Build Secure Smart Contracts: A Deep Dive into Automated Tools

Trufflecon 2020

Who Am I?

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

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

Goals

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

Before Starting

- **git clone https://github.com/crytic/building-secure-contracts**
  - cd building-secure-contracts
  - git checkout trufflecon2020
- **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
  - Does not track code changes

  → **Contact security company**

- **Unit tests**
  - Track code changes
  - Only cover “good” behaviors (*)
  - Covers only a small part

  → **Use Truffle**

Program Analysis

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

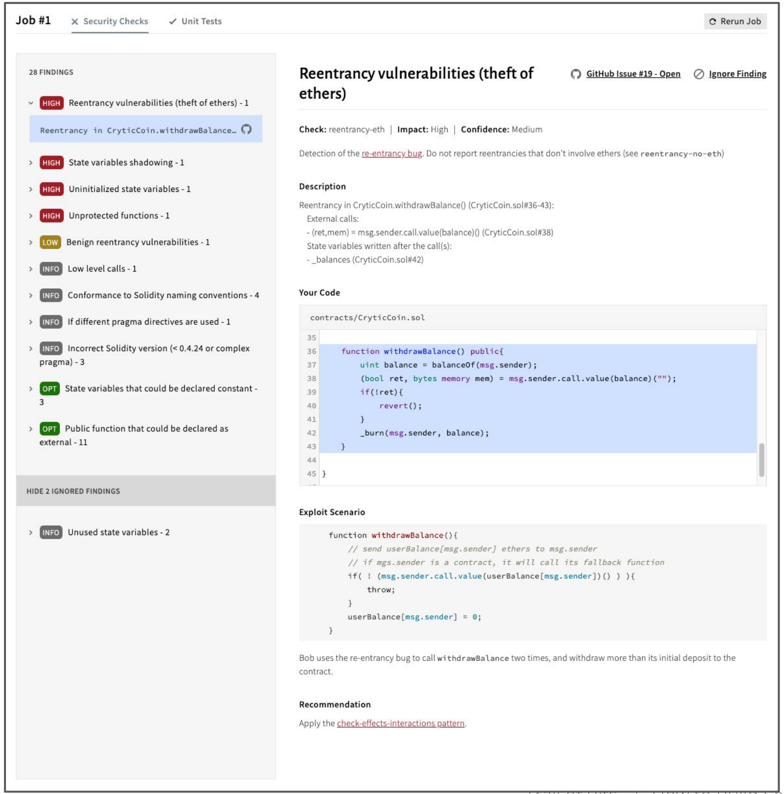
Finding Bugs With Automated Analysis

TRAIL OF BITS

- **Static analysis (e.g. Slither)**
    - All the program's paths are approximated and analyzed
    - Fast
    - Built-in detectors (>90 private, ~40 public)
    - Today: Custom API

