Testing and Verifying Smart Contracts: From Theory to Practice

Formal Methods for Computer Security 2021

Who Am I?

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

- **Trail of Bits: trailofbits.com**
  - We help organizations build safer software
  - R&D focused: we use the latest program analysis techniques
    - McSema https://github.com/lifting-bits/mcsema
    - Manticore https://github.com/trailofbits/manticore
    - Slither https://github.com/crytic/slither
    - Echidna https://github.com/crytic/echidna

Goals

- What is a Blockchain?
- What is a smart contract?
- What program analyses are applied in industry?
- Current challenges and research opportunities

Blockchain

Blockchain

- Ledger: Growing list of records

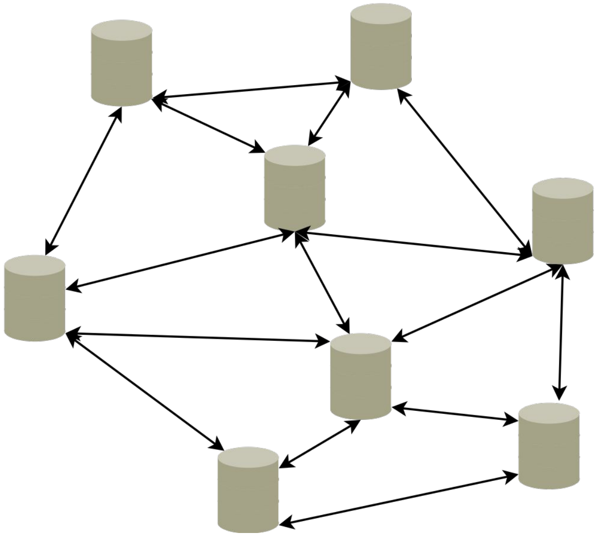
Alice + 100;

Alice - 50; Bob + 50;

...

Blockchain

- Distributed ledger: All participants store all the data
- Decentralized consensus: Everyone agrees on the data



Blockchain Application

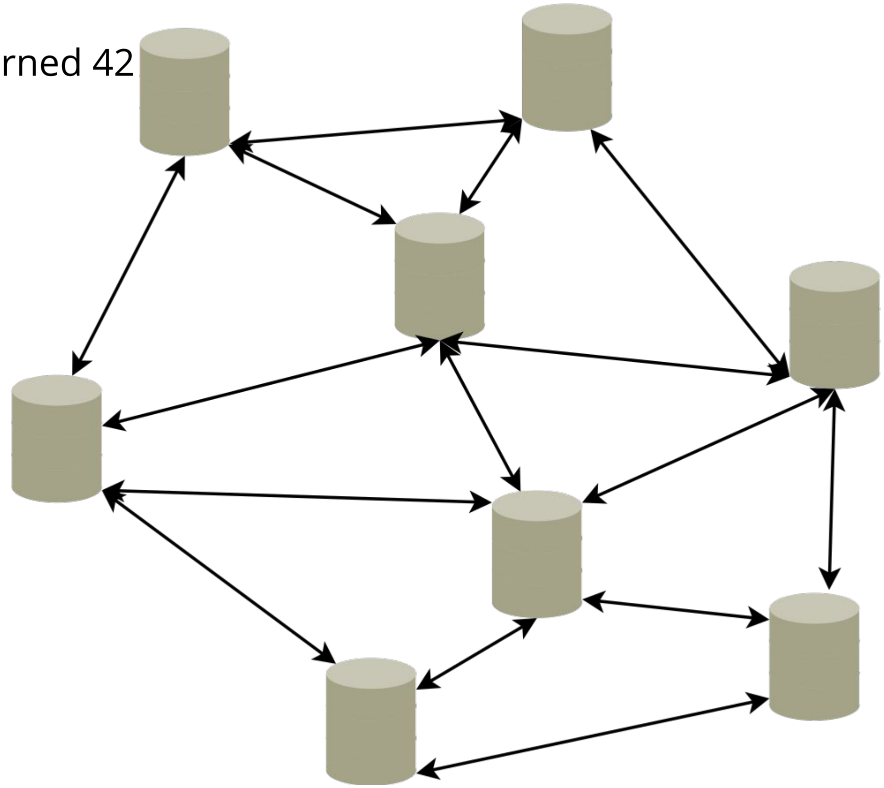
- **Bitcoin[1] (2009): First digital currency using blockchain**
  - Solved the double spend problem

