TRAIL OF BITS

Introduction to fuzzing

Who am I?

- **Josselin Feist (@montyly)**

- **Trail of Bits: trailofbits.com**
  - We help developers to build safer software
  - R&D focused: we use the latest program analysis techniques
  - Slither, medusa, Tealer, Caracal, solc-select, ..

ToB Twitter list

Agenda

- How to find bugs?
- What is property based testing?
- How to define good invariants?

How to Find Bugs?

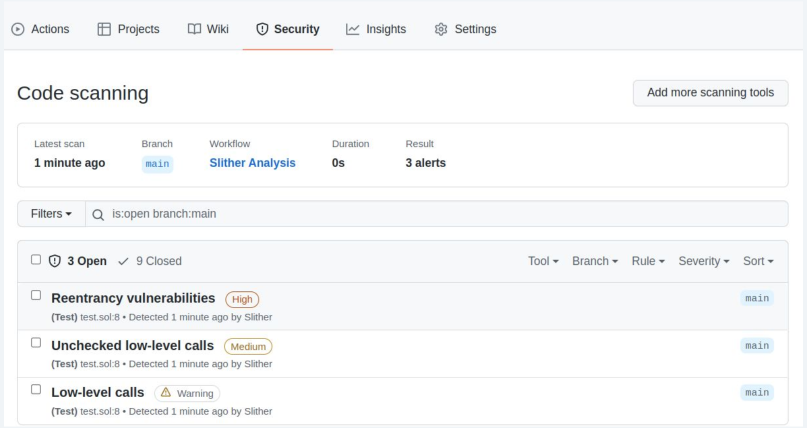
```
/// @notice Allow users to buy token. 1 ether = 10 tokens
/// @param tokens The numbers of token to buy
/// @dev Users can send more ether than token to be bought, to give gifts to the
team.
function buy(uint tokens) public payable{
    _valid_buy(tokens, msg.value);
    _mint(msg.sender, tokens);
}


/// @notice Compute the amount of token to be minted. 1 ether = 10 tokens
/// @param desired_tokens The number of tokens to buy
/// @param wei_sent The ether value to be converted into token
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);
}
```

How to Find Bugs?

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

Fully automated analysis

- **Benefits**
  - Quick & easy to use
- **Limitations**
  - Cover only some class of bugs
- **Example: Slither**



Code scanning

Latest scan	Branch	Workflow	Duration	Result
1 minute ago	main	Slither Analysis	0s	3 alerts

is:open branch:main

3 Open ✓ 9 Closed

- Reentrancy vulnerabilities High — main
  (Test) test.sol:8 • Detected 1 minute ago by Slither
- Unchecked low-level calls Medium — main
  (Test) test.sol:8 • Detected 1 minute ago by Slither
- Low-level calls Warning — main
  (Test) test.sol:8 • Detected 1 minute ago by Slither

https://github.com/crytic/slither-action

Semi automated analysis

- **Benefits**
  - Great for logic-related bugs
- **Limitations**
  - Require human in the loop
- **Example: Property based testing with Echidna or Medusa**

What is property based testing?

Fuzzing

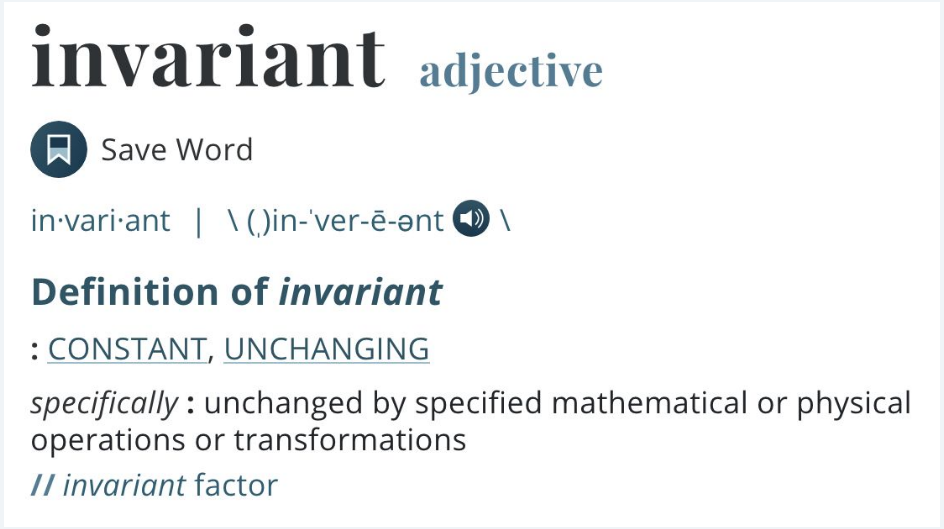
- **Stress the program with random inputs**
- **Fuzzing is well established in traditional software security**
  - AFL, Libfuzzer, go-fuzz, ..

