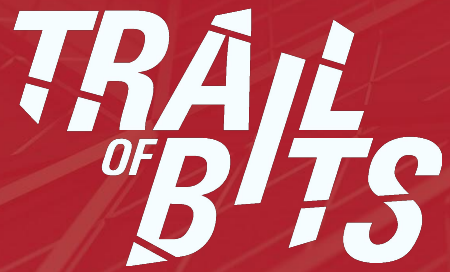


How to Build Secure Smart Contracts: A Deep Dive into Automated Tools

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

- Basic introduction to program analysis
- What are the tools to write secure code
- How to use these tools
- Hands-on with Slither, Echidna and Manticore

Before Starting

- git clone https://github.com/trailofbits/trufflecon-2019/
- docker pull trailofbits/eth-security-toolbox

Program Analysis

Problem: How to Find Bugs?

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

```
/// @notice Allow users to buy token. 1 ether = 10 tokens
/// @param tokens The numbers of token to buy
/// @dev Users can send more ether than token to be bought, to give gifts to the team.
function buy(uint tokens) public payable{
    uint required_wei_sent = (tokens / 10) * decimals;
    require(msg.value >= required_wei_sent);
    balances[msg.sender] = balances[msg.sender].add(tokens);
    emit Mint(msg.sender, tokens);
}
```

Problem: How to Find Bugs?

- **Manual review**
  - Can detect any bug
  - Time-consuming
  - Difficult
  - Do not track code change
  - **Contact security company**

- **Unit tests**
  - Track code change
  - Usually only cover “good” behaviors
  - Cover only a small part
  - **Use Truffle**

Program Analysis

- **Automatic bugs detection and code verification**
  - We will cover 3 types
  - **Static Analysis**: Slither
  - **Fuzzing**: Echidna
  - **Symbolic Execution**: Manticore

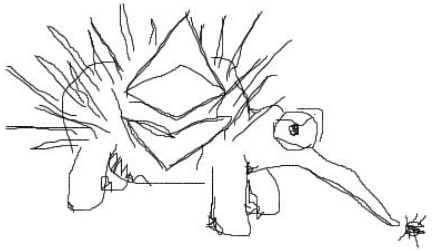
Finding Bugs With Automated Analysis

- **Static analysis (e.g. Slither)**
  - All the program's paths are approximated and analyzed
  - Fast
  - In-built detectors (>60, ~30 public)
  - Today: Custom API

< crytic >

Finding Bugs With Automated Analysis

- **Fuzzing (e.g. Echidna)**
  - Random transactions to stress the contract: **testing**
  - Successful technique for ‘classic software’ (e.g. AFL, libfuzzer)



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	cli: + API: ++	+
Fuzzing	Echidna	< hour	++	++
Symbolic Execution	Manticore	> hour	+++	+++ (Verification)

Secure Development Workflow

Rules

- **Rule 1: Follow coding best practice**
  - Well-architectured code will be simpler to verify
- **Rule 2: Determine what you want to test**
  - Many components can be tested
  - Each tool has situation where it is best suited for
- **Rule 3: Use the tools from the start of the development**

Rule 1: Follow coding best practice

- Strive for simplicity
- Write small and modular components

Rule 1: Follow coding best practice

buy does two things:

1. Check that the user sent enough ethers
2. Mint the tokens

```
function buy(uint tokens) public payable{
    uint required_wei_sent = (tokens / 10) * decimals;
    require(msg.value >= required_wei_sent);
    balances[msg.sender] = balances[msg.sender].add(tokens);
    emit Mint(msg.sender, tokens);
}
```

Rule 1: Follow coding best practice

Alternative version:

```
function buy(uint tokens) public payable{
    _valid_buy(tokens, msg.value);
    _mint(msg.sender, tokens);
}

function _mint(address addr, uint value) internal{
    balances[addr] = safeAdd(balances[addr], value);
    emit Mint(addr, value);
}

function _valid_buy(uint desired_tokens, uint wei_sent) internal view{
    uint required_wei_sent = (desired_tokens / 10) * decimals;
    require(wei_sent >= required_wei_sent);
}
```

