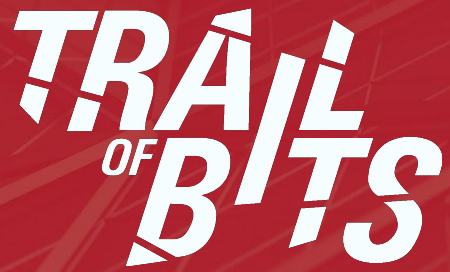


Automated Smart Contract Security

whoami

- 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

What we'll cover today

- Phases of code review, with tools
- Manticore (deep dive)
- Echidna (deep dive)

Tool installation

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 .

Before Starting

- Get stack
  - Google “haskell stack” and follow the directions
- git clone https://github.com/trailofbits/echidna
  - cd echidna
  - stack setup
  - stack install

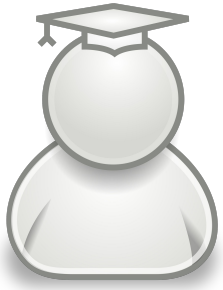
Code Review

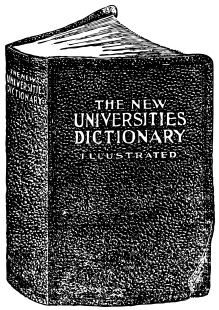
0: Do your research

- Read about real security bugs
- Read about what's new in the ecosystem
- Find application-specific threats
  - Will your oracles work OK with sharding?
  - Are all the ERC20 tokens you wanna trade *really* standard-compliant?
  - What's your model for key safety (for upgradeable contracts)
- Congrats! You're all doing this

(Not So) Smart Contracts

Educational Tool
learn about EVM and Solidity vulnerabilities





Working Examples of Contracts
real vulnerabilities found in the wild

Reference Material
useful when performing audits

https://github.com/trailofbits/not-so-smart-contracts

Community information

Awesome Ethereum Security

- What? A curated list of community-maintained and open-source references.
- Why? Everything in one place: no more searching through stack overflow, github, and reddit
- Features
  - Resources for secure development, CTFs & wargames, and even specific podcast episodes
  - Identifies security tools for visualization, linting, bug finding, verification, and reversing
  - Pointers to related communities

awesome-ethereum-security is open source!
https://github.com/trailofbits/awesome-ethereum-security

Blockchain Security Contacts

- What? A comprehensive list of security contacts for blockchain applications
- Why? Projects worth $10MM+ should have a way to engage with security researchers
- Features
  - Vuln disclosure program best practices
  - Deployed addresses template for dapps
  - Existing contact info for over 100 projects
    - Blockchains, dapps, ERC20 and 721 tokens, Exchanges, Wallet software

Blockchain-security-contacts is open source!
https://github.com/trailofbits/blockchain-security-contacts

Community information

Smart Contract Best Practices

- What? Another curated list of community-maintained and open-source references!
- Why? Slightly more organized than Awesome EVM Security, plus a bit more concise
- Features
  - Great site layout
  - Very digestible
  - New devs can go over in an afternoon

Ethereum Wiki on Safety

- What? Another list of important security considerations, but might not always be up to date
- Why? More resources are always valuable
- Features
  - Lots of gotchas that can be hard to catch
  - Heavy use of example code

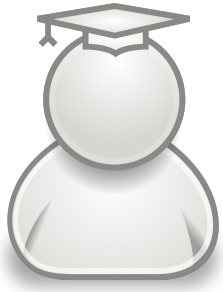
Smart Contract Best Practices is publically available
https://consensys.github.io/smart-contract-best-practices/

This wiki is on github
https://github.com/ethereum/wiki/wiki/Safety

Slightly Smarter Contracts Training

Features

- 1-day, expert instructor-led training course
- Detailed discussion of Solidity vulnerabilities
- Essential background on EVM, solc, SMT, tainting, etc
- 10 exercises for Echidna and Manticore
- Discover both well-known and unique bug classes



Slightly Smarter Contracts starts in October

Inputs

- Human familiar with writing Solidity
- 8 hours
- Laptop
- Motivation

Outputs

- Working knowledge of Solidity security issues
- Working knowledge of advanced security techniques like SMT, tainting, and property testing
- Ability to use Echidna and Manticore “out the box” to detect known flaws
- Ability to write custom modules for Echidna and Manticore to detect application-specific flaws

1: Read the code

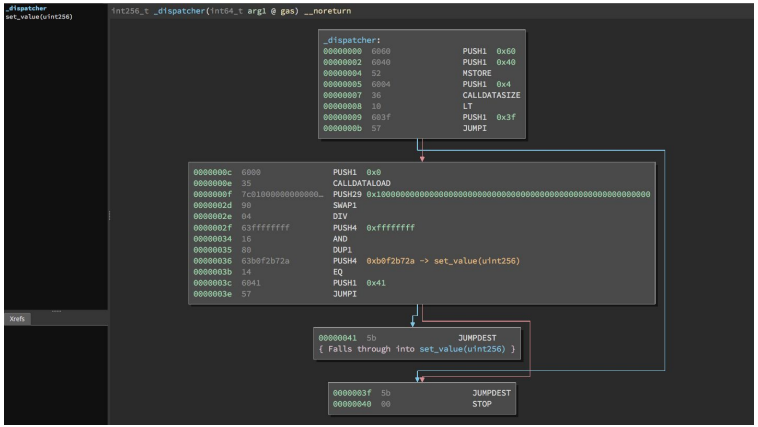
- Careful, mostly manual review
- By far, the most important step
- Prepares you for everything going forwards
- Sometimes, reversing is necessary
- Also the time for linters/simple analysis tools

Reversing tools

Ethersplay: Visual EVM Disassembler

Features

- Advanced control flow recovery
- Extensive function identification database
- Value set analysis of the program stack at each instruction
- Scriptable analysis using Binary Ninja's Python API



Ethersplay is open source!
https://github.com/trailofbits/ethersplay

Inputs

- EVM bytecode (that's it!)

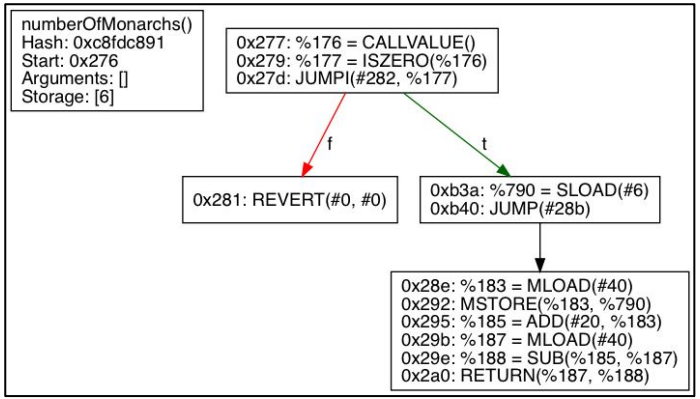
Outputs

- Interactive control flow graph
- Variable tracking
- Labeled functions and arguments
- Possible stack values for each instruction

Rattle: EVM analysis framework

Features

- Recovers EVM control flow graph (CFG)
- Lifts EVM to a single-static-assignment (SSA) IR
- Optimizes and simplifies the lifted code
- Recovers variables
- Generates function CFGs



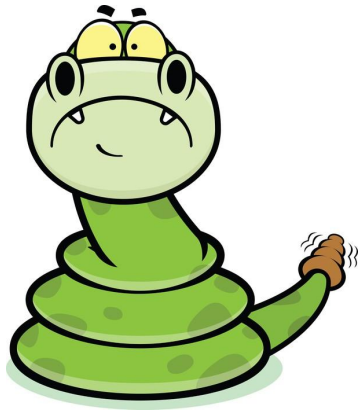
Rattle is open source!
https://github.com/trailofbits/rattle

Inputs

- EVM bytecode (that's it!)