Property based testing

- **Traditional fuzzers generally detect crashes**
  - Smart contracts don’t (really) have crashes
- **Property based testing**
  - User defines invariants
  - Fuzzer generates random inputs
  - Check whether specified “incorrect” state can be reached
- **“Unit tests on steroids”**

Invariant

- **Something that must always be 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

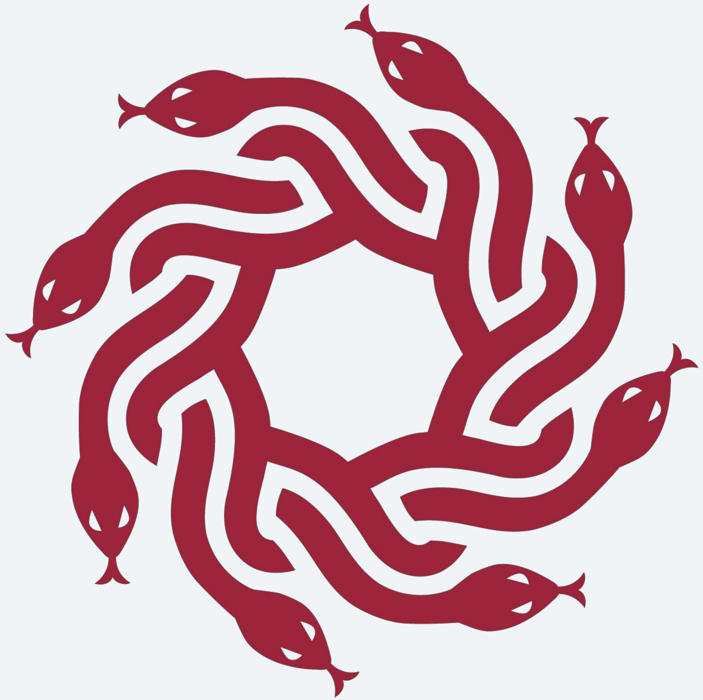
Invariant - Token’s total supply

User balance never exceeds total supply

Medusa

Medusa

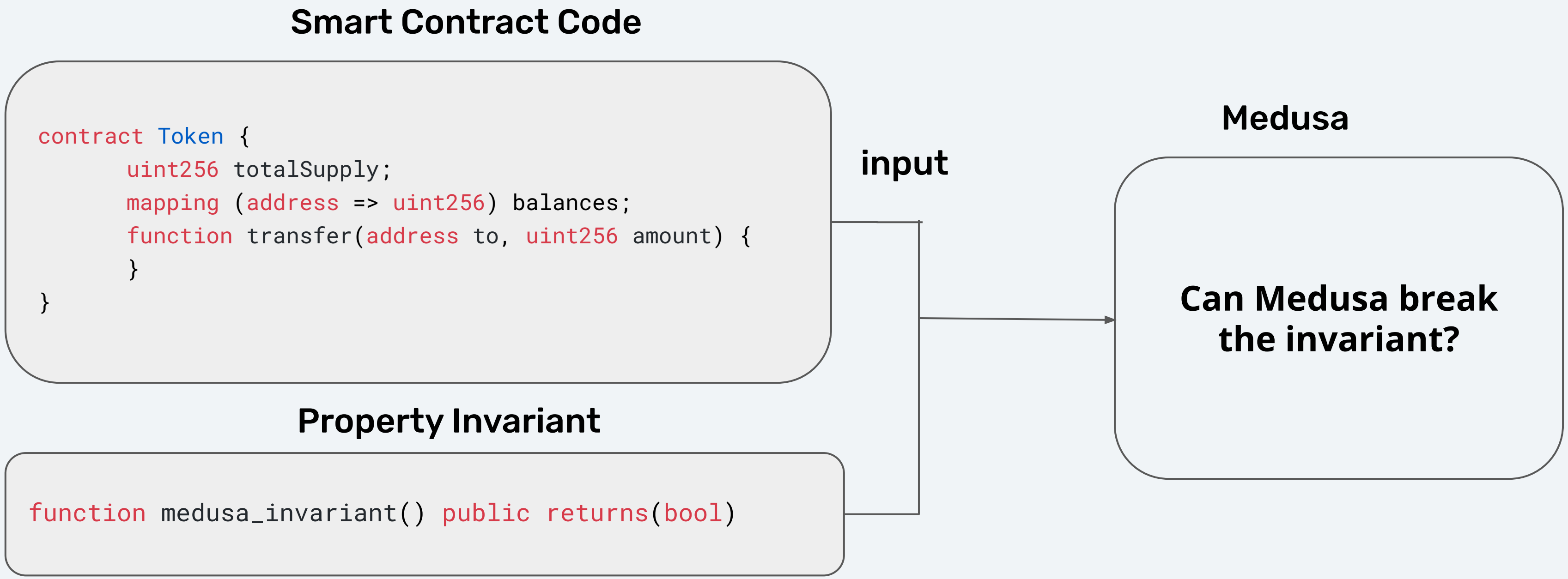
- https://github.com/crytic/medusa
  - New fuzzer built from Echidna's experience
  - Smart contract fuzzer
  - Written in Go