Rule 1: Follow coding best practice

- **The second version allows:**
  - Testing individual components separately
  - Re-use functionalities

Rule 2: Determine what you want to test

- **State machine**
  - Ex: Once the buying period ended, no token can be created
  - Tools: Echidna, Manticore
- **Access control**
  - Ex: Only the owner can call `mint`
  - Tools: Slither (simple setup), Echidna, Manticore (complex setup)
- **Arithmetic operations**
  - No integer overflow
  - Tools: Manticore, Echidna

Rule 2: Determine what you want to test

- **Inheritance correctness**
  - Ex: the function mint must never be overridden
  - Tools: Slither
- **External interactions**
  - Ex: what happen if your external dependency is compromised?
  - Tools: Manticore, Echidna
- **Standard conformance**
  - Ex: you rely on an ERC that require functions to return a boolean
  - Tools: Slither

Rule 2: Determine what you want to test

Component	Tools
State machine	Echidna, Manticore
Access control	Slither, Manticore, Echidna
Arithmetic operations	Manticore, Echidna
Inheritance correctness	Slither
External interactions	Manticore, Echidna
Standard conformance	Slither

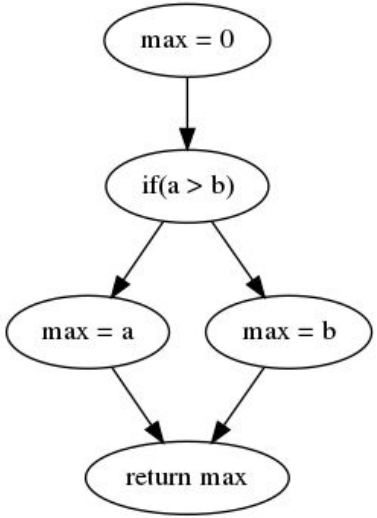
Rule 3: Use the tools from the start

- **Crytic / Slither cli**
  - From the first line written to catch early issues.
- **Echidna and Slither API**
  - As soon as you can determine a property of your contract.
- **Manticore**
  - Once you want to reach an in-depth level of confidence in your code.

Slither: Static Analysis

Static analysis

- **Static analysis**
  - From pattern matching (linter) to formal verification
  - Code representations
    - Ex: control flow graph



Slither

- **Static analysis framework for Solidity**
  - Vulnerability detection
  - Optimization detection
  - Code understanding
  - Assisted code review
- **“LLVM for smart contracts”**

Detectors

- **~30 public vulnerability detectors**
- **From critical issues:**
  - Reentrancy
  - Shadowing
  - Uninitialized variables
  - ...
- **To optimization issues**
  - Variables that should be constant
  - Functions that should be external
  - ...
- **Private detectors with more complex patterns**

Slither Cli

```
tob:$ catc uninitialized.sol
pragma solidity ^0.5.5;

contract Uninitialized{
    address payable destination;

    function buggy() external{
        destination.transfer(address(this).balance);
    }
}
tob:$ slither uninitialized.sol
INFO:Detectors:
Uninitialized.destination (uninitialized.sol#4) is never initialized. It is used in:
        - buggy (uninitialized.sol#6-8)
Reference: https://github.com/trailofbits/slither/wiki/Detectors-Documentation#uninitialized-state-variables
INFO:Slither:uninitialized.sol analyzed (1 contracts), 1 result(s) found
tob:$
```

https://asciinema.org/a/eYrdWBvasHXelpDob4BsNi6Qg

Slither API

- **Python API**
- **Allow to explore every aspect of the contracts**
- **Give access to powerful semantic information**
  - Inbuilt taint and dataflow
  - SlihtIR
  - Out of scope for today

Print Contract's Information

```
from slither import Slither

# Init slither
slither = Slither('coin.sol')
```

