ItyFuzz: Snapshot-Based Fuzzer for Smart Contract

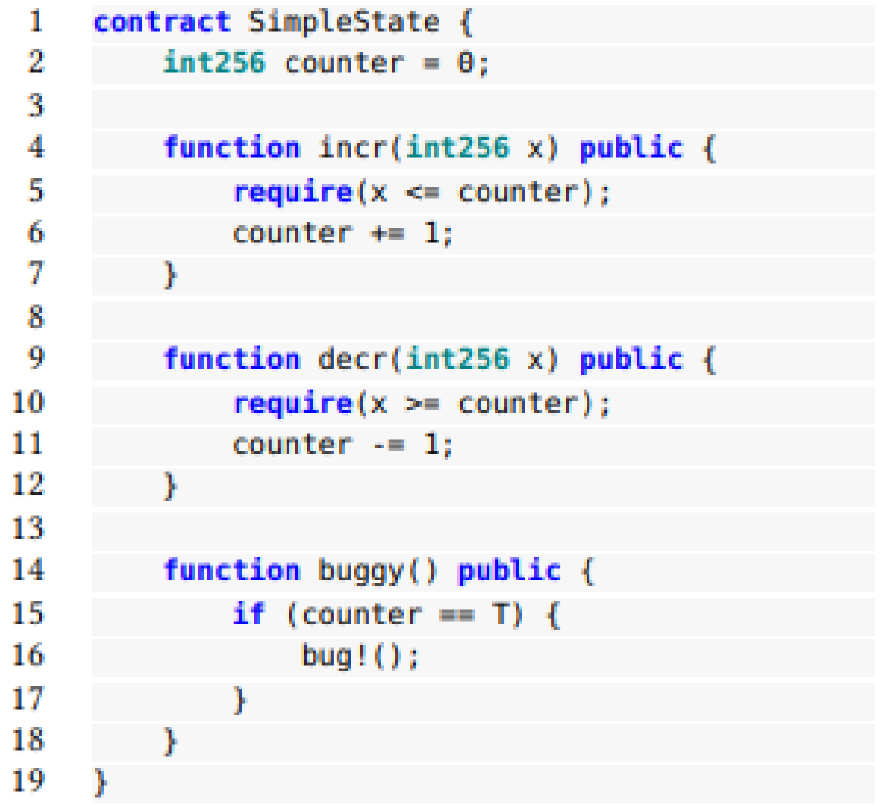
ISSTA2023

Chaofan Shou Shangyin TanKoushik Sen

UC Berkeley Berkeley

Smart Contract

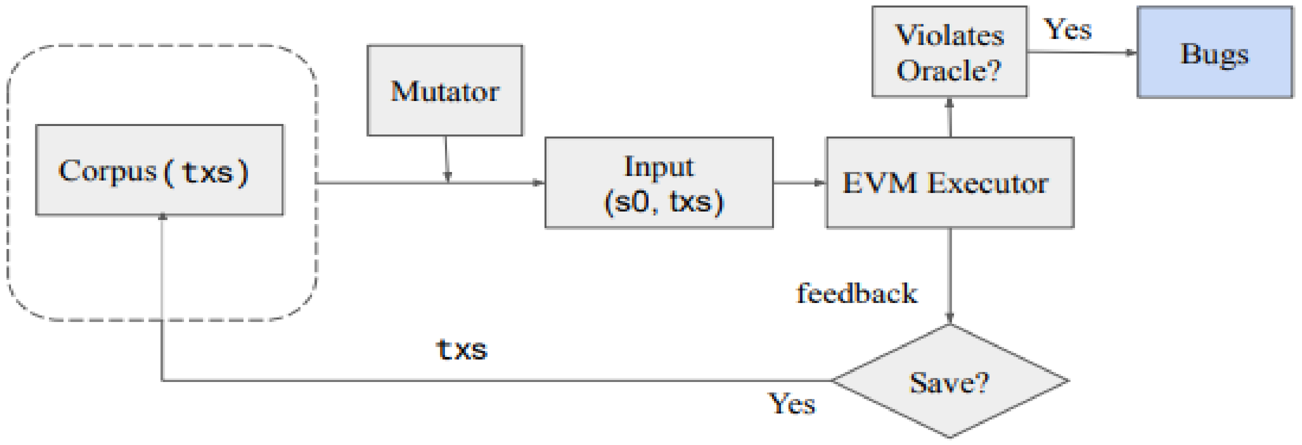
- ➢ A program running on the blockchain
- ➢ Transaction
  - **Invoking contract function**
  - token transfer
- ➢ The state of the contact
  - A Set of persistent data
    - persistent variables
  - Key/value pair
    - e.g., *counter => 0*