Medusa

- **medusa init**
  - Generate a medusa.json config file
- **medusa fuzz**
  - Fuzz the project
- **Want to learn more?**
  - https://secure-contracts.com/medusa

Medusa - Overview



```
contract Token {
        uint256 totalSupply;
        mapping (address => uint256) balances;
        function transfer(address to, uint256 amount) {
        }
}
```

```
function medusa_invariant() public returns(bool)
```

Example - Token

```
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;
        }
    }
}
```

Example - User balance never exceeds total supply

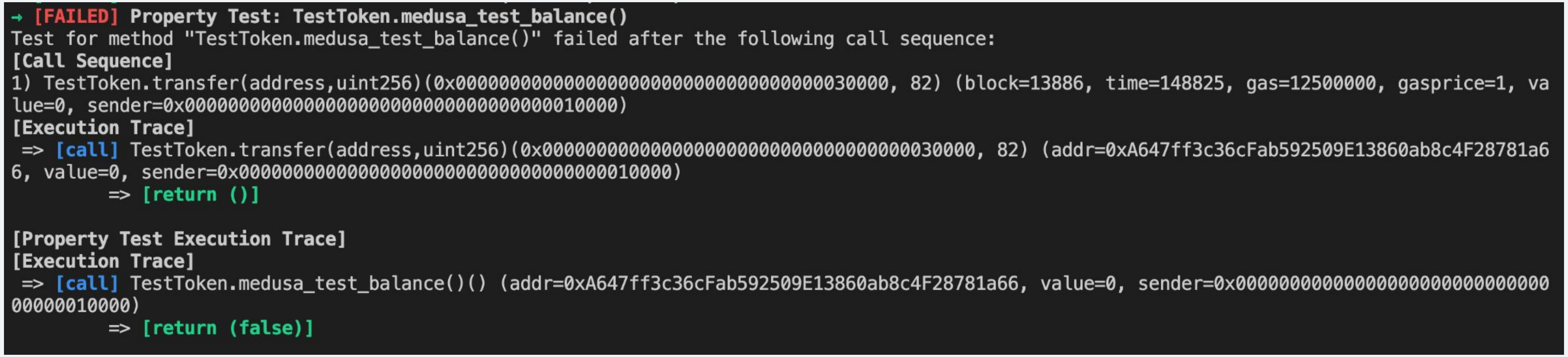
```
contract TestToken is Token {

   address medusa_caller = msg.sender;



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


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

Example - User balance never exceeds total supply

```
contract TestToken is Token {

   address medusa_caller = msg.sender;



