a place of mind
THE UNIVERSITY OF BRITISH COLUMBIA

Towards Effective Static Analysis Approaches for Security Vulnerabilities in Smart Contracts

Asem Ghaleb

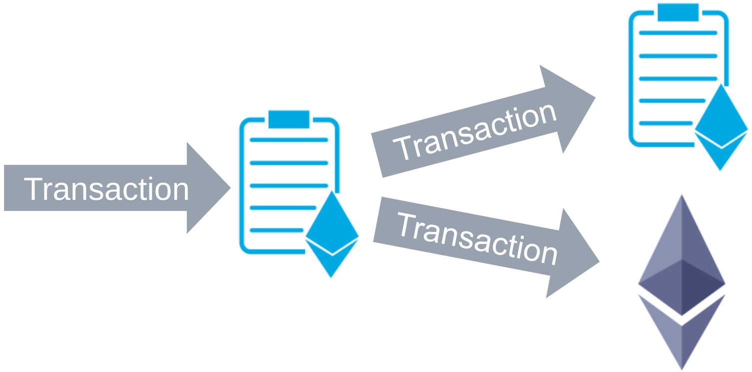
PhD Candidate @ UBC, Canada

Advisors: Karthik Pattabiraman and Julia Rubin

ASE 2022 Doctoral Symposium

October 10th, 2022

Ethereum smart contracts



Increasing adoption

- Finance, supply chain, gaming, etc
- Hold nearly 23% of Ethereum supply (~$161B), as of Sep 2022 [1] [2]

[1] https://etherscan.io/stat/supply

[2] https://crypto.news/23-ether-eth-supply-locked-smart-contracts

Security vulnerabilities in smart contracts

- Several attack incidents

(2016) The DAO Attacked: Code Issue Leads to $60 Million Ether Theft

(2017) Yes, this kid really just deleted $300 Million by messing around with Ethereum's smart contracts

(2021) ValueDeFi: $10 Million lost due to a basic mistake by the development team

(2022) Found a critical bug that could have blocked all future actions from a contract-owning governance system.

Vulnerability example

(2021) ValueDeFi: $10 Million lost due to a basic mistake by the development team

```
contract ProfitSharingRewardPool{
  address operator = msg.sender;
  bool initialized = false;
   modifier onlyOperator {
     require (operator == msg.sender);
       _;
     }
   function initialize() public {
      require (!initialized);
       // omitted code
      operator = msg.sender;
                                            initialized = true;
     }

    function governanceRecoverUnsupported external onlyOperator{
       //omitted code
     }

   }
```

Static analysis tools: current state

- Tools with high false-negatives and false-positives
- Our evaluation shows that static tools:
  - ➢ Search for predefined syntactic patterns
    - ➔ Fail on simple variations
    - ➔ Over-approximate
  - ➢ Enumerate symbolic traces
    - ➔ Sequence of transactions to trigger most vulnerabilities
    - ➔ Path explosion and scalability issues

Thesis goal

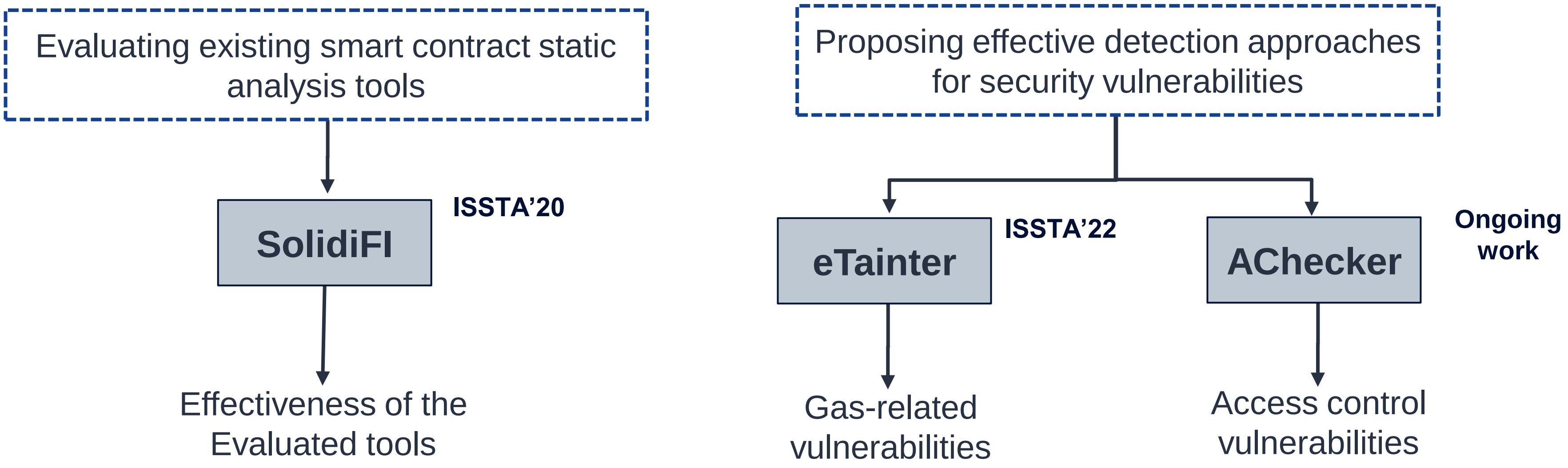
Build effective static analysis approaches for detecting security vulnerabilities in smart contracts