Outputs

- EVM::SSA IR, optimized to remove DUPs, SWAPs, PUSHs, and POPs
- Recovered storage variables, function arguments, memory locations, and storage locations
- Control flow graph diagrams



EVM bytecode analysis and info

pyevmasm

- What? Python library for assembling and disassembling EVM bytecode
- Why? Script disassembly instead of using Remix
- Features
  - CLI tool for quick analysis
  - Supports Python 2 and 3
  - Easy to embed with minimal dependencies
- Manticore and Ethersplay use pyevmasm as a library

pyevmasm is open source!
https://github.com/trailofbits/pyevasm

evm-opcodes

- What? Single page list of all EVM opcodes, descriptions, and gas information
- Why? Speed up reversing and avoid searching reddit, stack overflow, and elsewhere.
- Features
  - Centralized repository collected from the yellow paper, EVM implementations, solidity compiler, Manticore, and more!
  - Links the corresponding Ethereum Improvement Proposals (EIP) of each instruction
  - Community-maintained on Github

evm-opcodes is open source!
https://github.com/trailofbits/evm-opcodes

Linters

Slither: Smart contract static analysis

Features

- Solidity vulnerability detection with low false positives
- Detection of all major smart contract vulnerabilities
- Easily integrated into CI pipeline
- Integrates with Etherscan to obtain contract source
- Support advanced value- and taint-tracking
- Detector API to write custom analyses in Python

Detections

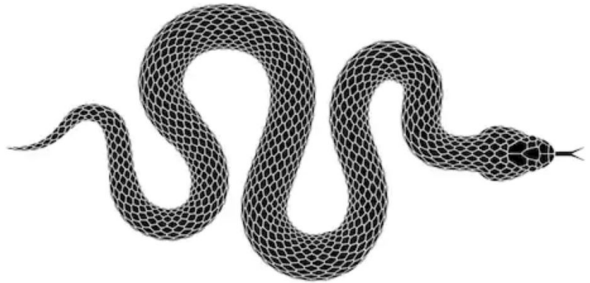
- Extensive list of existing vulnerability detectors:
  - Re-entrancy (DAO hack)
  - Missing constructor (Parity MultiSig Hack #1)
  - Uninitialized variables (Parity MultiSig Hack #2)
  - Variable shadowing (most honey pots)
  - Unimplemented functions (missed by solc)
  - Unsafe mapping deletion (missed by solc)
- Detection of poor coding practices

Inputs

- Solidity source code

Outputs

- Static analysis errors and warnings
- Inheritance graph and contract summary



Slither is open source!
https://github.com/trailofbits/slither

Other linters!

- Solcheck - https://github.com/federicobond/solcheck
- Solint - https://github.com/weifund/solint
- Solium - https://github.com/duaraghav8/Solium
- Solhint - https://github.com/protofire/solhint

Also, some symbolic execution

- I save Manticore for later, since it's so involved
- Mythril can work much like a linter though
  - https://github.com/ConsenSys/mythril
- Oyente as well
  - https://github.com/melonproject/oyente

2: Check important properties *thoroughly*

- I mostly use two tools
  - Manticore for symbolic execution/verification
  - Echidna for fuzzing
- Very complementary
  - If there's minimal control flow, Manticore can do formal verification
  - Manticore's great at exhausting small search spaces
  - Echidna, on the other hand is fast and resilient
  - Manticore takes days, Echidna takes minutes
  - Also, far harder to choke Echidna
- These are the rest of our workshop!