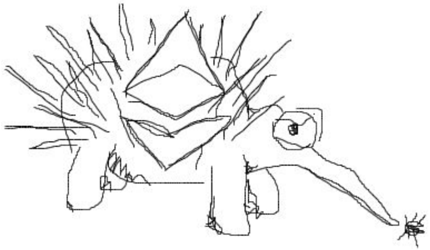
SLITHER

crytic.io


Job #1
Security Checks
Unit Tests
Rerun Job
28 FINDINGS
HIGH Reentrancy vulnerabilities (theft of ethers) - 1
Reentrancy in CryticCoin.withdrawBalance…
HIGH State variables shadowing - 1
HIGH Uninitialized state variables - 1
HIGH Unprotected functions - 1
LOW Benign reentrancy vulnerabilities - 1
INFO Low level calls - 1
INFO Conformance to Solidity naming conventions - 4
INFO If different pragma directives are used - 1
INFO Incorrect Solidity version (< 0.4.24 or complex pragma) - 3
OPT State variables that could be declared constant - 3
OPT Public function that could be declared as external - 11
HIDE 2 IGNORED FINDINGS
INFO Unused state variables - 2
Reentrancy vulnerabilities (theft of ethers)
GitHub Issue #19 - Open
Ignore Finding
Check: reentrancy-eth | Impact: High | Confidence: Medium
Detection of the re-entrancy bug. Do not report reentrancies that don't involve ethers (see reentrancy-no-eth)
Description
Reentrancy in CryticCoin.withdrawBalance() (CryticCoin.sol#36-43):
External calls:
- (ret,mem) = msg.sender.call.value(balance)() (CryticCoin.sol#38)
State variables written after the call(s):
- _balances (CryticCoin.sol#42)
Your Code
contracts/CryticCoin.sol
function withdrawBalance() public{
uint balance = balanceOf(msg.sender);
(bool ret, bytes memory mem) = msg.sender.call.value(balance)("");
if(!ret){
revert();
}
_burn(msg.sender, balance);
}
}
Exploit Scenario
function withdrawBalance(){
// send userBalance[msg.sender] ethers to msg.sender
// if mgs.sender is a contract, it will call its fallback function
if(! (msg.sender.call.value(userBalance[msg.sender])())){
throw;
}
userBalance[msg.sender] = 0;
}
Bob uses the re-entrancy bug to call withdrawBalance two times, and withdraw more than its initial deposit to the contract.
Recommendation
Apply the check-effects-interactions pattern.

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: **verification**

Finding Bugs With Automated Analysis

Technique	Tool	Speed	Complexity	Precision
Static Analysis	Slither	seconds	CLI: + API: ++	+
Fuzzing	Echidna	< hour	++	++
Symbolic Execution	Manticore	> hours	+++	+++ (Verification)

Secure Development Workflow

Rules

- **Rule 1: Follow coding best practices**
  - Well-architectured code is 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
- **Rule 3: Use the tools from the start**

Rule 1: Follow coding best practice

- Strive for simplicity
- Write small and modular components

Rule 1: Follow coding best practices

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 practices

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 practices

- **The second version allows:**
  - Testing individual components separately
  - Code reuse

Rule 2: Determine what you want to test

- **State machine**
  - Ex: Once the buying period ended, no tokens 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 overflows exist
  - 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 requires functions to return a boolean
  - Tools: Slither

Rule 2: Determine what you want to test

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

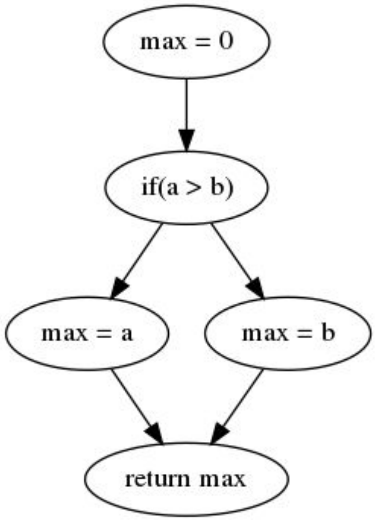
Rule 3: Use the tools from the start

- **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

- ~40 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-varia
bles
INFO:Slither:uninitialized.sol analyzed (1 contracts), 1 result(s) found
tob:$
```

https://asciinema.org/a/eYrdWBvasHXelpDob4BsNi6Qg

Slither API

- **Python API**
- **Allows exploring every aspect of the contracts**
- **Gives access to powerful semantic information**
    - Built-in taint and data-flow analyses
    - “SlithIR”
    - Out of scope for today

Print contract information

```
from slither import Slither

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

Load project

Print contract information

```python
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 information

```python
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/crytic/building-secure-contracts/tree/trufflecon2020/program-analysis/slither
- exercise1.md
- Goal: Function override protection in Solidity 0.5

Exercise 2

- https://github.com/crytic/building-secure-contracts/tree/trufflecon2020/program-analysis/slither
- exercise2.md
- Goal: Conservative access control

Slither: Exercises Solutions

Exercise 1: override protection

```python
from slither.slither import Slither

slither = Slither('coin.sol')
coin = slither.get_contract_from_name('Coin')

# Iterate over all the contracts
for contract in slither.contracts:
  # If the contract is derived from MyContract
  if coin in contract.inheritance:
     # Get the function definition
     mint = contract.get_function_from_signature('_mint(address,uint256)')
     # If the function was not declarer by coin, there is a bug !
     if mint.contract != coin:
          print(f'Error, {contract} overrides {mint}')
```