- **Ethereum[2] (2015): Extended blockchain to run apps**
  - Store & execute code

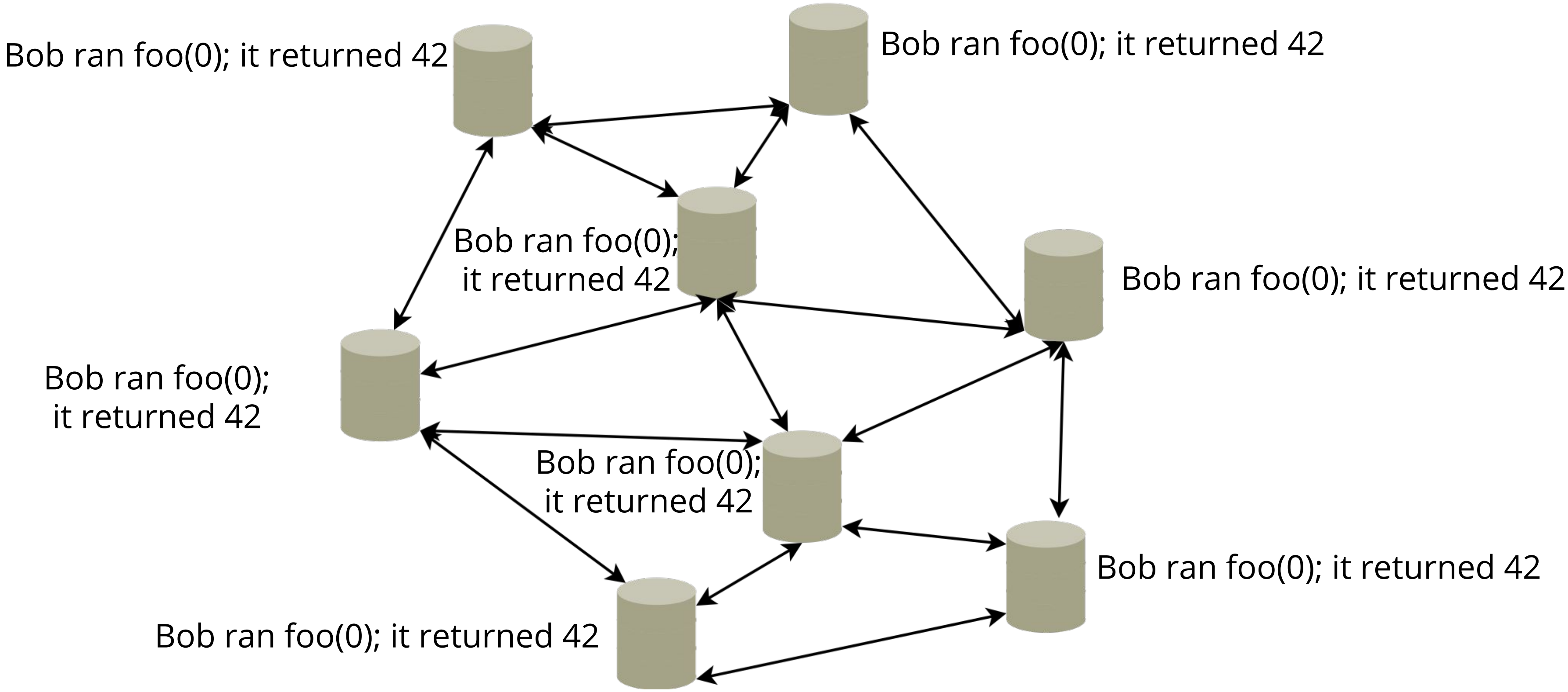
Bitcoin: distributed database => Ethereum: distributed VM