Load project

Print Contract's Information

```
from slither import Slither

# Init slither
slither = Slither('coin.sol')

for contract in slither.contracts:
    # Print the contract's name
    print(f'Contract: {contract.name}')
    # Print the name of the contract inherited
    print(f'\tInherits from{[c.name for c in contract.inheritance]}')
```

Iterate over the contracts

Print Contract's Information

```
from slither import Slither

# Init slither
slither = Slither('coin.sol')

for contract in slither.contracts:
    # Print the contract's name
    print(f'Contract: {contract.name}')
    # Print the name of the contract inherited
    print(f'\tInherits from{[c.name for c in contract.inheritance]}')
    for function in contract.functions:
        # For each function, print basic information
        print(f'\t{function.full_name}:')
        print(f'\t\tVisibility: {function.visibility}')
        print(f'\t\tContract: {function.contract}')
        print(f'\t\tModifier: {[m.name for m in function.modifiers]}')
        print(f'\t\tIs constructor? {function.is_constructor}')
```

Print functions information

Slither: Exercises

Exercise 1

- https://github.com/trailofbits/trufflecon-2019
- **slither/slither.pdf**
- **Goal: Function overridden protection**
  - C++ `final` but for Solidity!

Exercise 2: Bonus

- https://github.com/trailofbits/trufflecon-2019
- slither/slither.pdf
- Goal: Conservative access control

Slither: Summary

- Slither will automatically detect most of the common bugs
- It's API can be used to create complex setup
- Try https://crytic.io to have GitHub integration

Echidna: Property Based Testing

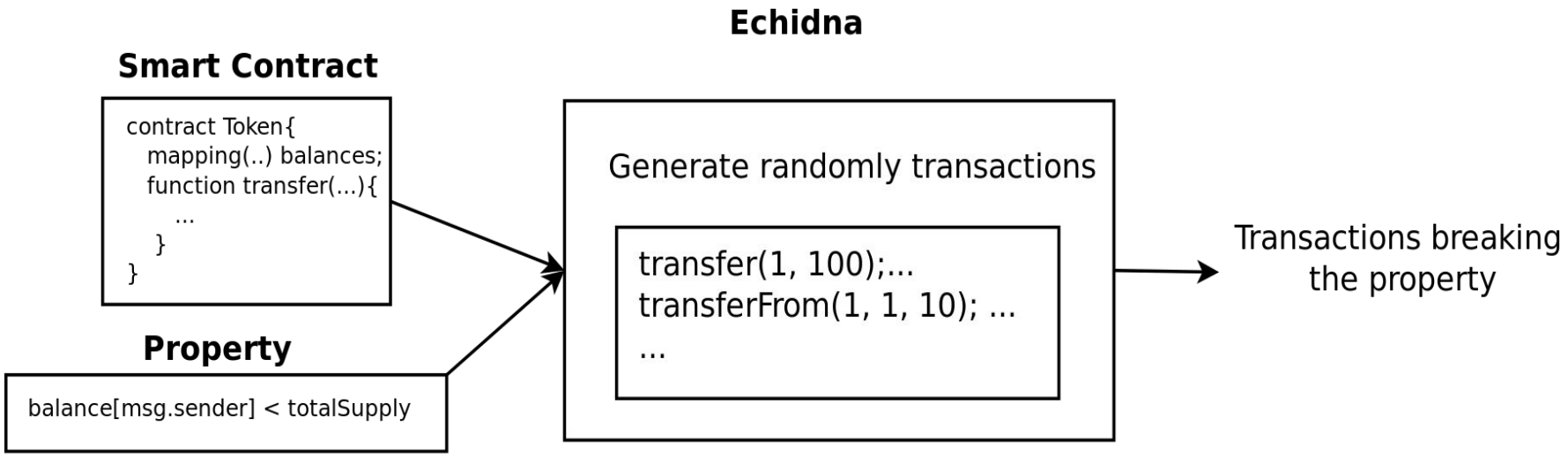
Echidna

- **Fuzzing**
  - Echidna explores the contract with random inputs

Echidna

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

Echidna


Echidna
Smart Contract
contract Token{
mapping(..) balances;
function transfer(...){
...
}
}
Property
balance[msg.sender] < totalSupply
Generate randomly transactions
transfer(1, 100);...
transferFrom(1, 1, 10); ...
...
Transactions breaking the property