Exercise 2: conservative access controls

```python
from slither import Slither

slither = Slither('coin.sol')

whitelist = ['balanceOf(address)']

for function in slither.functions:
    if function.full_name in whitelist:
        continue
    if function.is_constructor:
        continue
    if function.visibility in ['public', 'external']:
        if not 'onlyOwner()' in [m.full_name for m in function.modifiers]:
            print(f'{function.full_name} is unprotected!')
```

Slither: Summary

- Slither automatically detects most of common bugs
- Its API can be used to test complex properties
- Try https://crytic.io for integration with GitHub

Echidna: Property Based Testing

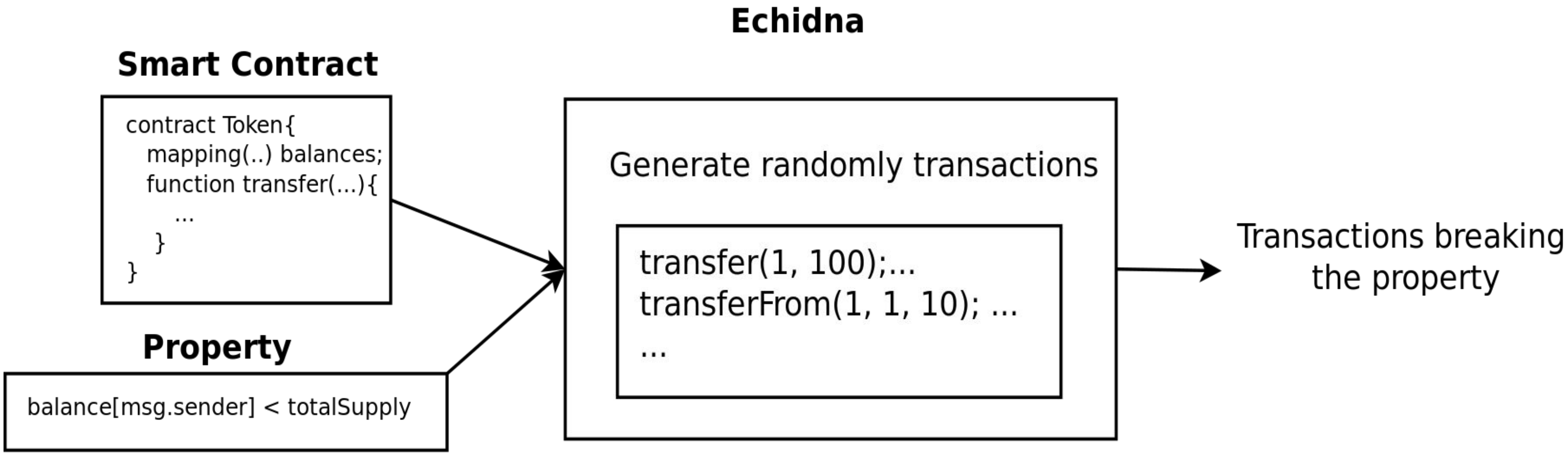
Echidna

- **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(...){
...
}
}
Generate randomly transactions
transfer(1, 100);...
transferFrom(1, 1, 10); ...
...
Transactions breaking
the property
Property
balance[msg.sender] < totalSupply

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/crytic/building-secure-contracts/tree/trufflecon2020/program-analysis/echidna
- Exercise-1.md
- Goal: check for correct arithmetic

First: try without the template!

Exercise 2

- https://github.com/crytic/building-secure-contracts/tree/trufflecon2020/program-analysis/echidna
- Exercise-2.md
- Goal: check for correct access control of the token

First: try without the template!