Decentralized Application

Bob ran foo(0); it returned 42



Decentralized Application



Smart Contracts

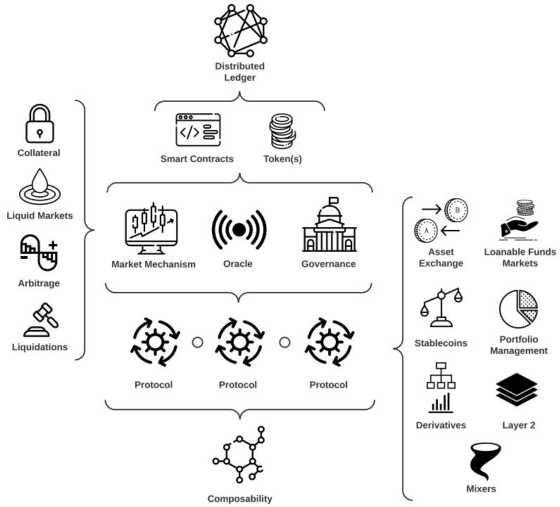
- **Smart Contracts: Applications that run on a blockchain**
  - Everyone executes and verifies it
  - Decentralized: nobody can stop or secretly modify data
  - => Ensures strong properties on your application

Smart Contract Usages

- **Digital currency is one example of an application**
  - ICOs, Crowdfunding system
  - Game (ex: Poker, lotteries, ...)
  - Supply chain
  - ...

DeFi

- **Decentralized Finance (DeFi)**
  - Adapt financial primitives to a permissionless and trusted execution
  - Lending and trading protocols
  - Significant composability



[3]

DeFi

- **A lot of money is invested into smart contracts**
  - ~$40-50B of value locked in major DeFi protocols [4]
  - Uniswap ~$36B in trading volume last month
    - ~5%-10% of crypto trades in decentralized exchanges

Smart Contract Risks

- **Smart contracts are programs = they have bugs**
- **Adversarial environment**
  - Attacker can steal directly funds
  - Rely on cryptographic primitives to hide funds and launder money
- **~$200M stolen in 2020 through smart contract hacks [5]**

Ethereum Internals

EVM

- **Ethereum runs EVM bytecode**
  - VM with <150 opcodes
  - 1 register (PC)
  - Stack-based
- **Calling a function = making a transaction**
  - It has a cost: gas, paid in ethers
- **Bytecode cannot be updated (!)**

```
00000000  PUSH1  0x60
00000002  PUSH1  0x40
00000004  MSTORE
00000005  CALLDATASIZE
00000006  ISZERO
00000007  PUSH2  0x131
0000000a  JUMPI
```

Solidity

- **Smart contracts are typically written in Solidity**
  - High-level language in “Javascript style”
  - Contracts organized as a set of methods
  - State = contract variables + balance (# ethers)

Solidity: Example

```
pragma solidity 0.8.0; // Compiler version
contract Bank{
    mapping(address => uint) private balances;

    constructor(uint initial_supply) public {
       balances[msg.sender] = initial_supply;
    }
    function transfer(address to, uint val) public {
        balances[msg.sender] -= val;
        balances[to] += val;
    }
    function balanceOf(address user) public view returns (uint){
        return balances[user];
    }
}
```

Solidity: Example

```
pragma solidity 0.8.0;
contract Bank{
    mapping(address => uint) private balances;

    constructor(uint initial_supply) public {
       balances[msg.sender] = initial_supply;
    }
    function transfer(address to, uint val) public {
        balances[msg.sender] -= val;
        balances[to] += val;
    }
    function balanceOf(address user) public view returns (uint){
        return balances[user];
    }
}
```

Solidity: Example

```
pragma solidity 0.8.0;
contract Bank{
    mapping(address => uint) private balances;

    constructor(uint initial_supply) public {
       balances[msg.sender] = initial_supply;
    }
    function transfer(address to, uint val) public {
        balances[msg.sender] -= val;
        balances[to] += val;
    }
    function balanceOf(address user) public view returns (uint){
        return balances[user];
    }
}
```