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

Echidna: Example

- Discover a hidden function:

```
// ...
function backdoor() public {
    balances[msg.sender] += 1;
}
// ...
```

Echidna: Exercises

Exercise 1

- https://github.com/trailofbits/trufflecon-2019
- echidna/echidna.pdf
- Goal: check the correct access contract of the token

First: try without the template!

Exercise 2: Bonus

- https://github.com/trailofbits/trufflecon-2019
- echidna/echidna.pdf
- Goal: check the correct arithmetic

First: try without the template!

Echidna: Summary

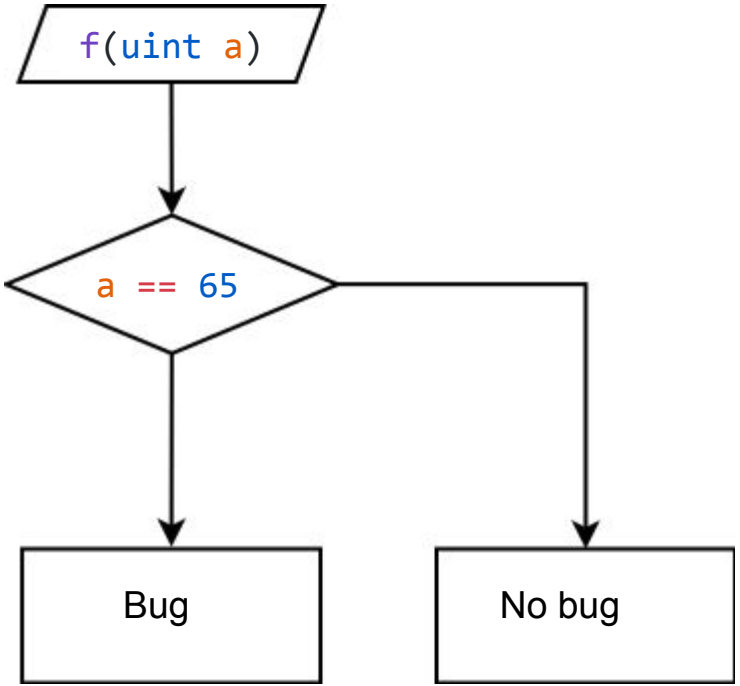
- Echidna will automatically test your code
- No complex setup, properties written in Solidity
- You can integrate Echidna into your development process!

Symbolic Execution

Symbolic Execution in a Nutshell

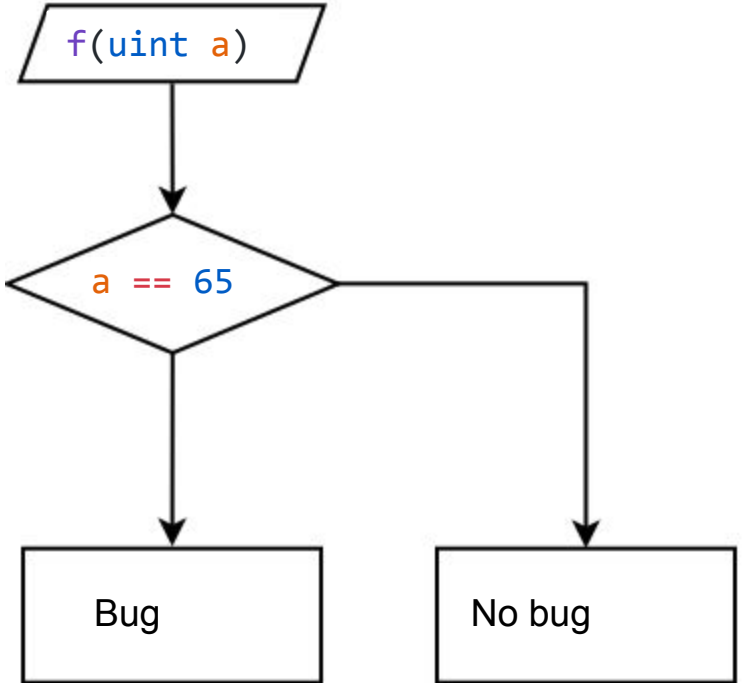
- **Program exploration technique**
- **Execute the program “symbolically”**
  - Represent executions as logical formulas
  - Fork on each condition
- **Use an SMT solver to check the feasibility of a path and generate inputs**