Echidna: Exercises Solutions

Exercise 1: Test arithmetic

```
contract TestToken is Token {

    address echidna_caller = 0x00a329c0648769a73afac7f9381e08fb43dbea70;


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

    // add the property

}
```

Exercise 1: test arithmetic

```
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: test arithmetic

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

echidna_test_balance: failed!💥
  Call sequence:
    transfer(e7646f3fc82caf8a5e409b9c370f9610c5c19515,16384)
```

Exercise 2: test access controls

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

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

   // add the property
}
```

Exercise 2: test access controls

```
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: test access controls

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

echidna_no_transfer: failed!💥
  Call sequence:
    Owner()
    resume()
```

Echidna: Summary

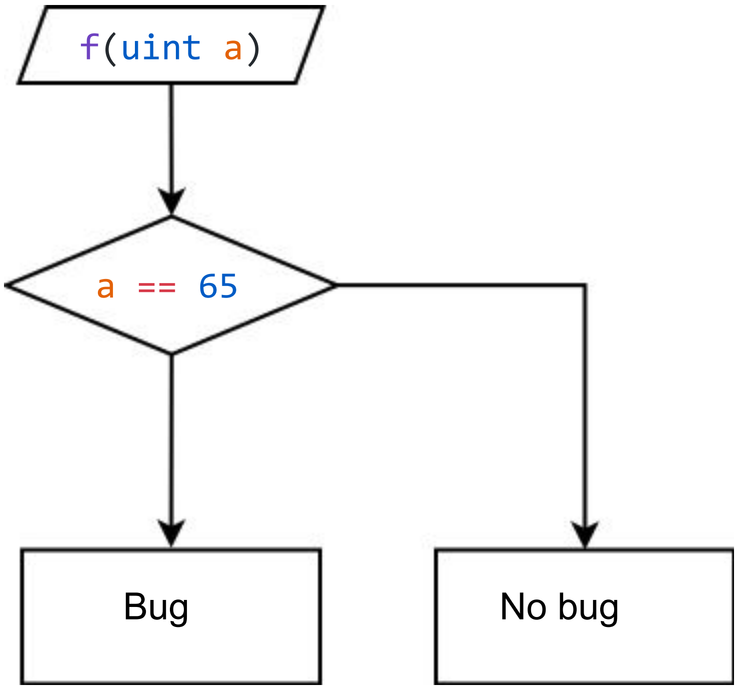
- Echidna will automatically test your code
- No complex setup, properties written in Solidity
- Integrate Echidna tightly with your development process!

Symbolic Execution

Symbolic Execution in a Nutshell

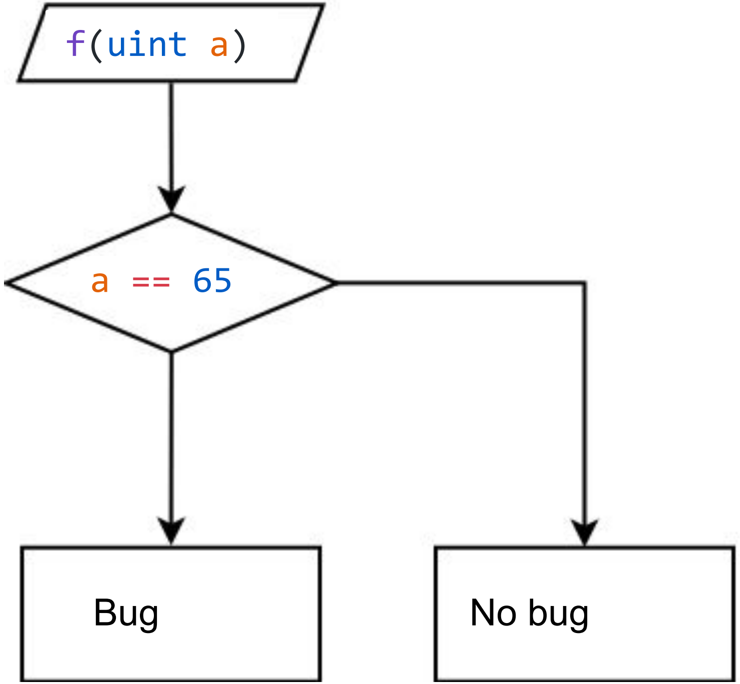
- **Program exploration technique**
- **Execute the program “symbolically”**
  - Represent executions as logical formulas
  - Fork on each condition
- **Use an SMT solver to check the feasibility of a path and generate inputs that reach it**

Symbolic Execution Example

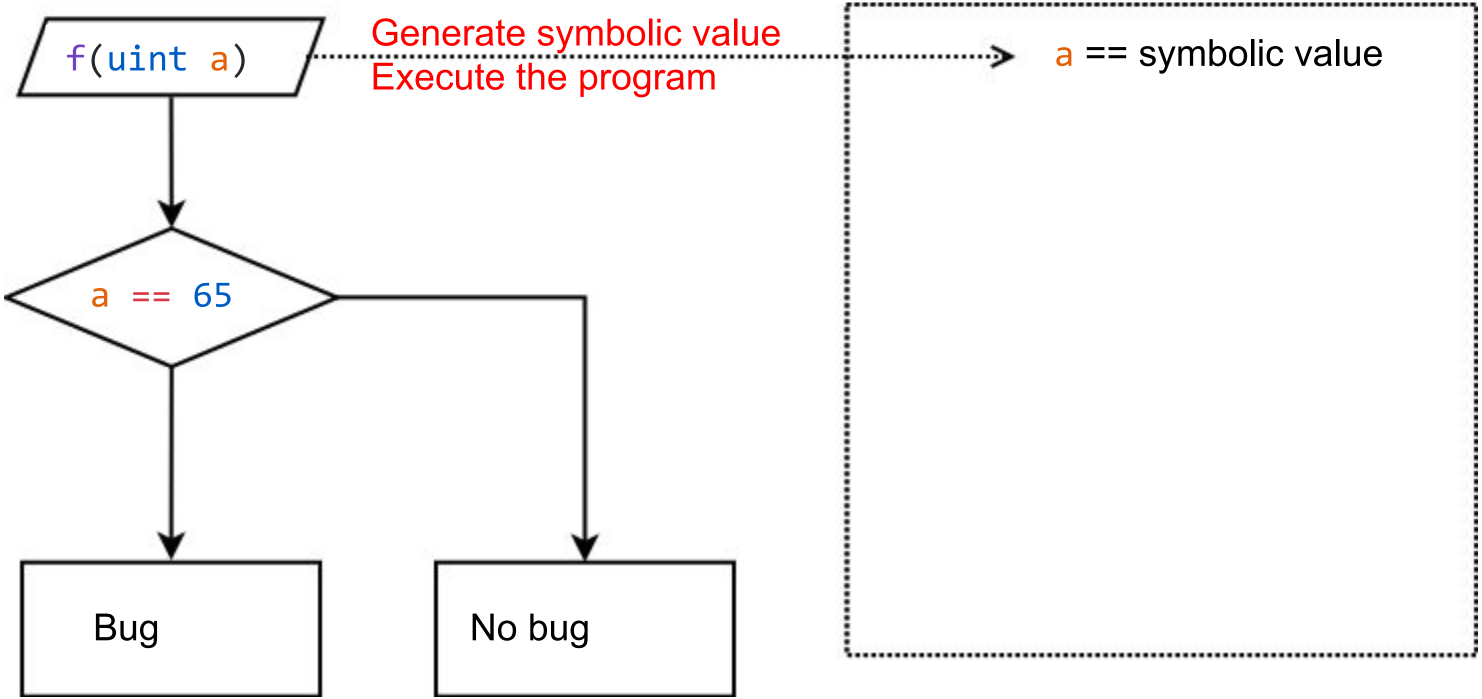