Manticore

Goals

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

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
        }
    }
}
```

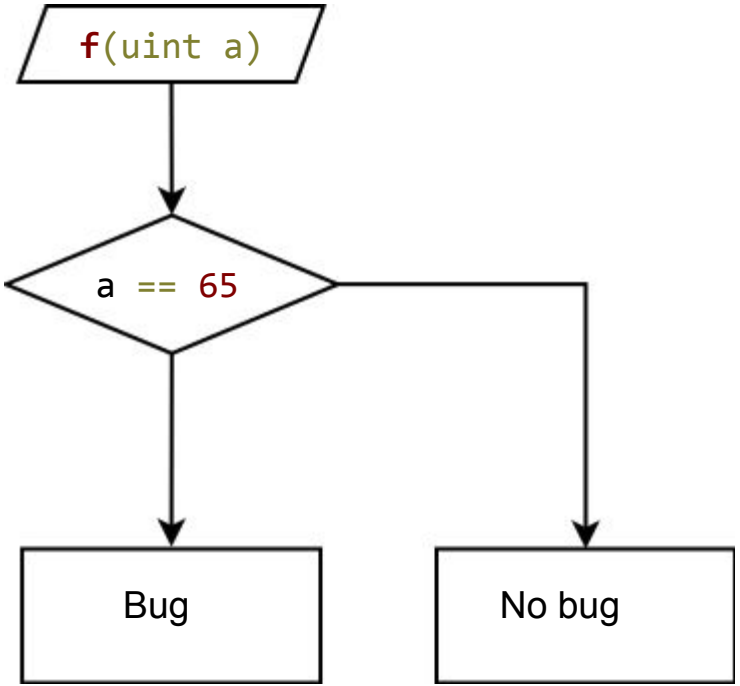
How to Review Code

- 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

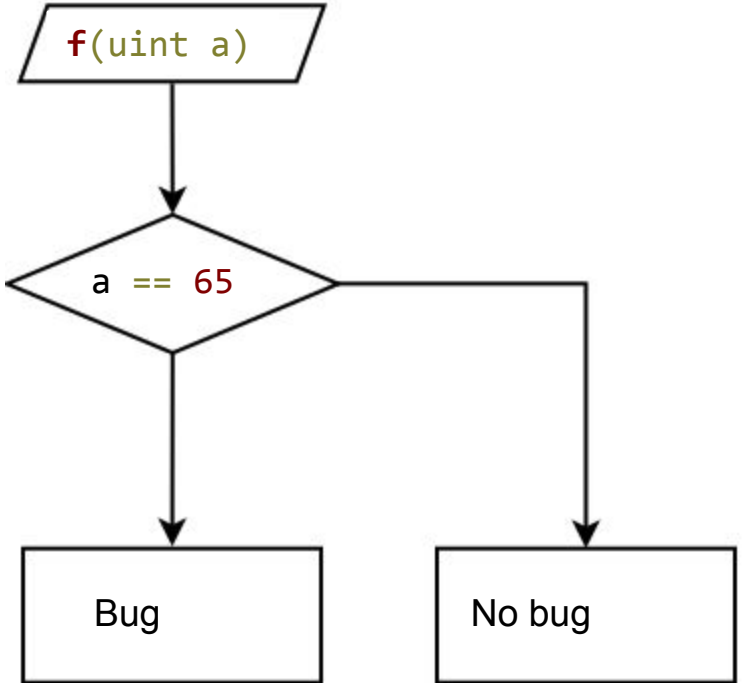
- Program exploration technique
- Execute the program “symbolically” = Represent executions as logical formulas
- Use an SMT solver to check the feasibility of a path and generate inputs

Symbolic Execution Example

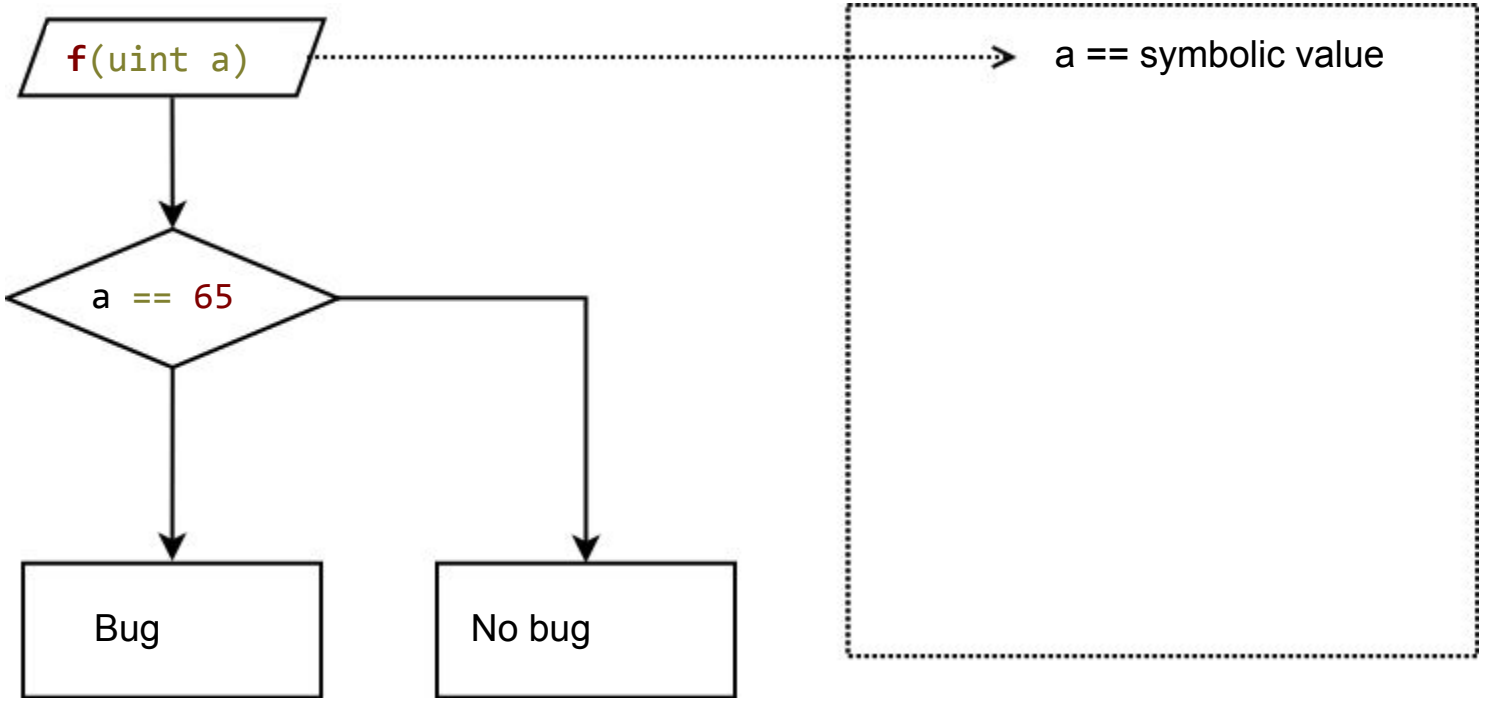



