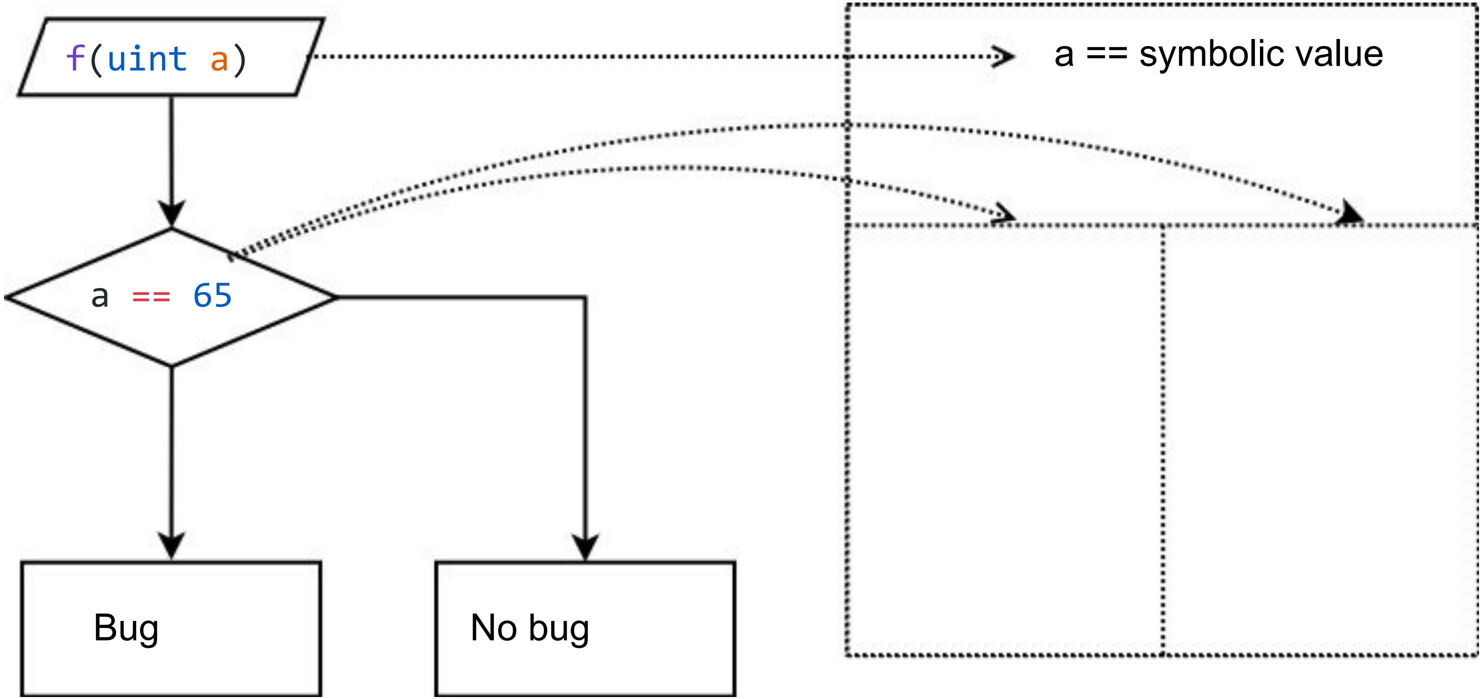
Symbolic Execution Example



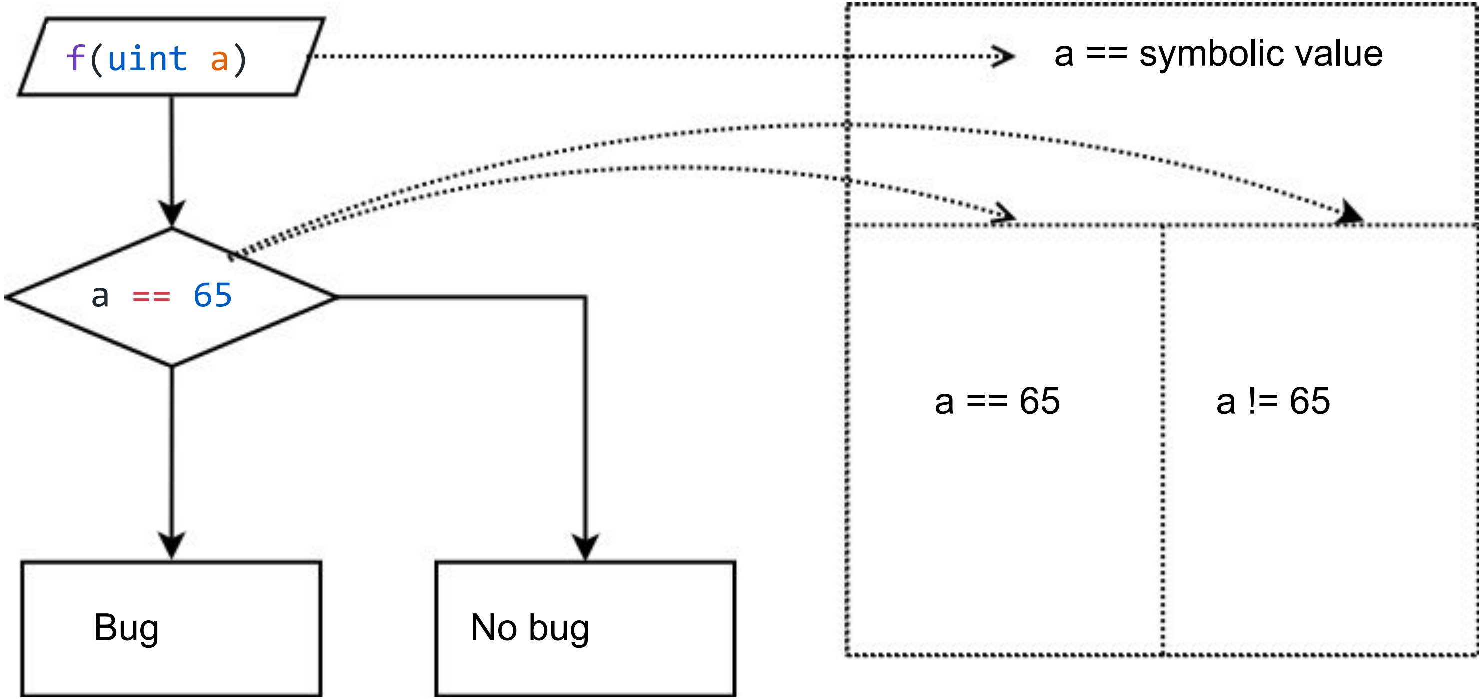
Symbolic Execution Example



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

- Verify that f() reverts only if var == 65

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

Manticore: Python API

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

m = ManticoreEVM()
with open('example.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)
```

Initialization

Manticore: Python API

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

m = ManticoreEVM()
with open('example.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)

symbolic_var = m.make_symbolic_value()
contract_account.f(symbolic_var)
```

Generate a symbolic input
Call f with the symbolic input

Manticore: Python API

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

m = ManticoreEVM()
with open('example.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)

symbolic_var = m.make_symbolic_value()
contract_account.f(symbolic_var)

for state in m.terminated_states:
   last_tx = state.platform.transactions[-1]
   if last_tx.result in ['REVERT', 'INVALID']:
       state.constrain(symbolic_var != 65)
       if solver.check(state.constraints):
           print("Bug found in {}".format(m.workspace))
           m.generate_testcase(state, 'BugFound')
```

Explore the states explored. There is a bug if on a reverted state, var != 65

Manticore: Python API

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

m = ManticoreEVM()
with open('example.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)

symbolic_var = m.make_symbolic_value()
contract_account.f(symbolic_var)

for state in m.terminated_states:
   last_tx = state.platform.transactions[-1]