```
contract Simple {
   function f(uint a) payable public {
       // lot of paths and conditions
       if (a == 65) {
           // bug here
       }
   }
}
```

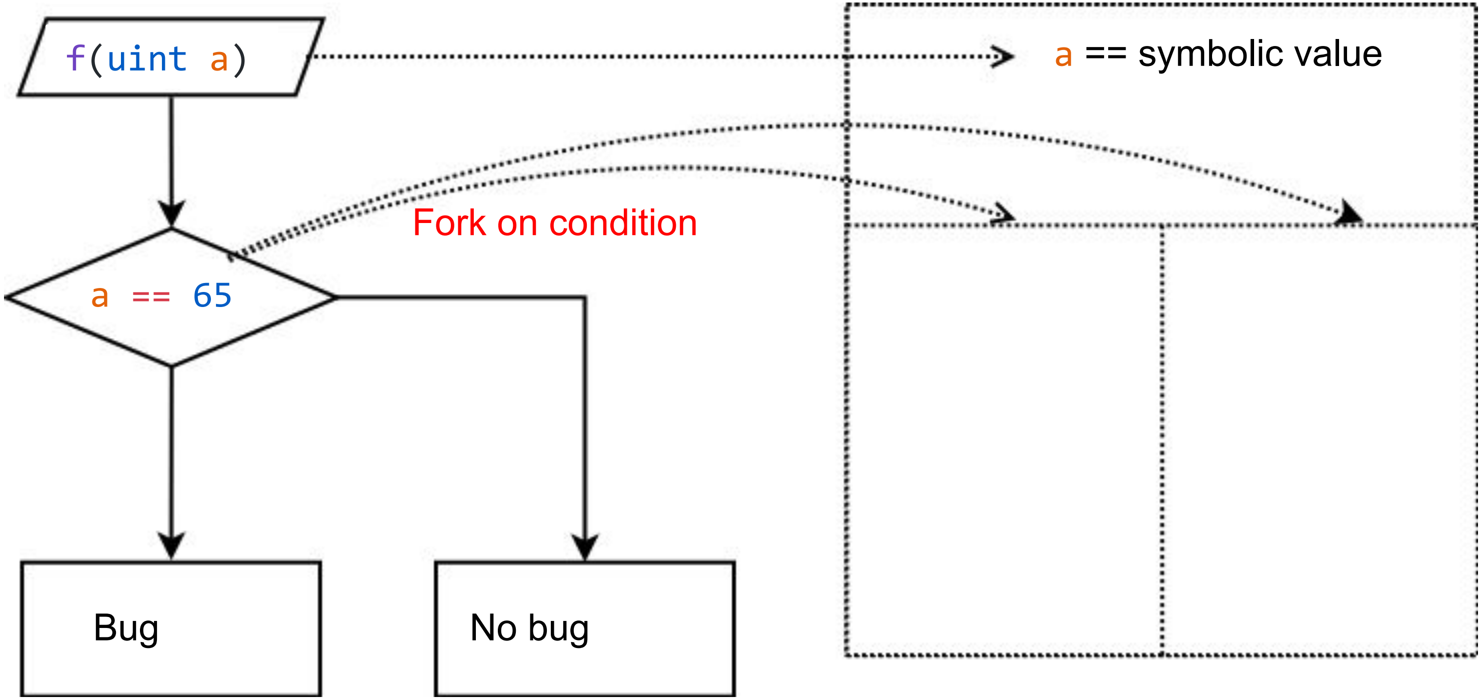
Symbolic Execution Example



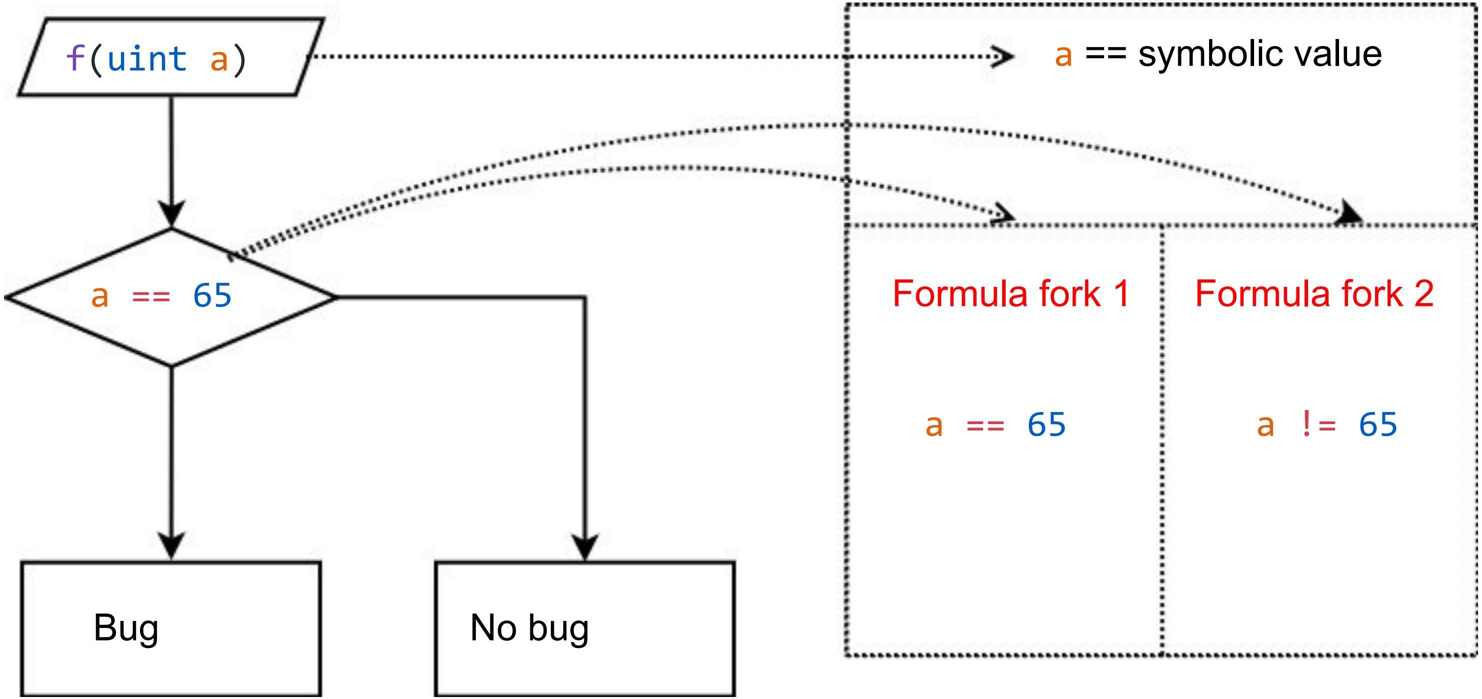
Symbolic Execution Example



Symbolic Execution Example



Symbolic Execution Example



Symbolic Execution in a Nutshell

- Explore the program automatically
- Allows finding unexpected paths
- Optional: Add arbitrary conditions to path exploration

Manticore

Manticore

- **A symbolic execution engine supporting EVM**
- **Built-in 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
```