```
contract Simple {
    function f(uint a){

        // .. lot of paths and conditions

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

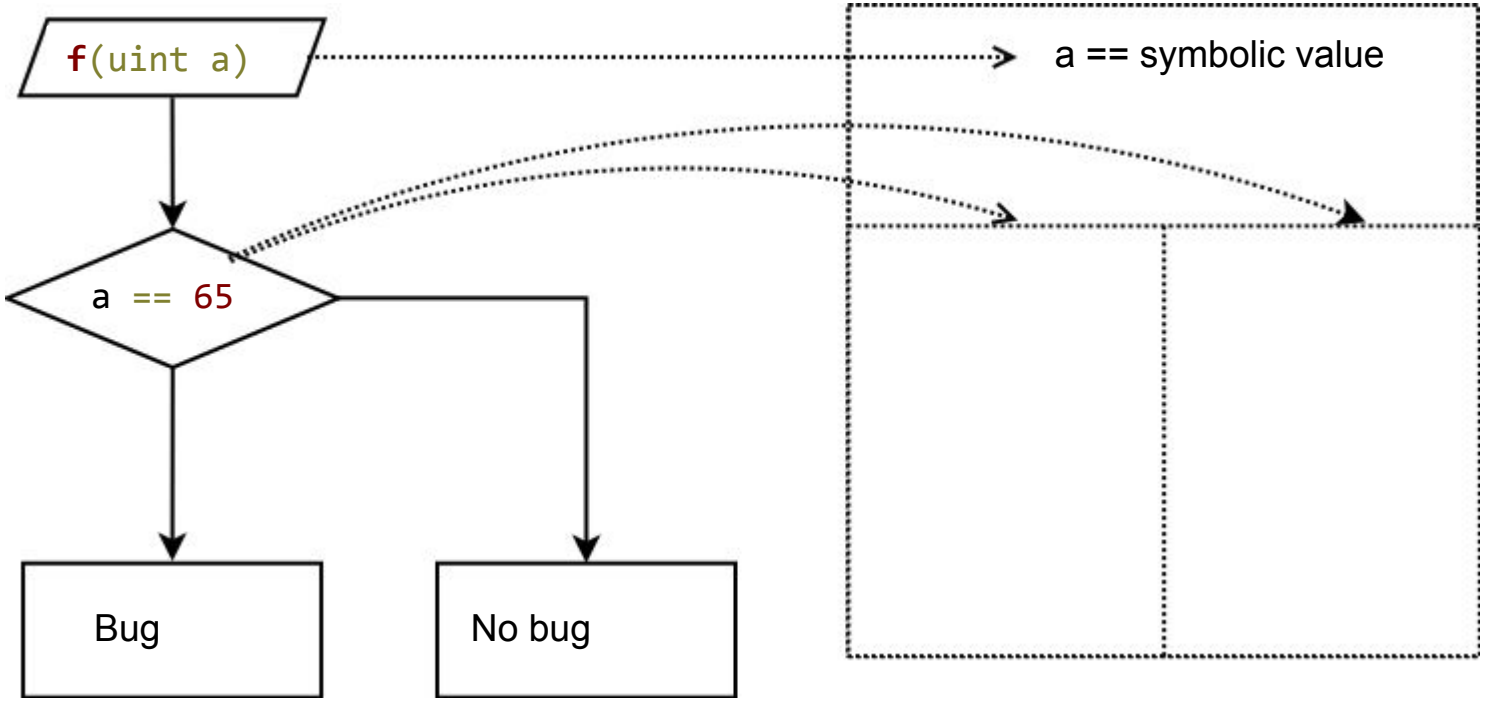
Symbolic Execution Example



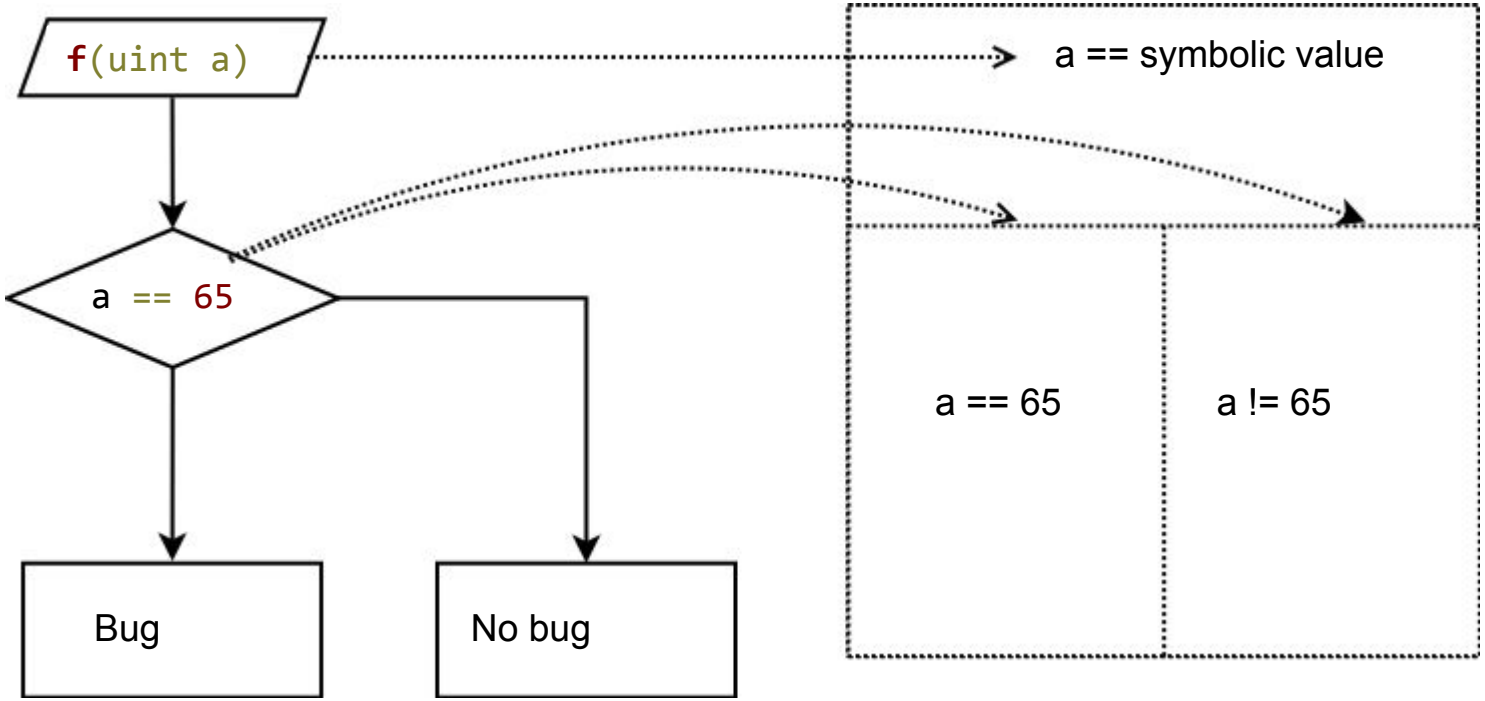
Symbolic Execution Example



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: 6060604052600436103603..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

Exercise 1: Solution

```
$python unprotectedWallet.py
Creator account: 0xd30a286ec6737b8b2a6a7b5fbb5d75b895f62956
(120482358228209984062407334714212197379227767023O)
Attacker account: 0x1bfa530d5d685155e98cd7d9dd23f7b6a801cfef
(159725171979439281175487293058222017669144629231)
Attacker can steal the ether! see /path/mcore_XXXX

$cat /path/mcore_XXXX/WalletHack_XXXX.tx

[..]

From: 0x1bfa530d5d685155e98cd7d9dd23f7b6a801cfef
changeOwner(159725171979439281175487293058222017669144629231L)

[..]

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(Operator.UGT(input0, last_return))

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

Add constraint input0 > sellerBalance

Exercise 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

Break

Echidna

What is echidna?

A smart fuzzer

```
➜ echidna git:(master) ✗ cat solidity/cli.sol
pragma solidity ^0.4.16;

contract Test {
  bool private flag0=true;
  bool private flag1=true;

  function set0(int val) returns (bool){
    if (val % 10 == 0) {flag0 = false;}
  }
  function set1(int val) returns (bool){
    if (val % 10 == 0 && flag0) {flag1 = false;}
  }
  function echidna_alwaystrue() returns (bool){
    return(true);
  }
  function echidna_sometimesfalse() returns (bool){
    return(flag0 || flag1);
  }
}
➜ echidna git:(master) ✗ ./echidna-test solidity/cli.sol
── solidity/cli.sol ──
  ✗ "echidna_sometimesfalse" failed after 36 tests and 681 shrinks.

      │ Call sequence: set0(7946810797001355118938603703351564369838113269809310950469780);
                       set1(804532980351965251305296916136264769537940399481075471846401995066776O);