- Evaluating existing smart contract static analysis tools
- Proposing effective detection approaches for security vulnerabilities

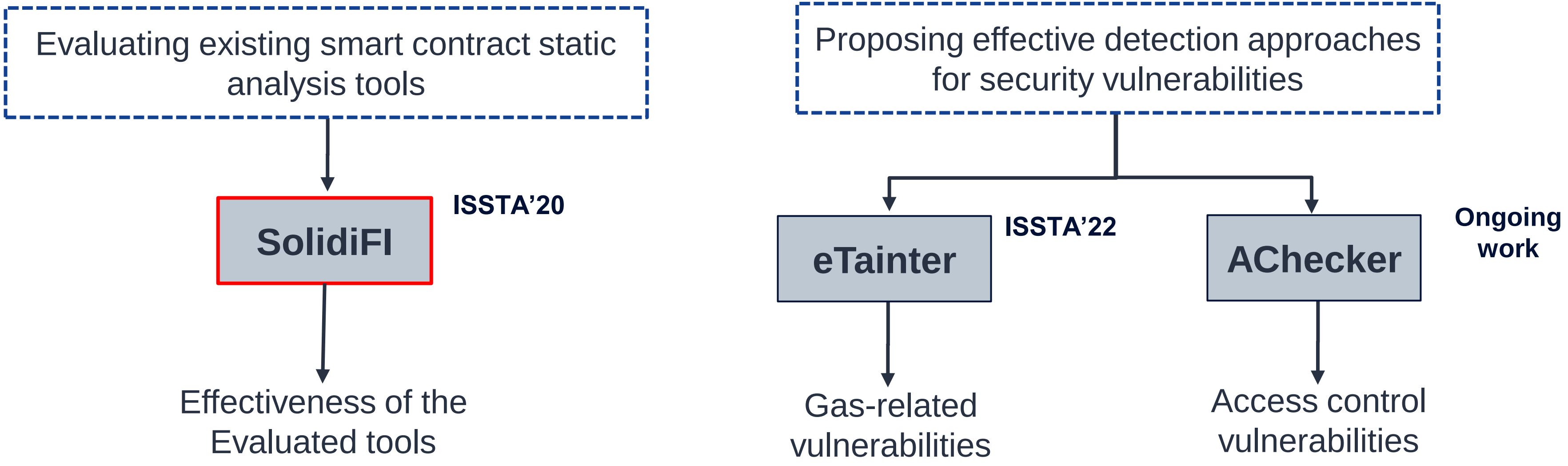
Solution insight

Find generic security properties and use lightweight static analysis to find violations of these properties

Contributions overview



Contributions overview



SolidiFI source code: https://github.com/DependableSystemsLab/SolidiFI

Goal

- Code vulnerabilities are still reported frequently
- No evaluation methodology of static analyzers

A systematic approach for evaluating efficacy of smart contract static analysis tools on detecting bugs