Symbolic Execution Example

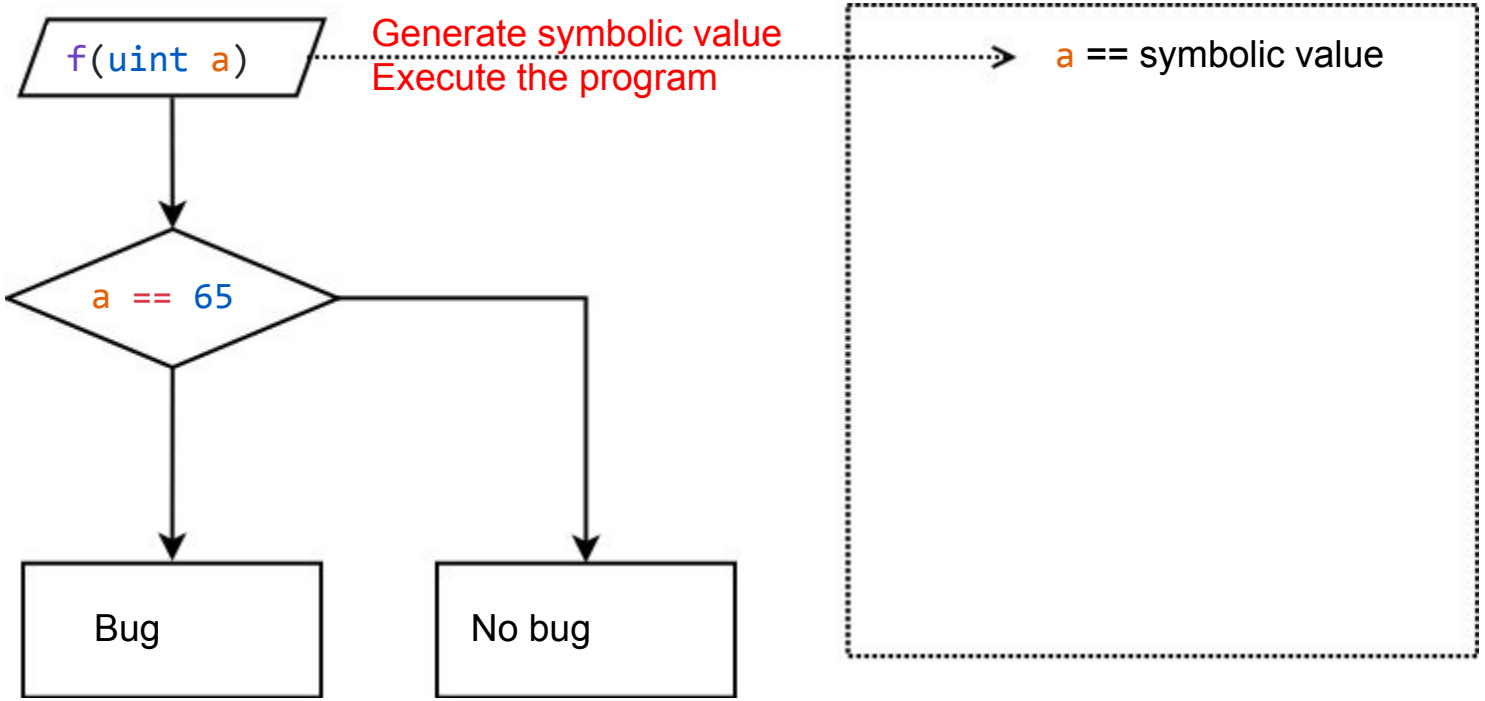



```
contract Simple {
   function f(uint a) payable public {
       // lot of paths and conditions
       if (a == 65) {
           // bug here
       }
   }
}
```