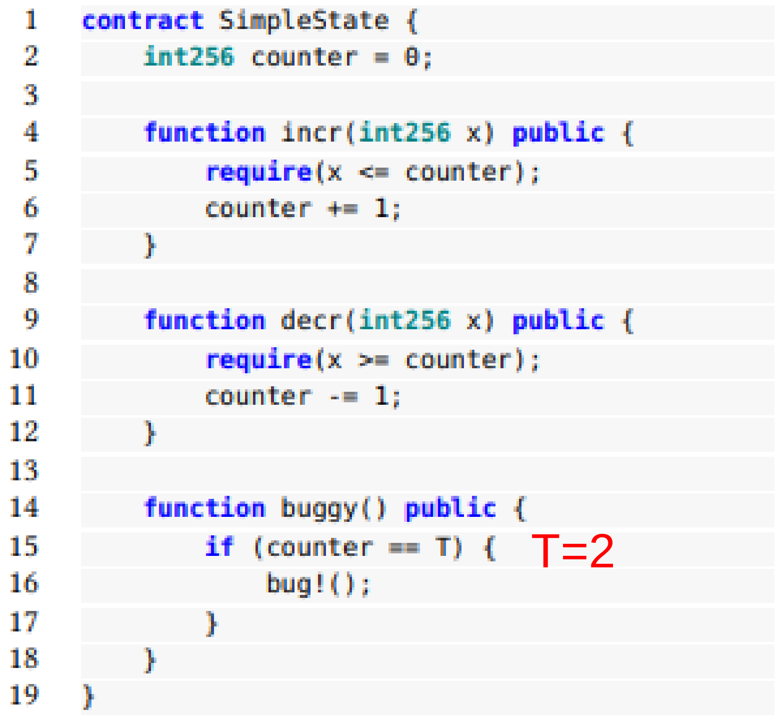


```
contract SimpleState {
    int256 counter = 0;

    function incr(int256 x) public {
        require(x <= counter);
        counter += 1;
    }

    function decr(int256 x) public {
        require(x >= counter);
        counter -= 1;
    }

    function buggy() public {
        if (counter == T) {
            bug!();
        }
    }
}
```

Fuzzing

- Feedback
  - bug&Flaw
  - branch coverage
  - state





```
contract SimpleState {
    int256 counter = 0;

    function incr(int256 x) public {
        require(x <= counter);
        counter += 1;
    }

    function decr(int256 x) public {
        require(x >= counter);
        counter -= 1;
    }

    function buggy() public {
        if (counter == T) {
            bug!();
        }
    }
}
```

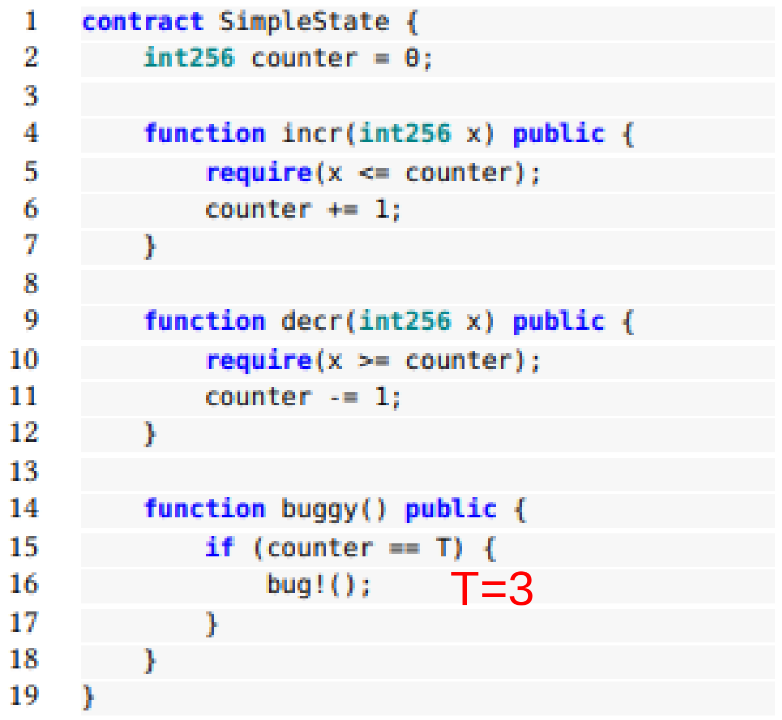
T=2

- explores transactions seqence
  - Iteration1: Incr(0)
  - Iteration2: Incr(0) => incr(1)
  - Iteration3: Incr(0) => incr(1) => buggy()