- **<u>Key Idea:</u>** inject bugs into the source code of smart contracts

Findings summary

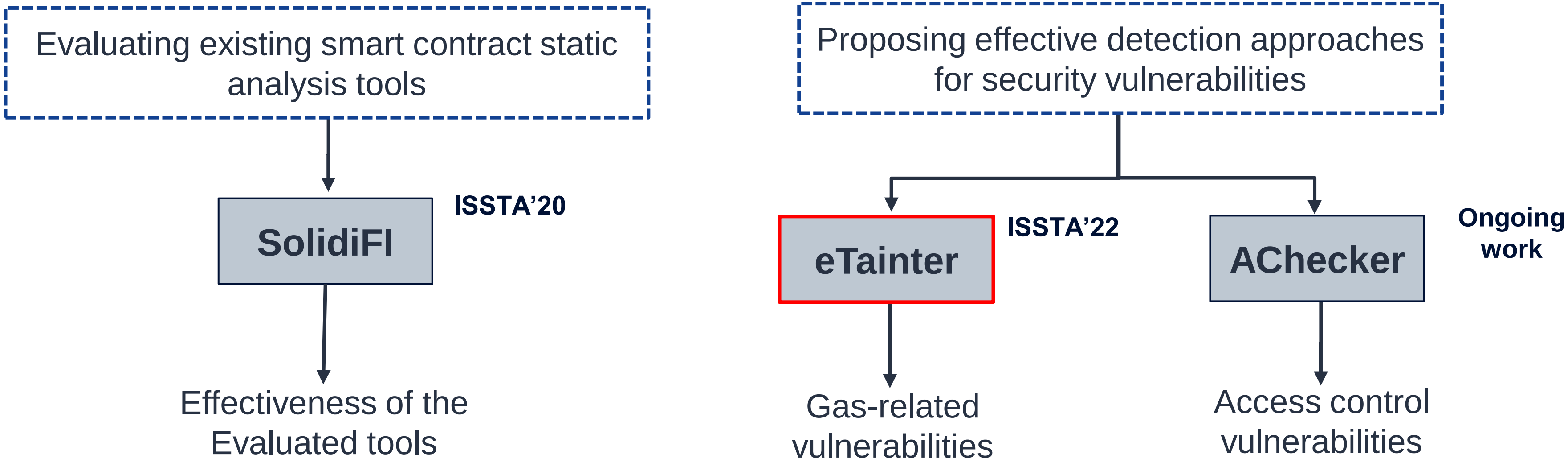
- All tools have many undetected cases
- All tools reported false positives
- Tools with low false negatives reported high false positives

Analyzers that **detect bugs** with **low false positives** are needed

SolidiFI artifact:

https://github.com/DependableSystemsLab/SolidiFI-benchmark

Contributions overview



eTainter source code: https://github.com/DependableSystemsLab/eTainter

Smart contracts: Gas concept

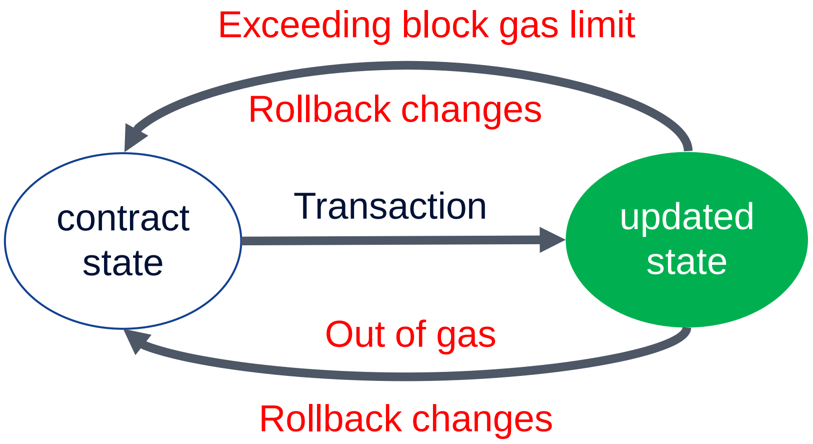
- Executing contract costs gas
- Gas cost for every EVM low-level instruction (opcode)
- Contract’s users pay the gas cost

User specifies
- Function to call
- Max gas user wants to spend on executing the function

```
                 Gas cost
PUSH1 0x64  |3
SWAP1       |3
CALLVALUE   |2
MUL         |5
PUSH1 0x02  |3
SLOAD       |100/2100
PUSH1       |3
SWAP1       |3
DUP2        |3
MSTORE      |X
PUSH1 0x08  |3
PUSH1 0x20  |3
MSTORE      |X
```