  ✓ "echidna_alwaystrue" passed 100 tests.
  ✗ 1 failed, 1 succeeded.
➜ echidna git:(master) ✗
```

- **Finds “counterexamples”**
  - Via trying tons of call sequences
  - Then “shrinking” to something minimal
- **Only needs [solidity] tests to work**
  - Anything named `echidna_*`, with no arguments + returns a bool
- **Fast and easy**
- **Figures out ABIs itself**

Fuzzers: unreasonably effective

Consider: afl (American Fuzzy Lop)

IJG jpeg [1]	libjpeg-turbo [1] [2]	libpng [1]
libtiff [1] [2] [3] [4] [5]	mozjpeg [1]	PHP [1] [2] [3] [4] [5] [6] [7] [8]
Mozilla Firefox [1] [2] [3] [4]	Internet Explorer [1] [2] [3] [4]	Apple Safari [1]
Adobe Flash / PCRE [1] [2] [3] [4] [5] [6] [7]	sqlite [1] [2] [3] [4]...	OpenSSL [1] [2] [3] [4] [5] [6] [7]
LibreOffice [1] [2] [3] [4]	poppler [1] [2]...	freetype [1] [2]
GnuTLS [1]	GnuPG [1] [2] [3] [4]	OpenSSH [1] [2] [3] [4] [5]
PuTTY [1] [2]	ntpd [1] [2]	nginx [1] [2] [3]
bash (post-Shellshock) [1] [2]	tcpdump [1] [2] [3] [4] [5] [6] [7] [8] [9]	JavaScriptCore [1] [2] [3] [4]
pdfium [1] [2]	ffmpeg [1] [2] [3] [4] [5]	libmatroska [1]
libarchive [1] [2] [3] [4] [5] [6]...	wireshark [1] [2] [3]	ImageMagick [1] [2] [3] [4] [5] [6] [7] [8] [9]...
BIND [1] [2] [3] --	QEMU [1] [2]	lcms [1]
Oracle BerkeleyDB [1] [2]	Android / libstagefright [1] [2]	iOS / ImageIO [1]
FLAC audio library [1] [2]	libsndfile [1] [2] [3] [4]	less / lesspipe [1] [2] [3]
strings (+ related tools) [1] [2] [3] [4] [5] [6] [7]	file [1] [2] [3] [4]	dpkg [1] [2]
rcs [1]	systemd-resolved [1] [2]	libyaml [1]

Fuzzers: unreasonably effective

Consider: afl (American Fuzzy Lop)

IJG jpeg [1]	libjpeg-turbo [1] [2]	libpng [1]
libtiff [1] [2] [3] [4] [5]	mozjpeg [1]	PHP [1] [2] [3] [4] [5] [6] [7] [8]
Mozilla Firefox [1] [2] [3] [4]	Internet Explorer [1] [2] [3] [4]	Apple Safari [1]
Adobe Flash / PCRE [1] [2] [3] [4] [5] [6] [7]	sqlite [1] [2] [3] [4] ...	OpenSSL [1] [2] [3] [4] [5] [6] [7]
LibreOffice [1] [2] [3] [4]	poppler [1] [2] ...	freetype [1] [2]
GnuTLS [1]	GnuPG [1] [2] [3] [4]	OpenSSH [1] [2] [3] [4] [5]
PuTTY [1] [2]	ntpd [1] [2]	nginx [1] [2] [3]
bash (post-Shellshock) [1] [2]	tcpdump [1] [2] [3] [4] [5] [6] [7] [8] [9]	[illegible] [1] [2] [3] [4]
pdfium [1] [2]	ffmp	
libarchive [1] [2] [3] [4] [5] [6] ...	wire	
BIND [1] [2] [3] ...	Q	
Oracle BerkeleyDB [1] [2]	Android /	
FLAC audio library [1] [2]	libsn	
strings (+ related tools) [1] [2] [3] [4] [5] [6] [7]	fi	
rcs [1]	system	

Info-Zip unzip [1] [2]	libtasn1 [1] [2] ...	OpenBSD pfctl [1]
NetBSD bpf [1]	man & mandoc [1] [2] [3] [4] [5] ...	IDA Pro [reported by authors]
clamav [1] [2] [3] [4] [5] [6]	libxml2 [1] [2] [4] [5] [6] [7] [8] [9] ...	glibc [1]
clang / llvm [1] [2] [3] [4] [5] [6] [7] [8] ...	nasm [1] [2]	ctags [1]
mutt [1]	procmail [1]	fontconfig [1]
pdksh [1] [2]	Qt [1] [2] ...	wavpack [1] [2] [3] [4]
redis / lua-cmsgpack [1]	taglib [1] [2] [3]	privoxy [1] [2] [3]
perl [1] [2] [3] [4] [5] [6] [7] ...	libxmp	radare2 [1] [2]
SleuthKit [1]	fwknop [reported by author]	X.Org [1] [2]
exifprobe [1]	jhead [?]	capnproto [1]
Xerces-C [1] [2] [3]	metacam [1]	djvulibre [1]
exiv [1] [2]	Linux btrfs [1] [2] [3] [4] [5] [6] [7] [8]	Knot DNS [1]
curl [1] [2] [3]	wpa_supplicant [1]	libde265 [reported by author]
dnsmasq [1]	libbpg (1)	lame [1] [2] [3] [4] [5] [6]
libwmf [1]	uudecode [1]	MuPDF [1] [2] [3] [4]
imlib2 [1] [2] [3] [4]	libraw [1]	libbson [1]

Fuzzers: unreasonably effective

Consider: afl (American Fuzzy Lop)

Fuzzers: unreasonably effective

Consider: afl (American Fuzzy Lop)

Fuzzers: unreasonably effective

Consider: afl (American Fuzzy Lop)

IJG jpeg [1]	libjpeg-turbo [1][2]	libpng [1]
libtiff [1][2][3][4][5]	mozjpeg [1]	PHP [1][2][3][4][5][6][7][8]
Mozilla Firefox [1][2][3][4]	Internet Explorer [1][2][3][4]	Apple Safari [1]
Adobe Flash / PCRE [1][2][3][4][5][6][7]	sqlite [1][2][3][4]...	OpenSSL [1][2][3][4][5][6][7]
LibreOffice [1][2][3][4]	poppler [1][2]...	freetype [1][2]
GnuTLS [1]	GnuPG [1][2][3][4]	OpenSSH [1][2][3][4][5]
PuTTY [1][2]	ntpd [1][2]	nginx [1][2][3]
bash (post-Shellshock) [1][2]	tcpdump [1][2][3][4][5][6][7][8][9]	[illegible] [1][2][3][4]
pdfium [1][2]	ffmp[illegible]	
libarchive [1][2][3][4][5][6]...	wire[illegible]	
BIND [1][2][3]...	Q[illegible]	
Oracle BerkeleyDB [1][2]	Android /[illegible]	
FLAC audio library [1][2]	libsn[illegible]	
strings (+ related tools) [1][2][3][4][5][6][7]	fi[illegible]	
rcs [1]	system[illegible]	

Info-Zip unzip [1][2]	libtasn1 [1][2]...	
NetBSD bpf [1]	man & mandoc [1][2][illegible]	
clamav [1][2][3][4][5][6]	libxml2 [1][2][4][5][6][7][illegible]	
clang / llvm [1][2][3][4][5][6][7][8]...	nasm [1][2]	
mutt [1]	procmail [1]	
pdksh [1][2]	Qt [1][2]...	
redis / lua-cmsgpack [1]	taglib [1][2][3]	
perl [1][2][3][4][5][6][7]...	libxmp	
SleuthKit [1]	fwknop [reported by [illegible]	
exifprobe [1]	jhead [?]	
Xerces-C [1][2][3]	metacam [1]	djvulibre [1]
exiv [1][2]	Linux btrfs [1][2][3][4][6][7][8]	Knot DNS [1]
curl [1][2][3]	wpa_supplicant [1]	libde265 [reported by author]
dnsmasq [1]	libbpg (1)	lame [1][2][3][4][5][6]
libwmf [1]	uudecode [1]	MuPDF [1][2][3][4]
imlib2 [1][2][3][4]	libraw [1]	libbson [1]

libsass [1]	yara [1][2][3][4]	W3C tidy-html5 [1]
VLC [1][2]	FreeBSD syscons [1][2][3]	John the Ripper [1][2]
screen [1][2][3]	tmux [1][2]	mosh [1]

UPX [1]	indent [1]	libpcre [1][2][3]	MySQL [1]	gnulib [1]
MMIX [1]	OpenMPT [1][2]	openexr [1]	libmad [1][2]	ettercap [1]
dhcpcd [1]	Mozilla NSS [1]	lrzip [1][2][3]	freetds [1]...	Asterisk [1]
mbed TLS [1]	Linux netlink [1]	ytnef [1][2][3][4]...	raptor [1]	mpg123 [1]
Linux xfs [1]	botan [1]	Apache httpd [1]	exempi [1][2]	libgmime [1][2][3]
Adobe Reader [1]	libav [1]	pev [1][2][3][4]	Linux mem mgmt [1]	sleuthkit [1]
OpenBSD kernel [1]	collectd [1]	Mongoose OS [1]	iOS kernel [1]	

MatrixSSL [1]	jasper [1][2][3][4][5][6][7]...	MaraDNS [1]
w3m [1][2][3][4]	Xen [1]	OpenH232 [1]...
irssi [1][2][3]	cmark [1]	OpenCV [1]
Malheur [1]	gstreamer [1]...	Tor [1]