Research Problem

- ➢ Design a high-speed fuzzer with real-time and on-chain analysis capabilities
  - Providing real time *reponse* to cease the attack.
  - More real-time fuzzing contracts
    - Update external contract information in real time.
  - More comprehensive fuzzing test
    - leveraging the pratical state on the chain.

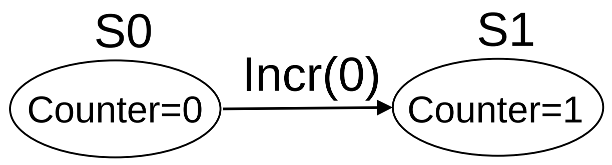
Previous research



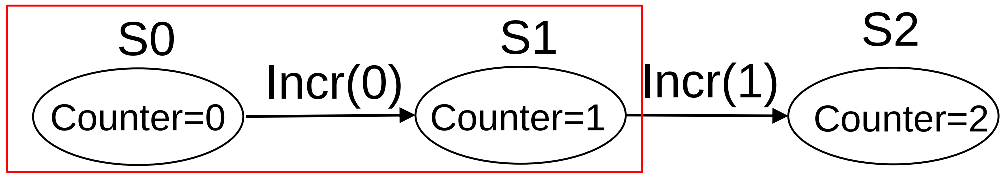
```
contract SimpleState {
    int256 counter = 0;

    function incr(int256 x) public {
        require(x <= counter);
        counter += 1;
    }

    function decr(int256 x) public {
        require(x >= counter);
        counter -= 1;
    }

    function buggy() public {
        if (counter == T) {
            bug!();
        }
    }
}
```

T=3

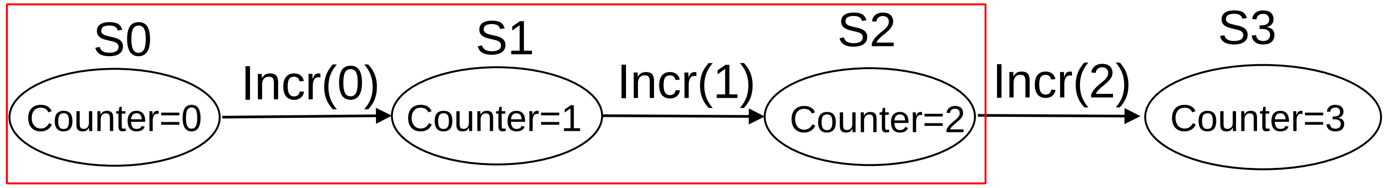
- Smartian[ase22]
    - Iteration1: incr(0)



    - Iteration2 Incr(0) => incr(1)



    - Iteration3: Incr(0) => incr(1) => incr(2)