Solidity: Example

```
pragma solidity 0.8.0;
contract Bank{
    mapping(address => uint) private balances;

    constructor(uint initial_supply) public {
       balances[msg.sender] = initial_supply;
    }
    function transfer(address to, uint val) public {
        balances[msg.sender] -= val;
        balances[to] += val;
    }
    function balanceOf(address user) public view returns (uint){
        return balances[user];
    }
}
```

Solidity: Example

```
pragma solidity 0.8.0;
contract Bank{
    mapping(address => uint) private balances;

     constructor(uint initial_supply) public {
       balances[msg.sender] = initial_supply;
    }
    function transfer(address to, uint val) public {
        balances[msg.sender] -= val;
        balances[to] += val;
    }
    function balanceOf(address user) public view returns (uint){
        return balances[user];
    }
}
```

Solidity: Example

```
pragma solidity 0.8.0;
contract Bank{
    mapping(address => uint) private balances;

    constructor(uint initial_supply) public {
       balances[msg.sender] = initial_supply;
    }
    function transfer(address to, uint val) public {
        balances[msg.sender] -= val;
        balances[to] += val;
    }
    function balanceOf(address user) public view returns (uint){
        return balances[user];
    }
}
```

State variable

Constructor

Public function

Constant function (gas-free)

Example Vulnerabilities

Reentrancy

- The DAO (2016)

```
if( ! (msg.sender.call.value(userBalance[msg.sender])() ) ){
    throw;
}

userBalance[msg.sender] = 0;
```

- **Re-enter in the contract before the balance is set to zero**
  - Repeat n times => withdraws n times the original deposit
- **~$70 millions stolen**

Improperly restricted functions

- Parity Wallet (2017)
  - Widely used library for storing ethers
- **Key function was callable by anyone**
  - Someone destructed the contract
  - Broke all third-party integrations
- **$300 million of frozen assets**

Oracle manipulation

- **Harvest Finance: DeFi yield aggregator (2020)**
    - Users deposit assets, and Harvest invest funds into various protocols
    - Bug: incorrect usage of a price Oracle
        - Generate fake price, such that deposit to share ratio is increased
        - Deposit with fake ratio to get more share than expected
        - Replace with original price
        - Withdraw the share and received more than initial deposit
- **~$30M stolen**

Program analysis

Program Analysis

- Smart contracts are small
    - <1,000 LoC
- Gas cost lead to bounded execution
- High value = require high confidence

Program Analysis

- **Fully automated**
  - Detect common patterns
  - Static analysis / symbolic execution
- **Semi-automated**
  - Property-based approach
  - Fuzzing / symbolic execution / abstract interpretation / ...
- **A lot of tools - not all maintained**

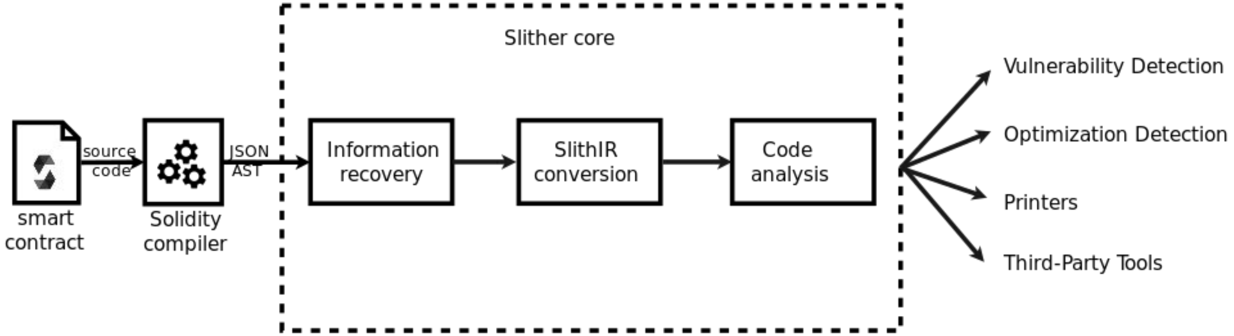
Fully automated

Fully automated

- **Static analysis**
  - Slither [6]
    - ~100 detectors (~70 public)
    - +40 trophies
  - Maru
    - Closed source - SaaS (Mythx.io)
    - 28 detectors

Fully automated - Slither

- **Common flaws**
  - Reentrancy, unprotected function, ...
- **Many language-level issues**
  - Variable shadowing, missing return statements, ...