EVM bytecode opcodes

Gas-related attacks and consequences



- Dependency on gas can result in vulnerabilities
- Attackers increase gas cost to force unwanted behavior (e.g., DoS)

eTainter approach: Example

```
contract PIPOT {
  struct order {
    address player;
    uint betPrice;
   }
  mapping (uint => order[]) orders ;

  function buyTicket (uint betPrice) public payable {
    orders[game].push(order(msg.sender, betPrice));
     //some code
    }

  function pickTheWinner(uint winPrice) public {
     //some code
     for(uint i=0; i< orders[game].length; i++){
        if (orders[game][i].betPrice == winPrice){
           orders[game][i].player.transfer(toPlayer);
          }
       }
    }
```

Taint tracking

Sink: i< orders[game].length

Sources:

msg.sender
betPrice
winPrice

eTainter approach: Example

```
contract PIPOT {
  struct order {
    address player;
    uint betPrice;
   }
  mapping (uint => order[]) orders ;

  function buyTicket (uint betPrice) public payable {
    orders[game].push(order(msg.sender, betPrice));
     //some code
    }

  function pickTheWinner(uint winPrice) public {
     //some code
     for(uint i=0; i< orders[game].length; i++){
        if (orders[game][i].betPrice == winPrice){
          orders[game][i].player.transfer(toPlayer);
         }
      }
    }
```

Taint tracking

Sink: i< orders[game].length

Sources:

msg.sender
betPrice
winPrice
orders[game]<needs validation>

Storage sink: orders[game]

eTainter approach: Example

```
contract PIPOT {
  struct order {
    address player;
    uint betPrice;
   }
  mapping (uint => order[]) orders ;

  function buyTicket (uint betPrice) public payable {
    orders[game].push(order(msg.sender, betPrice));
     //some code
    }

  function pickTheWinner(uint winPrice) public {
     //some code
     for(uint i=0; i< orders[game].length; i++){
        if (orders[game][i].betPrice == winPrice){
          orders[game][i].player.transfer(toPlayer);
         }
      }
   }
```

Taint written to orders[game] array

Taint tracking

Sink: i< orders[game].length

Sources:

msg.sender
betPrice
winPrice
orders[game]<needs validation>

Storage sink: orders[game] tainted

eTainter approach: Example

```
contract PIPOT {
  struct order {
    address player;
    uint betPrice;
   }
  mapping (uint => order[]) orders ;

  function buyTicket (uint betPrice) public payable {
   orders[game].push(order(msg.sender, betPrice));
     //some code
    }

  function pickTheWinner(uint winPrice) public {
     //some code
     for(uint i=0; i< orders[game].length; i++){
        if (orders[game][i].betPrice == winPrice){
          orders[game][i].player.transfer(toPlayer);
         }
      }
    }
```

Taint written to orders[game] array

Taint tracking

Sink: i< orders[game].length

Sources:

msg.sender
betPrice
winPrice
orders[game]<source of taints>

Storage sink: orders[game] tainted

eTainter approach: Example

```
contract PIPOT {
  struct order {
    address player;
    uint betPrice;
   }
  mapping (uint => order[]) orders ;

  function buyTicket (uint betPrice) public payable {
    orders[game].push(order(msg.sender, betPr
     //some code
    }

  function pickTheWinner(uint winPrice) public {
     //some code
     for(uint i=0; i< orders[game].length; i++){
        if (orders[game][i].betPrice == winPrice){
          orders[game][i].player.transfer(toPlayer);
         }
      }
   }
```

Loop is unbounded

Taint reaches sink (loop exit condition)

Taint tracking

Sink: i< orders[game].length

Sources:

msg.sender
betPrice
winPrice
orders[game]<source of taints>

Storage sink: orders[game] tainted

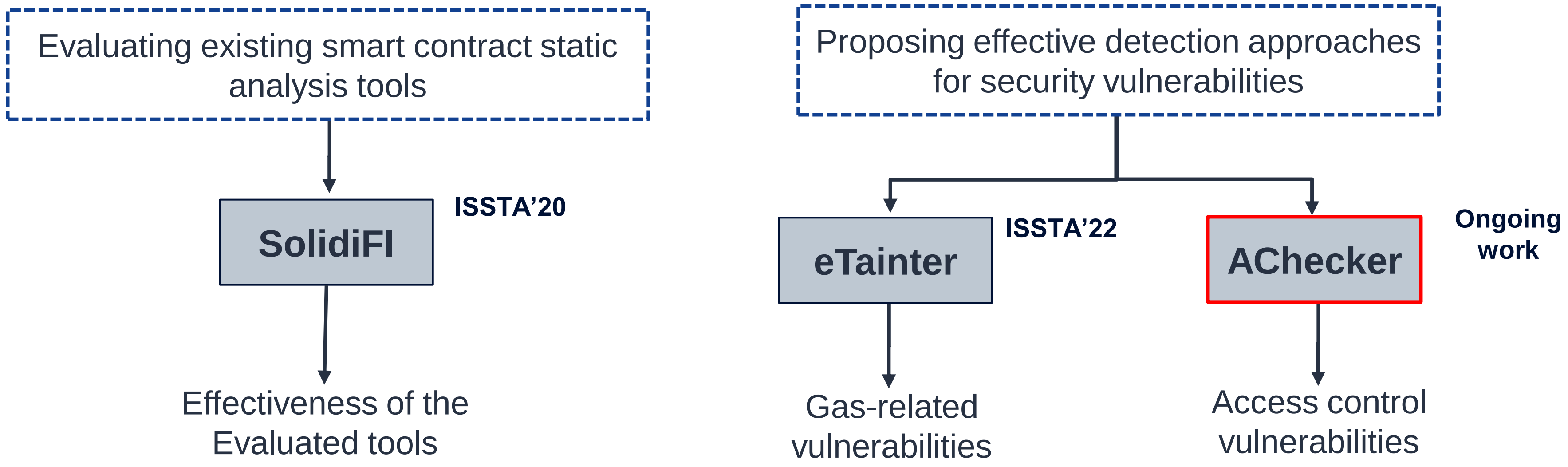
Findings summary

- eTainter achieved **92%** F1 score compared to **69%** for prior work (MadMax)
- Practical analysis time (8 seconds)
- Flagged 2,800 unique contracts on Ethereum as vulnerable
- Flagged 71 contracts of the most frequently used contracts on Ethereum

eTainter artifact:

https://github.com/DependableSystemsLab/eTainter

Contributions overview



Smart contracts: Access control

- Lack of built-in permission-based security model
- Access control implemented in ad-hoc manner
- Results in several access control vulnerabilities
  - Weak AC checks
  - Unprotected code statements

AChecker approach: Example

```
contract Wallet{
  address owner = msg.sender;
  modifier onlyOwner {
    require (owner == msg.sender);
      _;
    }

  function owner () public {
    owner = msg.sender;
    }

  function withdraw(uint256 amount) onlyOwner public{
    //some code
       msg.sender.transfer(amount);
    }

  }
```

Vulnerability

Anyone can write `owner`

Step 1: Data-flow analysis to identify AC checks

<u>AC data items:</u> owner

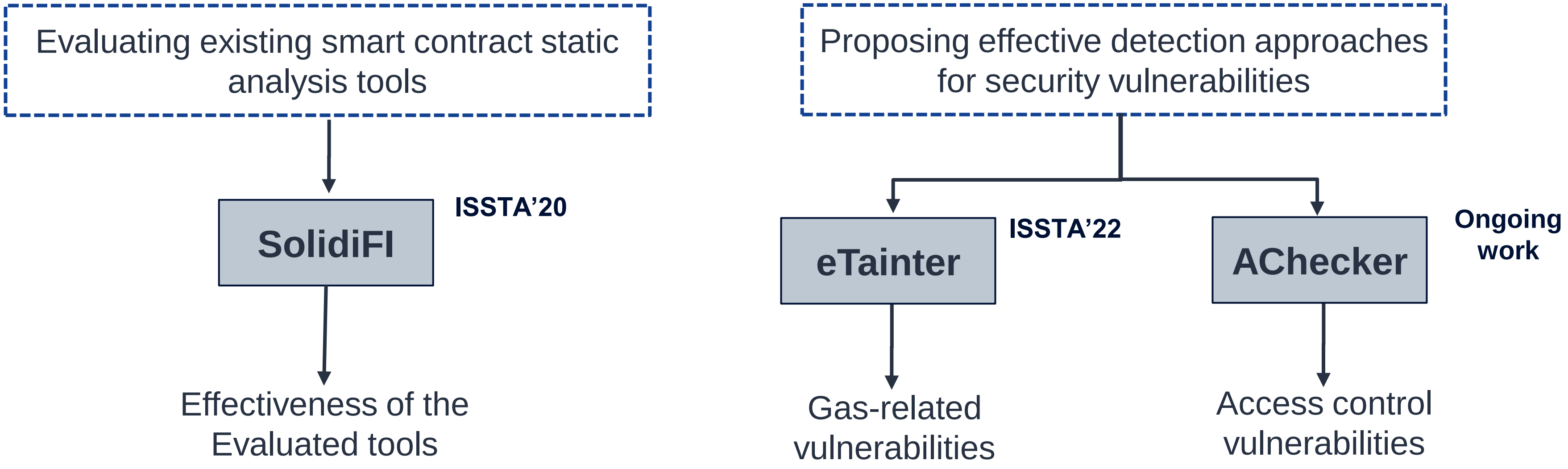
Step 2:Taint analysis to detect AC vulnerabilities