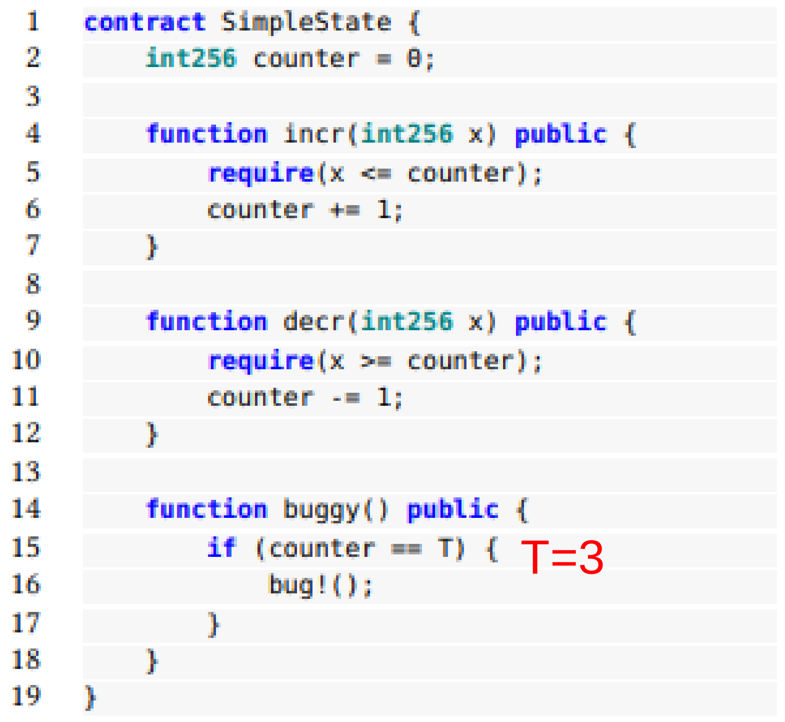


    - Iteration4: Incr(0) => incr(1) => incr(2) => buggy()

- **Obervation**:

re-execution takes more than **90 %** of the total fuzzing time.

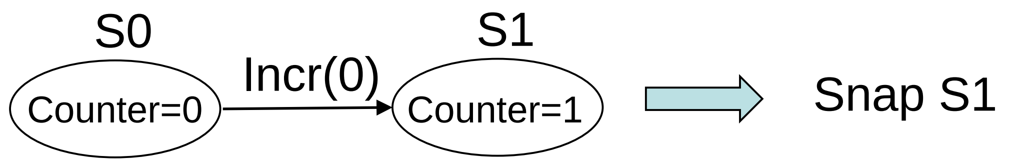
Main idea



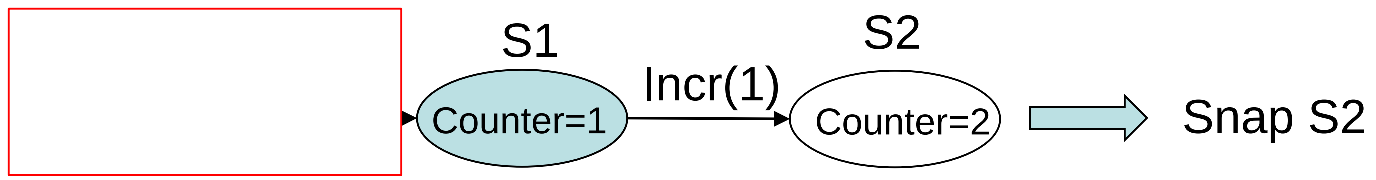
```
contract SimpleState {
    int256 counter = 0;

    function incr(int256 x) public {
        require(x <= counter);
        counter += 1;
    }

    function decr(int256 x) public {
        require(x >= counter);
        counter -= 1;
    }

    function buggy() public {
        if (counter == T) {
            bug!();
        }
    }
}
```

T=3

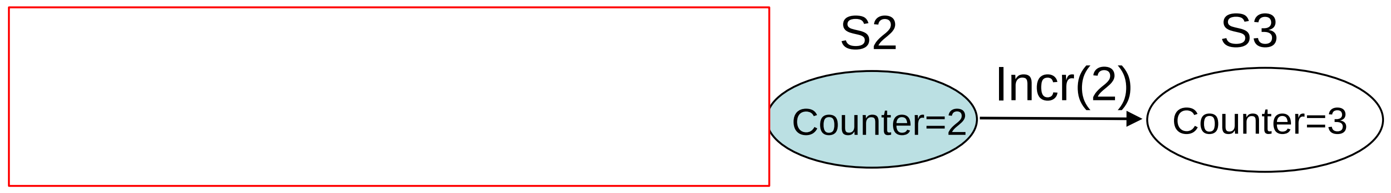
- Idea: snapshot the intermediate state
  - Iteration1: incr(0)



  - Iteration2 Incr(0) => incr(1)



  - Iteration3: Incr(0) => incr(1) => incr(2)



  - Iteration4: Incr(0) => incr(1) => incr(2) => buggy()