gdk-pixbuf [1]	audiofile [1][2][3][4][5][6]...	zstd [1]
lz4 [1]	stb [1]	cJSON [1]

On top of this, the fuzzer helped make countless non-security improvements to core tools (*u8, sed, awk, make, m4, yacc, PHP, ImageMagick, freedesktop.org, patch, libtasn1, libvorbis, zsh, lua, ninja, ruby, busybox, gcrypt, vim, Tor, poppler, libopus, BSD sh, gcc, qemu, w3m, zsh, dropbear, libtorrent, git, rust, gravity, ezfsprogs*, etc); found security issues in all sorts of less-widespread software (e.g., *parrot, lodepng, json-glib, cabextract, libmspack, qprint, gpsbabel, dmg2img, antiword, arj, unrar, unace, zoo, rzip, lrzip, libiso*, libtta, duktape, splint, zpaq, assimp, cppcheck, fasm, catdoc, pngcrush, cmark, p7zip, libjbig2, aaphoto, ttutils, apngopt, sqlparser, mdp, libtinyxml, freexl, bgpparser, testdisk, photorec, btcd, gumbo, chaiscript, teseq, colcrt, pttbbs, capstone, dex2oat, pillow, elftoolchain, aribas, universal-ctags, uriparser, jq, lha, xdelta, gnuplot, libwpd, teseq, cimg, libiberty, policycoreutils, libsemanage, renoise, metapixel, openclone, mp3splt, podofo, glslang, UEFITool, libchor, lldpd, pngquant, muparserx, mochilo, pyhocon, sysdig, Overpass-API, fish-shell, gumbo-parser, mapbox-gl-native, rapidjson, libjson, FLIF, MultiMarkdown, astyle, pax-utils, zziplib, PyPDF, spiffing, apk, pgpdump, icoutils, msitools, dosfstools, schoco, MojoShader*, and so on); and is likely responsible for quite a few other things that weren't publicly attributed to the tool.

Fuzzers: unreasonably effective

Consider: afl (American Fuzzy Lop)

IJG jpeg [1]	libjpeg-turbo [1] [2]	libpng [1]
libtiff [1] [2] [3] [4] [5]	mozjpeg [1]	PHP [1] [2] [3] [4] [5] [6] [7] [8]
Mozilla Firefox [1] [2] [3] [4]	Internet Explorer [1] [2] [3] [4]	Apple Safari [1]
Adobe Flash / PCRE [1] [2] [3] [4] [5] [6] [7]	sqlite [1] [2] [3] [4] ...	OpenSSL [1] [2] [3] [4] [5] [6] [7]
LibreOffice [1] [2] [3] [4]	poppler [1] [2] ...	freetype [1] [2]
GnuTLS [1]	GnuPG [1] [2] [3] [4]	OpenSSH [1] [2] [3] [4] [5]
PuTTY [1] [2]	ntpd [1] [2]	nginx [1] [2] [3]
bash (post-Shellshock) [1] [2]	tcpdump [1] [2] [3] [4] [5] [6] [7] [8] [9]	
pdfium [1] [2]	ffmp	
libarchive [1] [2] [3] [4] [5] [6] ...	wire	
BIND [1] [2] [3] ...	Q	
Oracle BerkeleyDB [1] [2]	Android /	
FLAC audio library [1] [2]	libsn	
strings (+ related tools) [1] [2] [3] [4] [5] [6] [7]	fi	
rcs [1]	system	

Info-Zip unzip [1] [2]	libtasn1 [1] [2] ...	
NetBSD bpf [1]	man & mandoc [1] [2]	
clamav [1] [2] [3] [4] [5] [6]	libxml2 [1] [2] [4] [5] [6] [7]	
clang / llvm [1] [2] [3] [4] [5] [6] [7] [8] ...	nasm [1] [2]	
mutt [1]	procmail [1]	
pdksh [1] [2]	Qt [1] [2] ...	
redis / lua-cmsgpack [1]	taglib [1] [2] [3]	
perl [1] [2] [3] [4] [5] [6] [7] ...	libxmp	
SleuthKit [1]	fwknop [reported by	
exifprobe [1]	jhead [?]	
Xerces-C [1] [2] [3]	metacam [1]	djvulibre [1]
exiv [1] [2]	Linux btrfs [1] [2] [3] [4] [6] [7] [8]	Knot DNS [1]
curl [1] [2] [3]	wpa_supplicant [1]	libde265 [reported by author]
dnsmasq [1]	libbpg (1)	lame [1] [2] [3] [4] [5] [6]
libwmf [1]	uudecode [1]	MuPDF [1] [2] [3] [4]
imlib2 [1] [2] [3] [4]	libraw [1]	libbson [1]

libsass [1]	yara [1] [2] [3] [4]	W3C tidy-html5 [1]
VLC [1] [2]	FreeBSD syscons [1] [2] [3]	John the Ripper [1] [2]
screen [1] [2] [3]	tmux [1] [2]	mosh [1]

UPX [1]	indent [1]	libpcre [1] [2] [3]	MySQL [1]	gnulib [1]
MMIX [1]	OpenMPT [1] [2]	openexr [1]	libmad [1] [2]	ettercap [1]
dhcpcd [1]	Mozilla NSS [1]	lrzip [1] [2] [3]	freetds [1] ...	Asterisk [1]
mbed TLS [1]	Linux netlink [1]	ytnef [1] [2] [3] [4] ...	raptor [1]	mpg123 [1]
Linux xfs [1]	botan [1]	Apache httpd [1]	exempi [1] [2]	libgmime [1] [2] [3]
Adobe Reader [1]	libav [1]	pev [1] [2] [3] [4]	Linux mem mgmt [1]	sleuthkit [1]
OpenBSD kernel [1]	collectd [1]	Mongoose OS [1]	iOS kernel [1]	

MatrixSSL [1]	jasper [1] [2] [3] [4] [5] [6] [7] ...	MaraDNS [1]
w3m [1] [2] [3] [4]	Xen [1]	OpenH232 [1] ...
irssi [1] [2] [3]	cmark [1]	OpenCV [1]
Malheur [1]	gstreamer [1] ...	Tor [1]
gdk-pixbuf [1]	audiofile [1] [2] [3] [4] [5] [6] ...	zstd [1]
lz4 [1]	stb [1]	cJSON [1]

On top of this, the fuzzer helped make countless non-security improvements to core tools (o8, *sed*, *awk*, *make*, *m4*, *yacc*, *PHP*, *ImageMagick*, *freedesktop.org*, *patch*, *libtasn1*, *libvorbis*, *zsh*, *lua*, *ninja*, *ruby*, *busybox*, *gcrypt*, *vim*, *Tor*, *poppler*, *libopus*, *BSD sh*, *gcc*, *qemu*, *w3m*, *zsh*, *dropbear*, *libtorrent*, *git*, *rust*, *gravity*, *e2fsprogs*, etc); found security issues in all sorts of less-widespread software (e.g., *parrot*, *lodepng*, *json-glib*, *cabextract*, *libmspack*, *qprint*, *gpsbabel*, *dmg2img*, *antiword*, *arj*, *unrar*, *unace*, *zoo*, *rzip*, *lrzip*, *libiso**, *libtta*, *duktape*, *splint*, *zpaq*, *assimp*, *cppcheck*, *fasm*, *catdoc*, *pngcrush*, *cmark*, *p7zip*, *libjbig2*, *aaphoto*, *ttutils*, *apngopt*, *sqlparser*, *mdp*, *libtinyxml*, *freexl*, *bgpparser*, *testdisk*, *photorec*, *btcd*, *gumbo*, *chaiscript*, *teseq*, *colcrt*, *pttbbs*, *capstone*, *dex2oat*, *pillow*, *elftoolchain*, *aribas*, *universal-ctags*, *uriparser*, *jq*, *lha*, *xdelta*, *gnuplot*, *libwpd*, *teseq*, *cimg*, *libiberty*, *policycoreutils*, *libsemanage*, *renoise*, *metapixel*, *openclone*, *mp3splt*, *podofo*, *glslang*, *UEFITool*, *libchor*, *lldpd*, *pngquant*, *muparserx*, *mochilo*, *pyhocon*, *sysdig*, *Overpass-API*, *fish-shell*, *gumbo-parser*, *mapbox-gl-native*, *rapidjson*, *libjson*, *FLIF*, *MultiMarkdown*, *astyle*, *pax-utils*, *zziplib*, *PyPDF*, *spiffing*, *apk*, *pgpdump*, *icoutils*, *msitools*, *dosfstools*, *schoco*, *MojoShader*, and so on); and is likely responsible for quite a few other things that weren't publicly attributed to the tool.

(also a bunch of other stuff that didn't make the trophy case)

Fuzzers: unreasonably effective

Consider: afl (American Fuzzy Lop)

(also a bunch of other stuff that didn't make the trophy case)
(also there are a bunch of other fuzzers)

A model checker

