Introduction to Fuzzing

& How to shift to Invariant Driven Development

Before starting

- git clone https://github.com/montyly/fuzzing-workshop
  - Exercises & slides

WhoAmI

- Josselin Feist (@Montyly)
- Independent security researcher
  - Trail of Bits: 2017 - 2025
  - Reviewed DeFi/L1/L2/…
  - Created Slither
- *seceureka.com*

Agenda

- Fuzzing
- Exercises
- Fuzzing in real life
- Invariant Driven Development
- Conclusion

Fuzzing

Do you have bugs?

```
function buy(uint tokens) public payable{
        require(msg.value >= _cost(tokens) * 10**18);
        _mint(msg.sender, tokens * 10**18);
}

/// @notice Compute the cost.  1 ether = 10 tokens
function _cost(uint desired_tokens) internal view returns(uint){
        return(desired_tokens / 10);
}
```

How to find bugs?

- 4 main techniques
  - Unit tests
  - Manual review
  - Fully automated analysis
  - Semi automated analysis

How to find bugs?

- 4 main techniques
  - Unit tests
  - Manual review
  - Fully automated analysis
  - **Semi automated analysis**
    - “Human in the loop”
    - Ex: Fuzzing, formal verification

Fuzzing

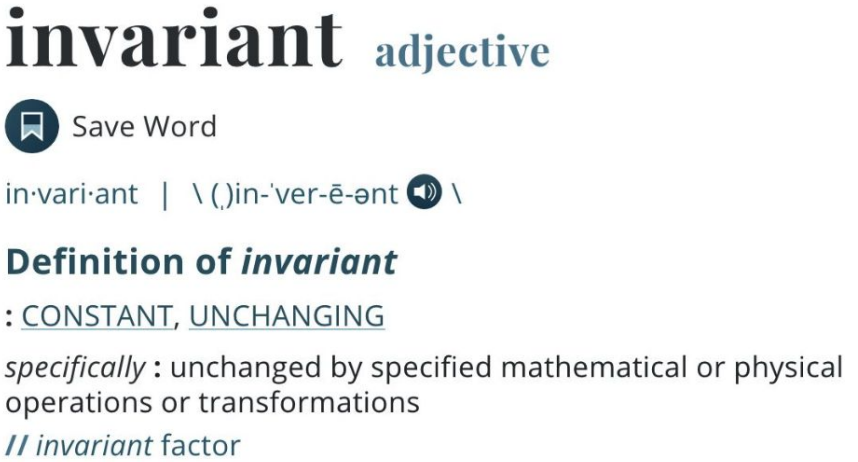
- Stress the program with random inputs
  - Most basic fuzzer “randomly type on your keyboard”
- Fuzzing is well established in web2 security
  - AFL, libfuzzer, go-fuzz etc

Fuzzing - Property based testing

- Web2 : crash
- Web3 : logic bugs
- Property based testing
  - User defines *invariants*
  - Fuzzer generates random inputs
  - Check if *invariants* hold
- “Unit tests on steroids”

Invariant

- Something that must be **always** true



invariant *adjective*

Save Word

in·vari·ant | \ (ˌ)in-ˈver-ē-ənt \

Definition of *invariant*

: CONSTANT, UNCHANGING

specifically : unchanged by specified mathematical or physical operations or transformations

// *invariant* factor

Invariant

User balance never exceeds total supply

Invariant

- How to express invariants in Solidity?
  - Dedicated function
    - function invariant_something()
  - Assertion
    - assert(something)
  - Events
    - Emit AssertionFailure(..)
  - Depends on the fuzzer / fuzzer mode

Fuzzers

- Foundry
  - Easiest to use
  - Lowest performance
- Echidna/Medusa
  - Expert level fuzzer
  - Similar performance
- Which one to use?
  - Foundry if you spend < 1 day
  - Echidna or Medusa otherwise

Exercises

Exercise 1

- git clone https://github.com/montyly/fuzzing-workshop
- Open Exercise-1.md

Goal: check if total supply invariant holds

Notes:

- Use Solidity 0.8 (see solc-select if needed)
- Try without the template!