Challenge

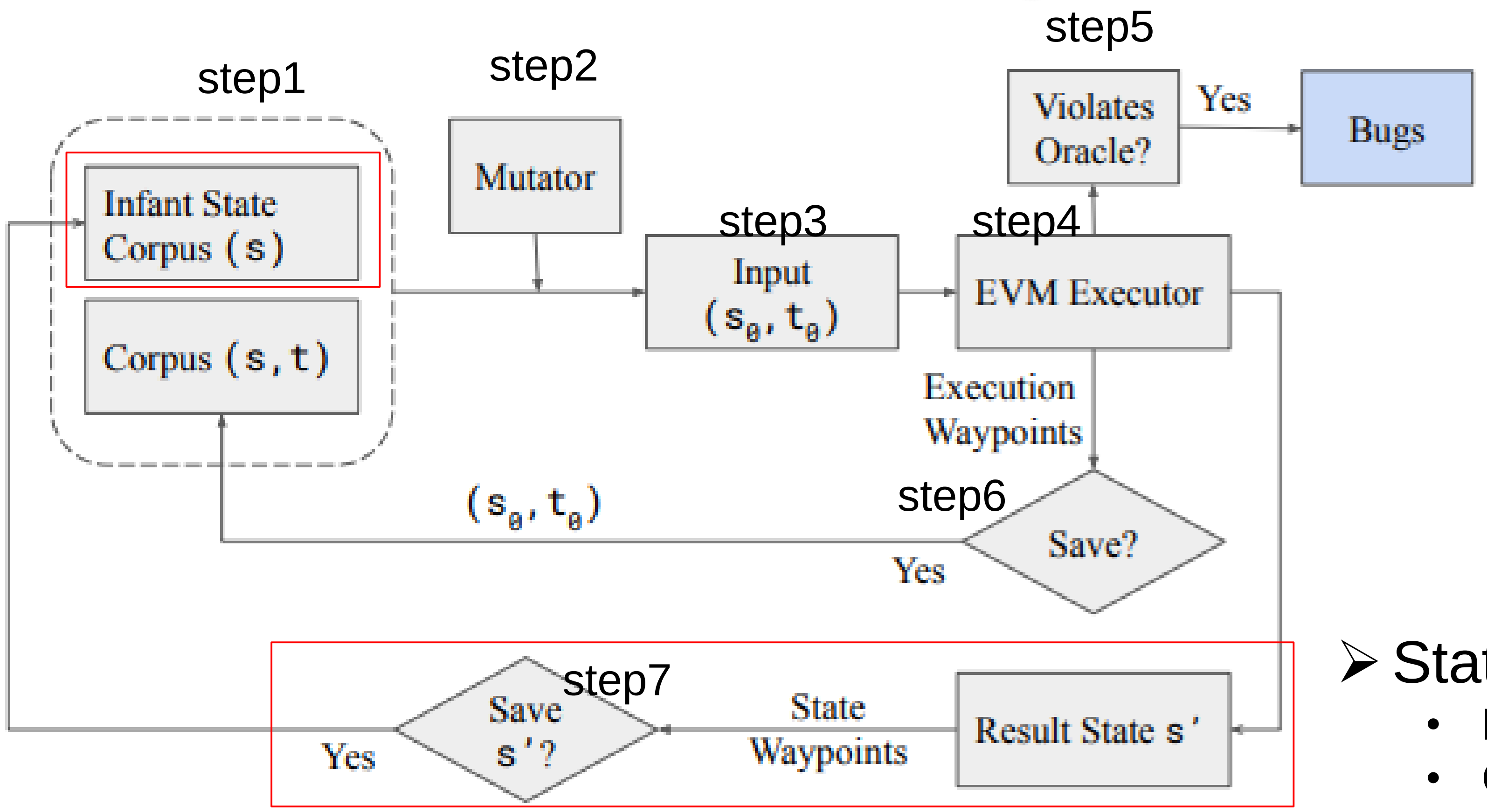
- ➢ State explosion
  - The size of the stored state snapshots could grow to several gigabytes in a few seconds.

- ➢ Solution
  - Only select **Important** state to state snapshots
    - Dataflow waypoint
    - Comparison waypoint
  - Prune the state snapshots.

Contribution

- The author presents a novel snapshot-based fuzzing algorithm to reduce re-execution overhead for stateful smart contract fuzzing, dubbed ityFuzz.
- The author creates new waypoint mechanisms optimized the scale of state snapshot.
- Based on ItyFuzz, the author propose a new auditing method for smart contracts to conduct testing based on state fetched from the blockchain on the fly