```
λ> check $ prop_turnstile v
  ✗ <interactive> failed after 4 tests and 1 shrink.

      ┏━━ examples/state-machine/StateMachine.hs ━━
   49 ┃ s_push_locked :: (Monad n, MonadTest m, MonadState VM m) => Command n m ModelState
   50 ┃ s_push_locked = Command (\s -> if s == TLocked then Just $ pure Push else Nothing)
   51 ┃   (\Push -> cleanUp >> execCall ("push", []))
   52 ┃   [ Require $ \s Push -> s == TLocked
   53 ┃   , Update $ \_ Push _ -> TLocked
   54 ┃   , Ensure $ \before after Push b -> do before === TLocked
   55 ┃                                         assert (match b False)
      ┃                                         ^^^^^^^^^^^^^^^^^^^^^^
   56 ┃                                         after === TLocked
   57 ┃   ]

      ┏━━ examples/state-machine/StateMachine.hs ━━
   69 ┃ prop_turnstile :: VM -> Property
   70 ┃ prop_turnstile v = property $ do
   71 ┃   actions <- forAll $ Gen.sequential (Range.linear 1 100) initialState
   72 ┃     [s_coin, s_push_locked, s_push_unlocked]
      ┃     | Var 0 = Coin
      ┃     | Var 1 = Push
      ┃     | Var 3 = Push
   73 ┃   evalStateT (executeSequential initialState actions) v

    This failure can be reproduced by running:
    > recheck (Size 3) (Seed 133816964769084861 (-8105329698605641335)) <property>

False
λ>
```

- Write symbolic state-machine models
- Check real code against them
- Write custom fuzzing campaigns against deep program logic

A model checker (cont.)

- **One of the hardest problems in fuzzing is good predicates**
  - How do you know a program state is bad?
- **To understand how the program fails, you need to understand the program**
  - We can do this with modeling
- **Many smart contracts are fundamentally FSMs (or an FSM + a little extra stuff)**
  - Echidna lets us express this in code, then use it to find bugs

A bunch of other small, helpful things

- Written to be a library first and an executable second
- ABI conveniences
- Execution conveniences
- Handy for general EVM futzing

CLI usage

echidna-test

- **Echidna ships one general-purpose, easy to use executable**
- **No Haskell required!**
- **All the user needs to do is write assertions**
  - An assertion is any solidity function with no arguments, returning a `bool` with a name of the form "`echidna_*`"
- **Assertions should always return true; echidna just tries to find ways to make them false**
  - It does this by calling other functions at random (with random arguments)
  - By default, it puts together 100 call sequences of 1-10 calls each
  - Then, it calls the assertion and if it returns false, starts "shrinking"
  - Shrinking is done by removing random calls and ensuring the assertion still fails
  - Should result in relatively minimal call sequences (much of the time)
  - Once there are no more shrinks, or 1000 shrinks have been tried, print the call sequence
  - If no falsification found, say the test passed

`echidna-test` (example)

```
➜ echidna git:(master) ✗ cat solidity/cli.sol
pragma solidity ^0.4.16;

contract Test {
  bool private flag0=true;
  bool private flag1=true;

  function set0(int val) returns (bool){
    if (val % 10 == 0) {flag0 = false;}
  }
  function set1(int val) returns (bool){
    if (val % 10 == 0 && flag0) {flag1 = false;}
  }
  function echidna_alwaystrue() returns (bool){
    return(true);
  }
  function echidna_sometimesfalse() returns (bool){
    return(flag0 || flag1);
  }
}
➜ echidna git:(master) ✗ ./echidna-test solidity/cli.sol
── solidity/cli.sol ──
  ✗ "echidna_sometimesfalse" failed after 36 tests and 681 shrinks.

      | Call sequence: set0(7946810797001355118938603703351564369838113269809310950469780);
      |                set1(8045329803519652513052969161362647695379403994810754718464019950667760);