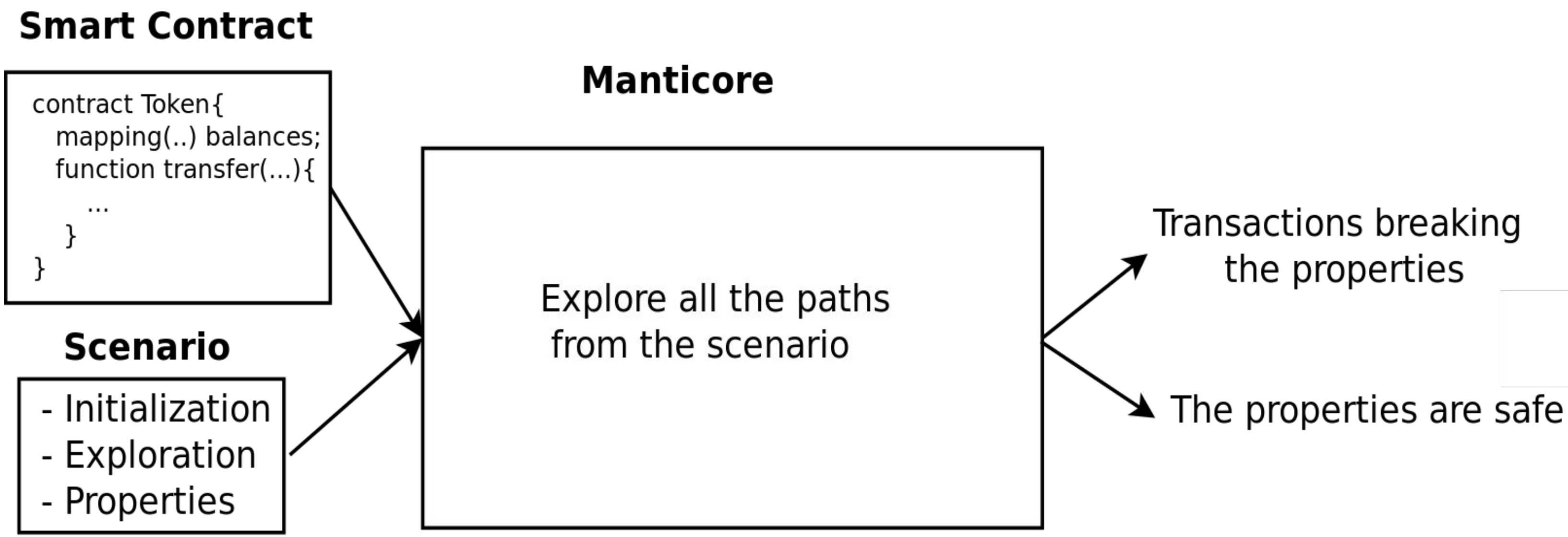
Manticore: Command Line

```
$ 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:**
    a. Initialization: what contracts, how many users?
    b. Exploration: what functions to explore, what is symbolic
    c. 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