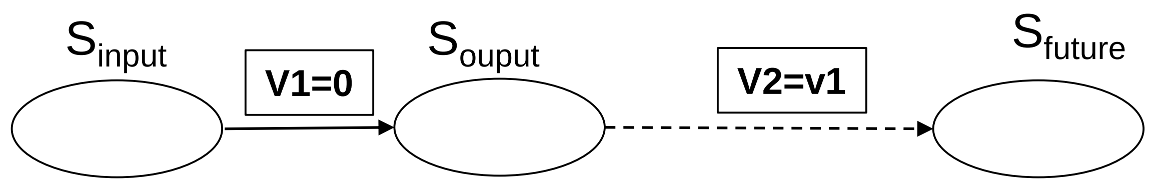
Design



- ➢ State waypoint
  - Dataflow waypoints
  - Comprasion waypoints

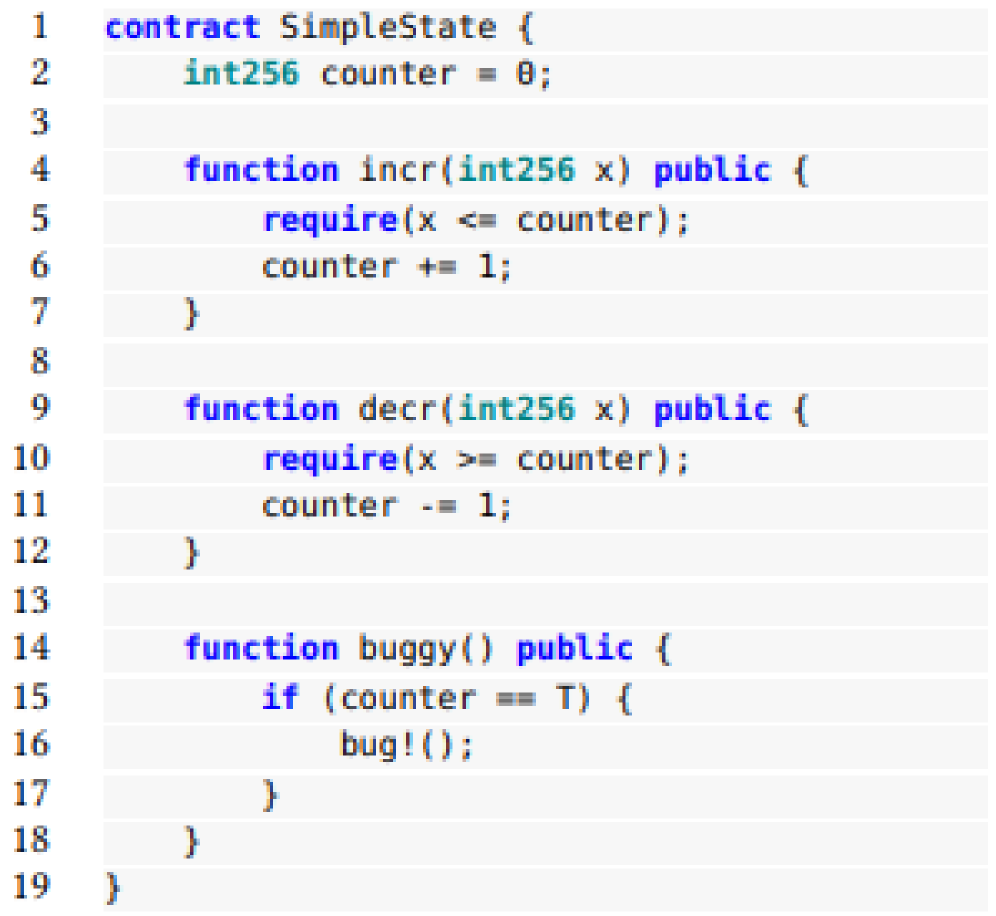
State waypoints

- **Dataflow waypoint**



- **Comparison waypoint**
  - Make the conditional closer to being Satisfied

  e.g., *counter* and *T*



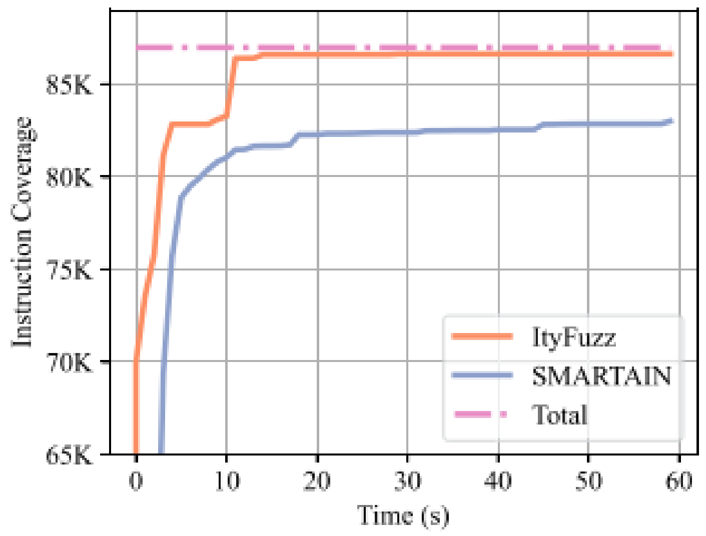
```
contract SimpleState {
    int256 counter = 0;

    function incr(int256 x) public {
        require(x <= counter);
        counter += 1;
    }

    function decr(int256 x) public {
        require(x >= counter);
        counter -= 1;
    }

    function buggy() public {
        if (counter == T) {
            bug!();
        }
    }
}
```