Exercise 1

```
contract Token is Ownable, Pausable {
    mapping(address => uint256) public balances;

    function transfer(address to, uint256 value) public whenNotPaused {
        // unchecked to save gas
        unchecked {
            balances[msg.sender] -= value;
            balances[to] += value;
        }
    }
}
```

Exercise 1 - Template

```
contract TestToken is Token {
    address echidna_caller = msg.sender;

    constructor() public {
        balances[echidna_caller] = 10000;
    }
    // add the property
}
```

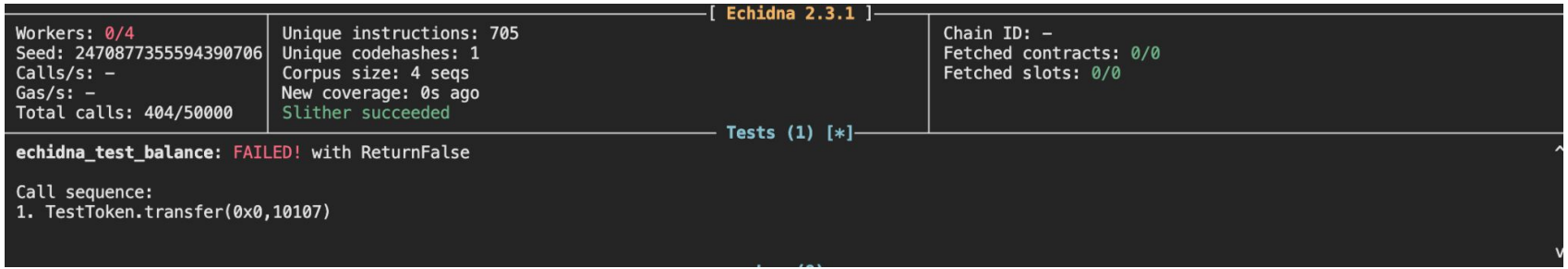
Exercise 1 - Solution

```
contract TestToken is Token {
    address echidna_caller = msg.sender;
    constructor() public {
        balances[echidna_caller] = 10000;
    }

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

Exercise 1 - Solution

$ echidna solution.sol



Exercise 1 - Solution

```
contract Token is Ownable, Pausable {
    mapping(address => uint256) public balances;

    function transfer(address to, uint256 value) public whenNotPaused {
        // unchecked to save gas
        unchecked {
            balances[msg.sender] -= value;
            balances[to] += value;
        }
    }
}
```

Exercise 2

- git clone https://github.com/montyly/fuzzing-workshop
- Open Exercise-2.md

Goal: can you unpause the system?

Notes:

- Use Solidity 0.8 (see solc-select if needed)
- Try without the template!

Exercise 2

```
contract Ownable {
        address public owner = msg.sender;
        function Owner() public {
                owner = msg.sender;
        }
        modifier onlyOwner() {
                require(owner == msg.sender);
                _;
        }
}
```

```
contract Pausable is Ownable {
        bool private _paused;
        function paused() public view returns
        (bool) {
                return _paused;
        }
        function pause() public onlyOwner {
                _paused = true;
        }
        function resume() public onlyOwner {
                _paused = false;
        }
```

Exercise 2 - Template

```
contract TestToken is Token {

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

    // add the property
}
```

Exercise 2 - Solution

```
contract TestToken is Token {
	constructor() {
		paused();
		owner = 0x0; // lose ownership
	}

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

Exercise 2 - Solution

```
contract Ownable {
        address public owner = msg.sender;
        function Owner() public {
                owner = msg.sender;
        }
        modifier onlyOwner() {
                require(owner == msg.sender);
                _;
        }
}
```

```
contract Pausable is Ownable {
        bool private _paused;
        function paused() public view returns
        (bool) {
                return _paused;
        }
        function pause() public onlyOwner {
                _paused = true;
        }
        function resume() public onlyOwner {
                _paused = false;
        }
```

Exercise 2 - Solution

$ echidna solution.sol

```
echidna_no_transfer: FAILED! with ReturnFalse

Call sequence:
1. TestToken.Owner()
2. TestToken.resume()
```

Is there a bug?

Is there a bug?

```
function buy(uint tokens) public payable{
        require(msg.value >= _cost(tokens)  * 10**18);
        _mint(msg.sender, tokens  * 10**18);
}

/// @notice Compute the cost.  1 ether = 10 tokens
function _cost(uint desired_tokens) internal view returns(uint){
        return(desired_tokens / 10);
}
```

Is there a bug?

```
function buy(uint tokens) public payable{
        require(msg.value >= _cost(tokens)  * 10**18);
        _mint(msg.sender, tokens  * 10**18);
}

/// @notice Compute the cost.  1 ether = 10 tokens
function _cost(uint desired_tokens) internal view returns(uint){
        return(desired_tokens / 10);
}
```

- buy is stateful
- _cost is stateless
    - Start with it

Is there a bug?