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


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

Exercise 1 - Solution

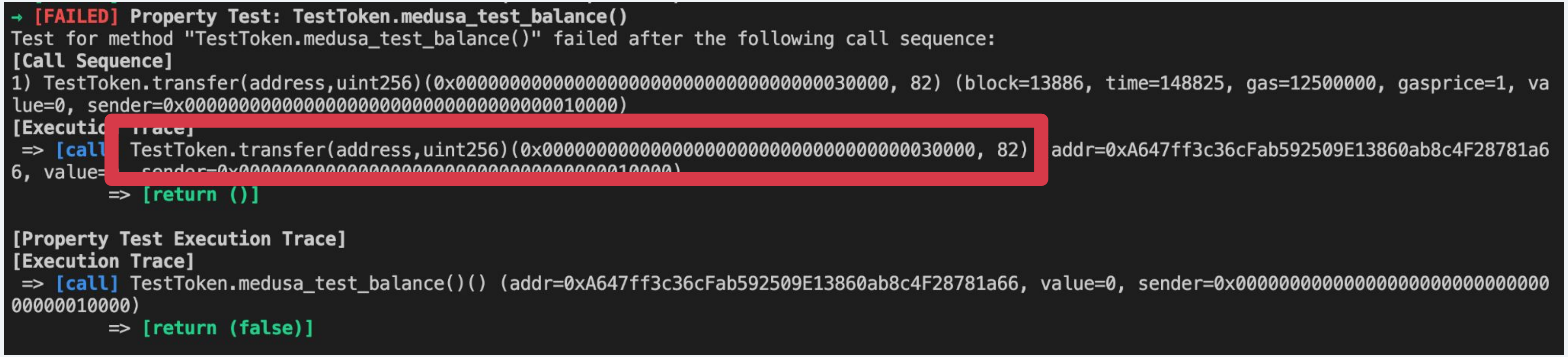
```
$ medusa fuzz
```



```
→ [FAILED] Property Test: TestToken.medusa_test_balance()
Test for method "TestToken.medusa_test_balance()" failed after the following call sequence:
[Call Sequence]
1) TestToken.transfer(address,uint256)(0x0000000000000000000000000000000000030000, 82) (block=13886, time=148825, gas=12500000, gasprice=1, va
lue=0, sender=0x0000000000000000000000000000000000010000)
[Execution Trace]
 => [call] TestToken.transfer(address,uint256)(0x0000000000000000000000000000000000030000, 82) (addr=0xA647ff3c36cFab592509E13860ab8c4F28781a6
6, value=0, sender=0x0000000000000000000000000000000000010000)
         => [return ()]

[Property Test Execution Trace]
[Execution Trace]
 => [call] TestToken.medusa_test_balance()() (addr=0xA647ff3c36cFab592509E13860ab8c4F28781a66, value=0, sender=0x000000000000000000000000000000
00000010000)
         => [return (false)]
```

Exercise 1 - Solution

```
$ medusa fuzz
```



```
→ [FAILED] Property Test: TestToken.medusa_test_balance()
Test for method "TestToken.medusa_test_balance()" failed after the following call sequence:
[Call Sequence]
1) TestToken.transfer(address,uint256)(0x0000000000000000000000000000000000030000, 82) (block=13886, time=148825, gas=12500000, gasprice=1, va
lue=0, sender=0x0000000000000000000000000000000000010000)
[Executic  Trace]
 => [call  TestToken.transfer(address,uint256)(0x0000000000000000000000000000000000030000, 82)  addr=0xA647ff3c36cFab592509E13860ab8c4F28781a6
6, value=
         => [return ()]

[Property Test Execution Trace]
[Execution Trace]
 => [call] TestToken.medusa_test_balance()() (addr=0xA647ff3c36cFab592509E13860ab8c4F28781a66, value=0, sender=0x000000000000000000000000000000
00000010000)
         => [return (false)]
```

Example - Token

```
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;
        }
    }
}
```

How to define good invariants

Defining good invariants

- **Start small, and iterate**
- **Steps**
    1. Define invariants in English
    2. Write the invariants in Solidity
    3. Run medusa
        - If invariants broken: investigate
        - Once all the invariants pass, go back to (1)

Identify invariants: Maths

- **Math library**
  - Commutative property
    - 1 + 2 = 2 + 1
  - Identity property
    - 1 * 2 = 2
  - Inverse property
    - x + (-x) = 0

Identify invariants: tokens

- **ERC20.total_supply**
  - No user should have a balance > total_supply
- **ERC20.transfer:**
  - After calling `transfer`
    - My balance should have decreased by the amount
    - The receiver’s balance should have increased by the amount

Identify invariants: tokens

- **ERC20.total_supply**
  - No user should have a balance > total_supply
- **ERC20.transfer:**
  - After calling `transfer`
    - My balance should have decreased by the amount
    - The receiver's balance should have increased by the amount
    - **If the destination is myself, my balance should be the same**

Identify invariants: tokens

- **ERC20.total_supply**
  - No user should have a balance > total_supply
- **ERC20.transfer:**
  - After calling `transfer`
    - My balance should have decreased by the amount
    - The receiver's balance should have increased by the amount
    - If the destination is myself, my balance should be the same
  - If I don't have enough funds, the transaction should revert/return false

Write invariants in Solidity

- **Identify the target of the invariant**
  - Function-level invariant
    - Ex: arithmetic associativity
    - Usually stateless invariants
    - Can craft scenario to test the invariant
  - System-level invariant
    - Ex: user's balance < total supply
    - Usually stateful invariants
    - All functions must be considered