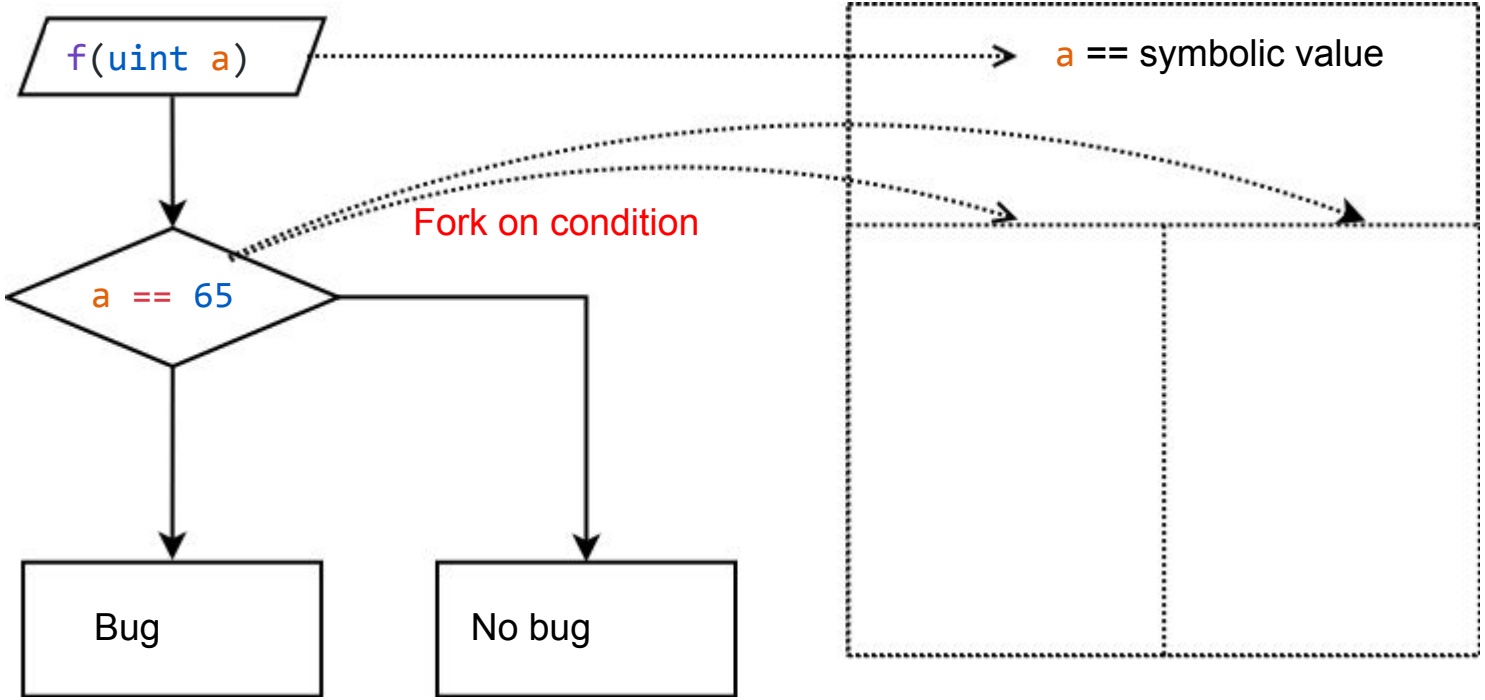
Symbolic Execution Example



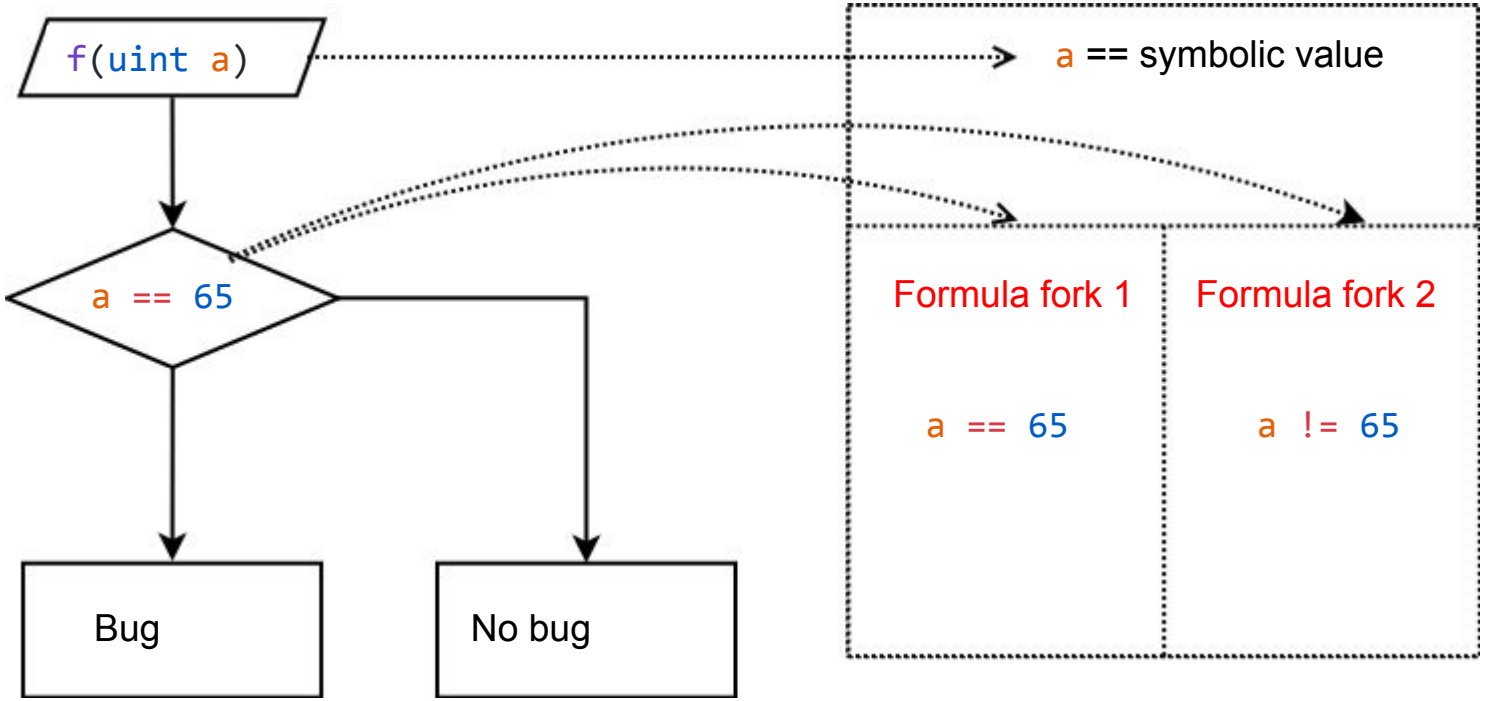
Symbolic Execution Example



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

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
```

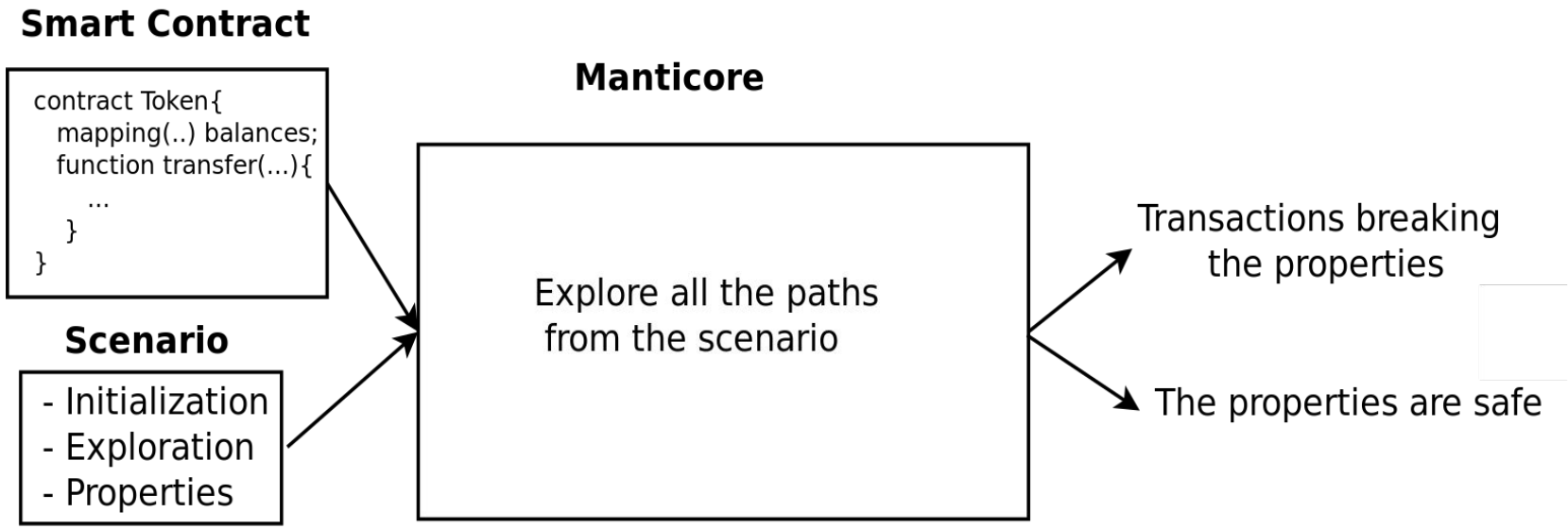
Manticore: Command Line

```
$ 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

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

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)
```

```
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

```
# Explore all the forks
for state in m.ready_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)


    # Check if it is possible to have balance_after > balance_before
    condition = Operators.UGT(balance_after, balance_before)