```python
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=1*10**18)
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=1*10**18)
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

Bug found if:

balance_after(sender) > balance_before(sender)

```
# 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!

```
$ 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/crytic/building-secure-contracts/tree/trufflecon2020/program-analysis/manticore
- exercises/exercise1.md
- Goal: check the correctness of the valid_buy function

First: try without the template!

Exercise 2

- https://github.com/crytic/building-secure-contracts/tree/trufflecon2020/program-analysis/manticore
- exercises/exercise2.md
- 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: Exercise Solution

Exercise 1: correctness of valid_buy

```python
ETHER = 10**18

m = ManticoreEVM() # initiate the blockchain
# Init
user_account = m.create_account()
with open('token.sol', 'r') as f:
    contract_account = m.solidity_create_contract(f, owner=user_account)
```

Exercise 1: correctness of valid_buy

```python
# Exploration
# Call two times f() with a different symbolic value
tokens_amount = m.make_symbolic_value()
wei_amount = m.make_symbolic_value()

contract_account.is_valid_buy(tokens_amount, wei_amount)

# Property
for state in m.ready_states:

   condition = Operators.AND(wei_amount == 0, tokens_amount >= 1)

   if m.generate_testcase(state, name="BugFound", only_if=condition):
       print(f'Bug found, results are in {m.workspace}')
```

```
$ cat mcore_.../BugFound_00000000.tx
...
Function call:
is_valid_buy(1,0) -> RETURN (*)
return: 1 (*)
```

Exercise 2: verify arithmetic

```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)
```

Exercise 2: verify arithmetic

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


   # last_return is the data returned
   sellerBalance_tx = state.platform.transactions[-1]
   # retrieve last_return and input0 in a similar format
   seller_balance = ABI.deserialize("uint", sellerBalance_tx.return_data)


   condition = Operators.UGT(value_0, seller_balance)


   if m.generate_testcase(state, name="BugFound", only_if=condition):
       print(f'Bug found, results are in {m.workspace}')
```

```
$ 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

Bonus: Manticore-verifier

manticore-verifier

- Verify Solidity properties with Manticore, without using the Python API!
- https://blog.trailofbits.com/2020/07/12/new-manticore-verifier-for-smart-contracts/

Workshop Summary

Workshop Summary

- **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/

Workshop Summary

- crytic.io: CI with access to private code analyzers
- github.com/crytic/building-secure-contracts
  - More exercises in master branch
- If you need help: https://empireslacking.herokuapp.com/
  - #ethereum, #manticore, #crytic
- Try writing an Echidna property for your contracts!