Function-level invariant

- **Inherit the targets**
- **Create function and call the targeted function**
- **Use assert to check the property**

```
contract TestMath is Math{
    function test_commutative(uint a, uint b) public {
        assert(add(a, b) == add(b, a));
   }
}
```

System level invariant

- **Require specific initialization**
  - Constructors
  - Larger harness
  - Actors based fuzzing
- **medusa will explore all the other functions**

Advanced fuzzing tips

Advanced fuzzing tips

- **Think of design before implementing**
  - Free vs guided exploration
  - Long vs short run
  - Prank vs actor based fuzzing
- **Building a complex fuzzing harness is *(almost)* more about soft. development than security**

Advanced fuzzing tips

- **Actor based fuzzing**
  - Every actor is itw own contract
  - Harness include an orchestra for actors’ behavior
  - Allows to test complex multiple users transactions

Advanced fuzzing example

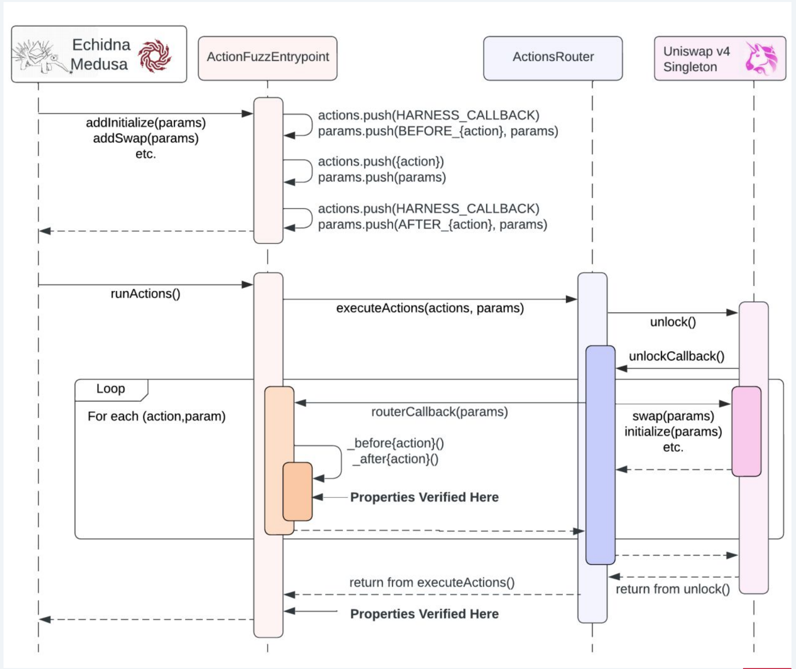
- **Ex: Uniswap V4**
    - Multiple actors
    - Hook system with callbacks

Report

https://t.ly/8N3SZ

Invariants

https://t.ly/9MUEf



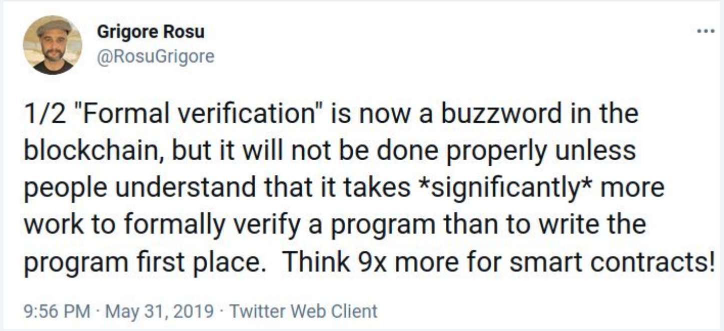
Comparison with similar tools

Other fuzzers

- **Inbuilt in foundry**
    - Might be easier for simple test, however less powerful
- **Echidna**
    - Medusa is the next gen of Echidna

Formal methods based approach

- **KEVM, Certora, ..**
- **Provide proofs, however**
  - More difficult to use
  - Return on investment is significantly higher with fuzzing



Grigore Rosu
@RosuGrigore

1/2 "Formal verification" is now a buzzword in the blockchain, but it will not be done properly unless people understand that it takes *significantly* more work to formally verify a program than to write the program first place. Think 9x more for smart contracts!

9:56 PM · May 31, 2019 · Twitter Web Client

Conclusion

Conclusion

- **To learn more**
  - Secure-contracts.com
  - github.com/crytic/properties
- **Start with invariants in English, then Solidity**
  - Start simple and iterate
  - Try medusa on your current project

Do you want help? Invariant as a service:

https://www.trailofbits.com/contact/