- What invariants?

```
/// @notice Compute the cost.  1 ether = 10 tokens
function _cost(uint desired_tokens) internal view returns(uint){
        return(desired_tokens / 10);
}
```

Is there a bug?

- What invariants?
  - If the function returns zero, desired_tokens must be zero

```
/// @notice Compute the cost.  1 ether = 10 tokens
function _cost(uint desired_tokens) internal view returns(uint){
        return(desired_tokens / 10);
}
```

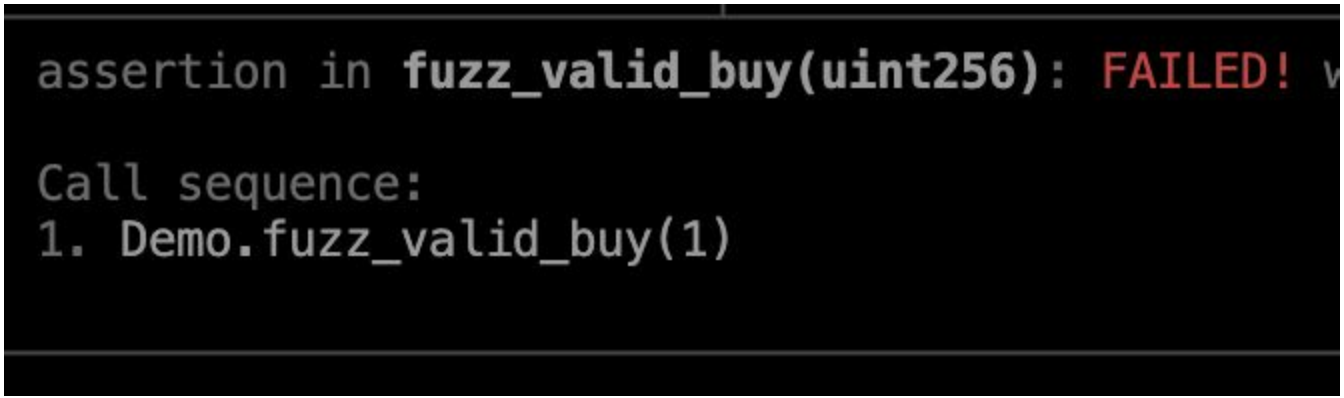
Is there a bug?

- What invariants?
  - If the function returns zero, desired_tokens must be zero

```
function fuzz_valid_buy(uint desired_tokens) public{
        uint cost = _cost(desired_tokens);

        if(cost == 0){
                assert(desired_tokens ==0);
        }
}
```

Is there a bug?



Is there a bug?

```
/// @notice Compute the cost.  1 ether = 10 tokens
function _cost(uint desired_tokens) internal view returns(uint){
        return(desired_tokens / 10);
}
```

Fuzzing in real life

How to define good invariants

- Start small, and iterate
- Steps
  1. Define invariants in English
  2. Write the invariants in Solidity
  3. Run the fuzzer
     - If invariant broken: investigate
     - Otherwise go back to (1)

Invariants categories

- Function level
  - Ex: Interest is monotonically increasing
  - Inherit the targets & use assert()
- System level
  - Ex: User balance <= total supply
  - Require initialization & use property

Main challenges

- Defining / refining invariants
- Setup / initialization
  - Multiple contracts
  - Parameters bounds
  - “Harness”
- Tools limitations

Invariant Driven Development

Invariant driven development

- Invariants go beyond fuzzing
  - Monitoring
  - Manual reviews
  - On-chain invariants
    - Ex: Uniswap’ K invariant

End goal: Invariant driven development

- Before writing the code
  - What are the main invariants?
  - How will these invariants be checked?
  - How will these invariants be specified and kept in sync with the code?
- Design pattern: Pre condition / Command / Post condition
  - *Arrange, Act, Assert* or *Given, When,Then*

Example

ID	Invariants	Components	Testing Strategy
00	The balance of any user must never exceed the total supply of the token	MyToken	Fuzzing
01	If the pool has no fee (*pre-condition*) Call the swap function (*command*) $x * y = k$ has not changed (*post-condition*)	MyAMM	Fuzzing
02	The function computing the interest earned over time is an increasing monotonic function	Lending.compute_interest	Formal verification

Conclusion

Conclusion

- Invariant driven development
  - Paradigm shift
- Fuzzing
  - Key technique to find bugs
  - First step to practice invariant driven development

Ressources

- secure-contracts.com
- Perimeter’s discord (ask for invite)