  ✓ "echidna_alwaystrue" passed 100 tests.
  ✗ 1 failed, 1 succeeded.
➜ echidna git:(master) ✗
```

How do I write good asserts?

- John Regehr has the answers
- **When possible, use math**
  - If you have a triangle, the angles should sum to 2π
  - If you have a random distribution, check its chi-squared
- **When possible, use invariants of your data structure**
  - If you have a red/black tree, it should be balanced
  - If you have a linked list, it shouldn't loop
- **If you have a spec, this is a lot easier**

Other tips

- **Don’t test your main contract, inherit from it**
  - Also helps with the constructor problem
  - You can even simulate multi-contract systems
  - Override functions that are problematic with simpler alternatives
- **Don’t be afraid to use the API**
  - Custom generators are very easy to write
  - Multi-contract stuff is even doable
  - Simulate custom setup/chain state
- **Ask for help early!**
  - Very young software
  - Github gets good responses

What's in the pipeline?

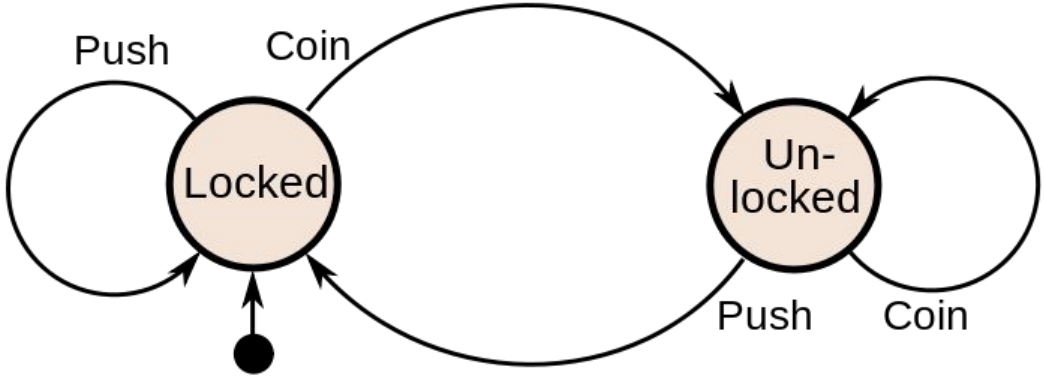
- Tweakable parameters (callseq length, number of callseqs)
- Multi-contract testing
- More parallelism
- Taking requests!

Custom Analyses

Custom analyses (the fun stuff)

- **Echidna supports writing much more involved tests with the API than the CLI**
- **Particularly good at “state machine” testing**
  - Enumerate states, transitions, and invariants (in haskell)
  - Test implementation vs. model (in solidity)
- **Result: fuzz against sophisticated program logic**

Let's write an analysis



Turnstile state machine

Turnstile.sol

```
➜ echidna git:(master) ✗ cat solidity/turnstile/turnstile.sol
pragma solidity ^0.4.16;

contract Turnstile {
  bool private locked = true; // initial state is locked

  function coin() {
    locked = false;
  }

  function push() returns (bool) {
    if (locked) {
      return(false);
    } else {
      locked = true;
      return(true);
    }
  }
}
➜ echidna git:(master) ✗
```

statemachine.hs

```
➜ echidna git:(master) ✗ cat examples/state-machine/StateMachine.hs | head -n 35 | tail -n 16
data ModelState (v :: * -> *) = TLocked
                              | TUnlocked
                              deriving (Eq, Ord, Show)

initialState :: ModelState v
initialState = TLocked

data Coin (v :: * -> *) = Coin deriving (Eq, Show)

data Push (v :: * -> *) = Push deriving (Eq, Show)

instance HTraversable Coin where
  htraverse _ Coin = pure Coin

instance HTraversable Push where
  htraverse _ Push = pure Push
➜ echidna git:(master) ✗
```

statemachine.hs

```
➜ echidna git:(master) ✗ cat examples/state-machine/StateMachine.hs | head -n 43 | tail -n 6
s_coin :: (Monad n, MonadTest m, MonadState VM m) => Command n m ModelState
s_coin = Command (\_ -> Just $ pure Coin)
  (\Coin -> execCall ("coin", []))
  [ Update $ \_ Coin _ -> TUnlocked
  , Ensure $ \_ s Coin _ -> s === TUnlocked
  ]
➜ echidna git:(master) ✗
```

statemachine.hs

```
➜ echidna git:(master) ✗ cat examples/state-machine/StateMachine.hs | head -n 57 | tail -n 12
match (VMSuccess (B s)) b = s == encodeAbiValue (AbiBool b)
match _ _ = False

s_push_locked :: (Monad n, MonadTest m, MonadState VM m) => Command n m ModelState
s_push_locked = Command (\s -> if s == TLocked then Just $ pure Push else Nothing)
  (\Push -> execCall ("push", []))
  [ Require $ \s Push -> s == TLocked
  , Update $ \_ Push _ -> TLocked
  , Ensure $ \before after Push b -> do before === TLocked
                                        assert (match b False)
                                        after === TLocked
  ]
➜ echidna git:(master) ✗
```

statemachine.hs

```
➜ echidna git:(master) ✗ cat examples/state-machine/StateMachine.hs | head -n 67 | tail -n 9
s_push_unlocked :: (Monad n, MonadTest m, MonadState VM m) => Command n m ModelState
s_push_unlocked = Command (\s -> if s == TUnlocked then Just $ pure Push else Nothing)
  (\Push -> execCall ("push", []))
  [ Require $ \s Push -> s == TUnlocked
  , Update $ \_ Push _ -> TLocked
  , Ensure $ \before after Push b -> do before === TUnlocked
                                        assert (match b True)
                                        after === TLocked
  ]
```

statemachine.hs

```
➜ echidna git:(master) ✗ cat examples/state-machine/StateMachine.hs | head -n 73 | tail -n 5
prop_turnstile :: VM -> Property
prop_turnstile v = property $ do
  actions <- forAll $ Gen.sequential (Range.linear 1 100) initialState
    [s_coin, s_push_locked, s_push_unlocked]
  evalStateT (executeSequential initialState actions) v
➜ echidna git:(master) ✗
```

Let's check it!

```
λ> (v,_,_) <- loadSolidity "solidity/turnstile/turnstile.sol"
λ> check $ prop_turnstile v
  ✓ <interactive> passed 100 tests.
True
λ>
```

What's in the pipeline?

- Lots of UI stuff, focused on ease of use
- Support fancier state machines
- Lots more docs
- TBD (taking requests!)

Hacking on Echidna

What to know

Necessary
- dapphub/hevm
- hedgehogqa/hedgehog

Helpful
- lens
- mtl

We're super open to contributors! Message @japesinator on Empire Hacking (empireslacking.herokuapp.com), email jp@trailofbits.com, or just open an issue!

Exercises

exercises/testme.sol

Implements a canal lock: has two gates, which can be up or down, but both can't be down at once. `raise` raises a gate, taking a bool to determine which, and `lower` lowers a gate, taking a bool to determine which and returning a bool indicating success.

1. Add some CLI tests
2. Modify the contract so they fail (can you have both gates down at once)?
3. Write a state machine test a la turnstile