Fully automated

- Symbolic execution
  - Oyente [7] (< 10 detectors)
- Unmainted tools
  - Securify [8]
  - SmartCheck [9]

Fully automated - Example

Ernst & Young Nightfall

- github.com/EYBlockchain/nightfall/
- zk-SNARK-based platform to allow private asset transfer on Ethereum
- Users deposit assets, and get a “withdrawal proof”, allowing to withdraw the assets with another account

Ernst & Young Nightfall

- *transferFrom* returns a boolean, indicating if the transfer was a success

```
function transferFrom(address, address, uint256) public returns (bool)
```

- Nightfall was not checking the returned value

```

        // Finally, transfer the fTokens from the sender to this contract
        fToken.transferFrom(msg.sender, address(this), _value);

```

- Create a withdrawal proof without transferring the asset
- Found by Slither [15]

Semi-automated

User-defined property

- **User-defined property**
  - DSL or Solidity's `assert`
- **Target business logic**
  - State machine transition
  - Access controls
  - Arithmetic operations
  - External interactions

Semi-Automated

- **Fuzzers**
    - Echidna [10]
    - ContractFuzzer [11]
    - Harvey (Closed source - SaaS (Mythx.io)

Semi-Automated

- **Formal method based approach**
  - Manticore [12] - Symbolic execution
  - K [13] - Symbolic execution
  - Verisol - Solidity to Boogie
  - Mythx - Symbolic execution (Closed source - SaaS (Mythx.io)
  - Certora - Abstract interpretation (Closed source)

Semi-Automated

- **Fuzzing versus formally-based methods**
  - From experience, fuzzing is more effective to finds bugs
  - But formal methods lead to higher confidence
- **Require expertise and deep understanding of the target**

Semi-Automated

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

Semi-automated - Example

Balancer

- https://balancer.finance
- **Trading platform**
  - Liquidity provider earn interests
    - Bookkeeping: the share of the pool’s liquidity, not of the assets sent
  - Complex arithmetics

Balancer

- "How many assets I should send to receive `poolAmountOut` liquidity share?"

```
function joinPool(uint poolAmountOut, uint[] calldata maxAmountsIn)
     external
     _logs_
     _lock_
 {
     require(_finalized, "ERR_NOT_FINALIZED");

     uint poolTotal = totalSupply();
     uint ratio = bdiv(poolAmountOut, poolTotal);
     require(ratio != 0, "ERR_MATH_APPROX");

     for (uint i = 0; i < _tokens.length; i++) {
         address t = _tokens[i];
         uint bal = _records[t].balance;
         uint tokenAmountIn = bmul(ratio, bal);
```

Balancer

- **Fixed-point arithmetic**
  - c = ((a * b) + BONE / 2) / BONE
  - If ((a * b) + BONE / 2) **<** BONE, returns 0

```
function bmul(uint a, uint b)
      internal pure
      returns (uint)
 {
      uint c0 = a * b;
      require(a == 0 || c0 / a == b, "ERR_MUL_OVERFLOW");
      uint c1 = c0 + (BONE / 2);
      require(c1 >= c0, "ERR_MUL_OVERFLOW");
      uint c2 = c1 / BONE;
      return c2;
 }
```

Balancer

- You could receive pool’s share for free for pool with low liquidity
- Found with Echidna & Manticore

Semi-automated - Limitations

Property limitations

- Aave was “formally verified”

35. Integrity of deposit ✔*

When actor ***u*** deposits ***x*** tokens of asset ***t*** on behalf of actor ***b*** (can be ***a***)
The asset balance of ***u*** is decreased and the aToken of **b** is increased.

```
{ t_ = t.balanceOf(u)  ∧  a_ = getAToken(t).balanceOf(b) }
      deposit(u, t, x, b );
{ t.balanceOf(u) = t_ - x ∧ getAToken(t).balanceOf(b) = a_ + x   }
```