Evaluation

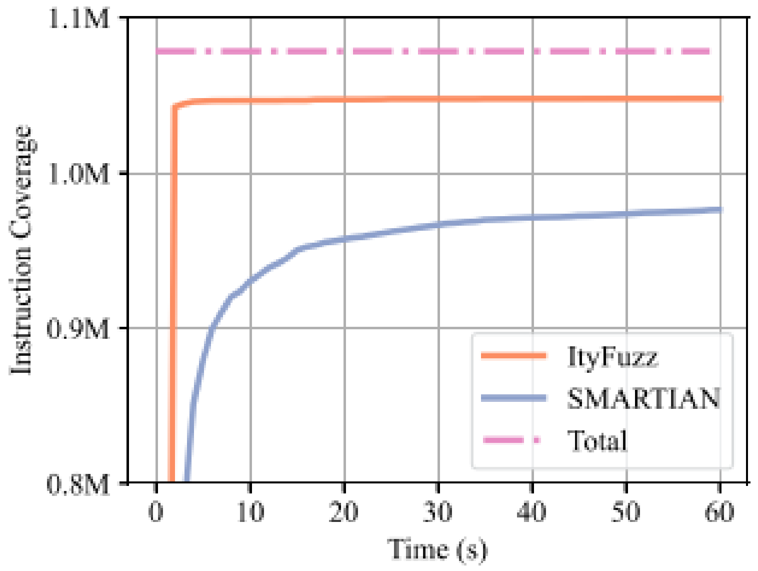
- ➢ Setup
  - AMD Epyc CPUs (128 cores)
  - 256 GB memory.
  - All ablations and other tools used in the evaluation are compiled with optimization (-O3).
- ➢ Metric
  - Coverage
  - State overhead
  - Vulnerabilities
  - On-Chain Audition