   if last_tx.result in ['REVERT', 'INVALID']:
       state.constrain(symbolic_var != 65)
       if solver.check(state.constraints):
           print("Bug found in {}".format(m.workspace))
           m.generate_testcase(state, 'BugFound')
```

Manticore: Python API

```
contract Simple {
   function f(uint a) payable public {
       if (a == 65) {
           revert();
       }
       if (a == 64) {
           revert();
       }
   }
}
```

```
$ python3 simple.py
Bug found /home/manticore/examples/mcore_q3csvx7t
$ cat mcore_q3csvx7t/BugFound_00000000.tx
...
f(64) -> REVERT (*)
```

Manticore: Exercise

Is an Integer Overflow Possible?

- Open workshops/Automated Smart Contracts Audit - TruffleCon 2018/manticore/exercises.pdf (https://github.com/trailofbits/trufflecon)

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 Solution

Solution

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

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

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

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
   # retrieve last_return and input0 in a similar format
   last_return = Operators.CONCAT(256, *last_return)

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

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

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

...

add(60661326726858329439570428285975556647751607463109167504653840941059568861185)
-> RETURN (*)

add(69672080359326334380633291372539722228333936369746749109609793890948973854721)
-> RETURN (*)

sellerBalance() -> RETURN

return:
14541317848868468396632734649827371022815559167215352574806050824095413075970 (*)
```

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

Workshop Summary

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