- Bug was found [16], allowed for property break
- Verification did not consider the code in its whole architecture

Program analysis in practice

Industry usage

- **Fully automated tool - Slither**
  - All our audits
- **Semi-automated tools - Echidna/Manticore**
  - ~50% of the audits
  - Some clients write properties before our engagements

Industry Usage

- Example: [Yield Protocol]
- Different levels of properties
  - End-to-end
  - Scenario-based
  - Single component property

General properties

#	Property	Result
1	Calling `erase` in the Controller never reverts.	PASSED
2	Calling `locked` in the Controller never reverts.	PASSED
3	Calling `powerOf` in the Controller never reverts.	PASSED
4	Calling `totalDebtDai` in the Controller never reverts.	PASSED
5	Posting, borrowing, repaying, and withdrawing using `CHAI` as collateral properly updates the state variables.	PASSED
6	Posting, borrowing, repaying, and withdrawing using `WETH` as collateral properly updates the state variables.	PASSED
7	All the WETH balances are above dust or zero in the Controller.	**FAILED** (TOB-YP-006)
8	All the WETH balances are above dust or zero in the Liquidations.	PASSED
9	Calling `price` never reverts on Liquidations	PASSED
10	Transferring tokens to the null address (`0x0`) causes a revert.	PASSED
11	The null address (`0x0`) owns no tokens.	PASSED
12	Transferring a valid amount of tokens to a non-null address reduces the current balance.	PASSED
13	Transferring an invalid amount of tokens to a non-null address reverts or returns false.	PASSED

 Yield Protocol Assessment | 8

Program analysis challenges

Challenges - Engineering

- **Not all tools have the same maturity**
- **Space evolving fast**
  - Solidity/EVM updates
  - New application trends require new heuristics
- **No property writing standard**
  - Solidity's assert, but limited

Challenges - Research

- **Contract composability**
  - Small code, but high interactions
- **Solidity/EVM specificity**
  - Array indexes are the results of hash functions
  - Gas modeling
- **Application specific modeling**
  - DeFi
- **Combining techniques**

Conclusion

Conclusion

- **Blockchain: new technology**
  - With challenges and research opportunities for program analysis
- **Tools are already helping developers and auditors**

- **Crytic $10k Prize**
  - Reward academic publications built on top of ToB tools (inc. Slither/Echidna/Manticore)

References

1. Bitcoin: A Peer-to-Peer Electronic Cash System - Satoshi Nakamoto
2. https://ethereum.github.io/yellowpaper/paper.pdf - G.Wood
3. SoK: Decentralized Finance (DeFi), Sam M. Werner and al. (preprint)
4. Dex Volume, Dex to Cex Spot trade volume (%) - dex-non-custodial
5. The 2021 Crypto Crime Report - Chainalysis
6. Slither: A Static Analysis Framework For Smart Contracts, Josselin Feist and al - WETSEB '19
7. Making Smart Contracts Smarter, Loi Luu and al - CCS16
8. Securify: Practical Security Analysis of Smart Contracts, Petar Tsankov and al - CCS18
9. SmartCheck: Static Analysis of Ethereum Smart Contracts, Sergei Tikhomirov and al - WETSEB18
10. Echidna: effective, usable, and fast fuzzing for smart contracts, Gustavo Grieco, Will Song, Artur Cygan, Josselin Feist, Alex Groce - ISSTA '20
11. ContractFuzzer: Fuzzing Smart Contracts for Vulnerability Detection, Bo Jiang - ASE18
12. Manticore: A User-Friendly Symbolic Execution Framework for Binaries and Smart Contracts, Mark Mossberg and al - WETSEB18

References

13. Kevm: A complete formal semantics of the ethereum virtual machine, Everett Hildenbrandt and al - CSF18
14. Formal Specification and Verification of Smart Contracts for Azure Blockchain, Yuepeng Wang and al
15. Bug Hunting with Crytic
16. Breaking Aave Upgradeability