Evaluation

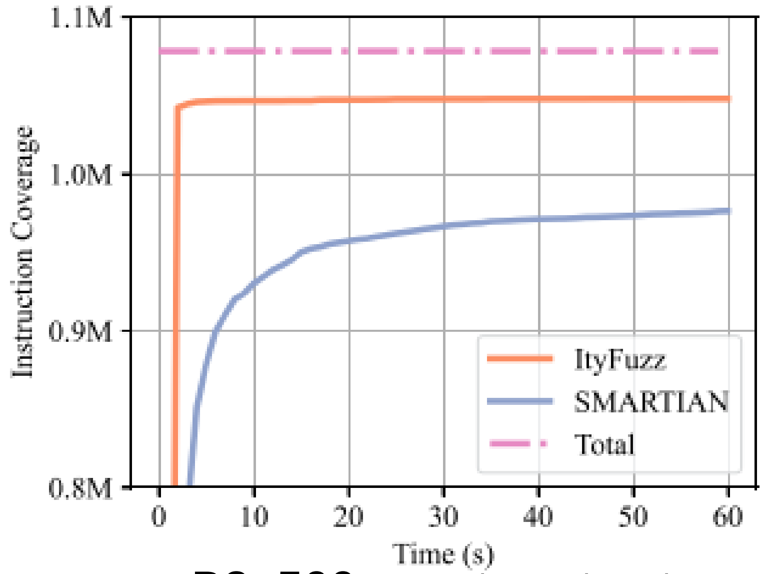
➢ Code Coverage



B1: 57 tokens contracts



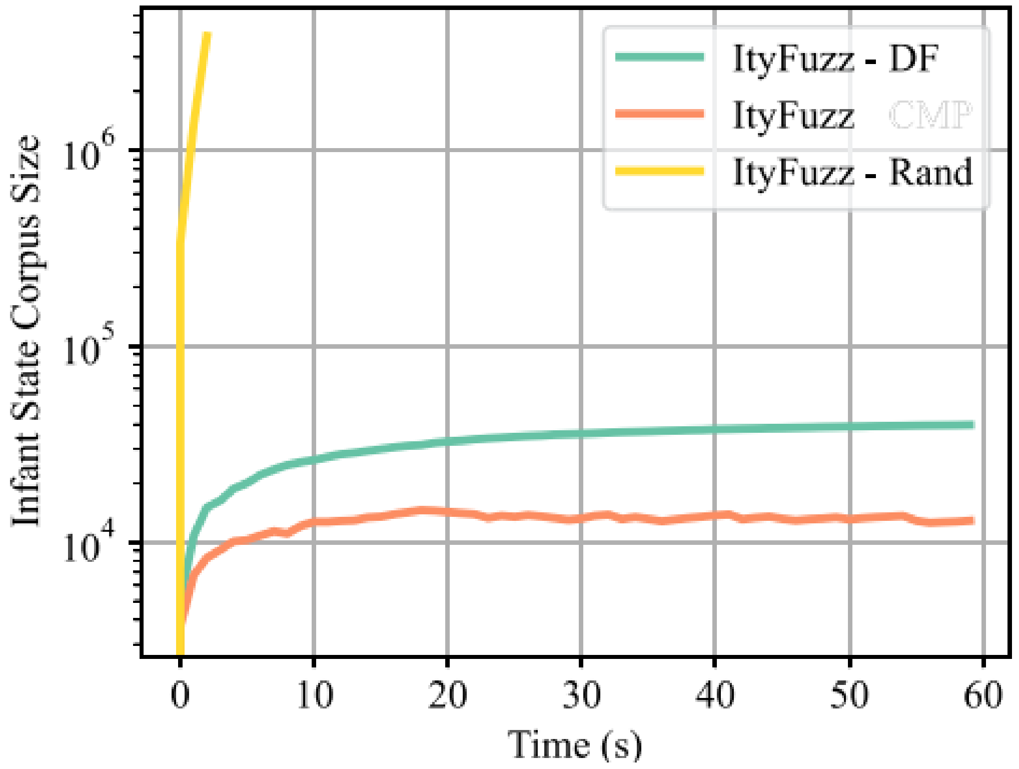
B2: 72 smart contracts



B3: 500 smart contracts

Evaluation

- State Overhead
  - ItyFuzz- Rand: snapshots states **with a likelihood of 50%.**
  - ItyFuzz- DF: snapshots states based **on only dataflow waypoint.**



Evaluation

- Vulnerabilities
  - Authors gathered **42 previously exploited projects**, ItyFuzz was able to identify concrete exploits for **36 of them**, with an average time of **13.8 seconds**.
  - Authors also applied tool to **45000 smart contract projects** (with more than **150k smart contracts**), ItyFuzz is able to exploits for stealing assets valued at over **$500k** among **21 vulnerable projects**.

Evaluation

- On-Chain Auditing

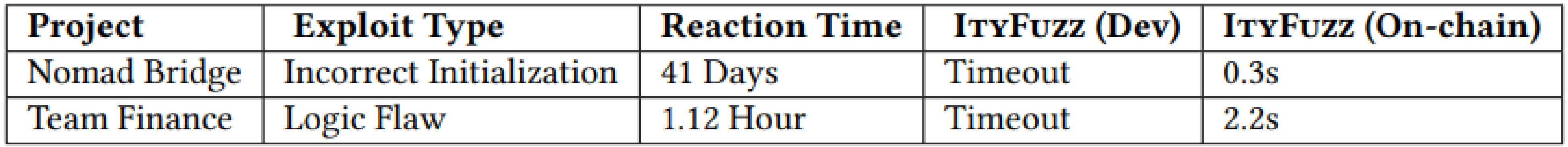


Table 2: Vulnerability Detection Time

Project	Exploit Type	Reaction Time	ItyFuzz (Dev)	ItyFuzz (On-chain)
Nomad Bridge	Incorrect Initialization	41 Days	Timeout	0.3s
Team Finance	Logic Flaw	1.12 Hour	Timeout	2.2s

Conclusion

- The author design a new snapshot-based fuzzer *ItyFuzz* for testing smart contracts that effectively stores intermediate states to reduce re-execution overhead.
- The author multiple customized waypoint mechanisms to efficient categorize and store interesting states for better program explorations.
- we can perform on-chain auditing to identity and prevent exploits for realworld smart contract applications.