    if m.generate_testcase(state, name="BugFound", only_if=condition):
        print("Bug found! see {}".format(m.workspace))
```

Bug found if:

balance_after(sender) > balance_before(sender)

Bug found!

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



balances(..) -> 100

transfer(...,20430840703553386272388160528996790065041473555354846411818661786570194
945)

balances(..)
->115771658396612642037298596848158911063204943192085209193045765346126559445091
```

Bug found!

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

Manticore: Exercise

Exercise 1

- https://github.com/trailofbits/trufflecon-2019
- manticore/manticore.pdf
- Goal: check the correctness of the valid_buy function

First: try without the template!

Exercise 2: Bonus

- https://github.com/trailofbits/trufflecon-2019
- manticore/manticore.pdf
- Goal: arithmetic check with multiple transactions

First: try without the template!

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: Summary

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

Workshop Summary

Workshop Summary

- **crytic.io: CI with access to private code analyzers**
- **Our tools will help you building safer smart contracts**
  - Slither: https://github.com/trailofbits/slither/
  - Echidna: https://github.com/trailofbits/echidna/
  - Manticore: https://github.com/trailofbits/manticore/
- **If you need help: https://empireslacking.herokuapp.com/**
  - #ethereum, #manticore, #crytic
- **Office hours every two weeks (free, on hangout)**