<u>Sinks:</u> owner tainted

Findings summary

- Compared AChecker with eight static analysis tools
- AChecker outperformed all tools in both recall and precision
- Average analysis time (11 seconds)
- Flagged vulnerabilities in 21 popular real-world contracts with 90% precision

Summary



Asem Ghaleb
Personal website: asemghaleb.com
Email: aghaleb@alumni.ubc.ca

Analysis Challenges

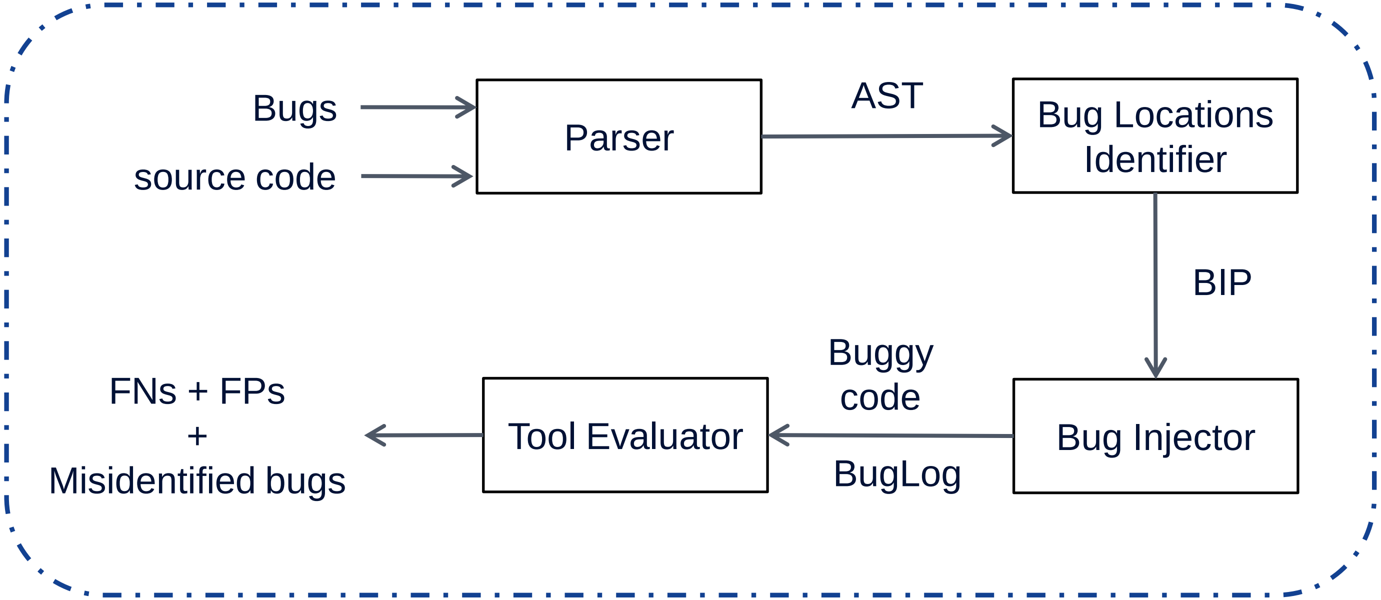
- Many vulnerabilities in smart contracts

Developers …

- Have no guarantee that they are finding vulnerabilities by static analyzers
- Have no guarantee that a found vulnerability is a true vulnerability
- May not know which tool they can trust its result more

SolidiFI

Solidity Fault Injector



SolidiFI approach: Overview

- Code snippets which lead to vulnerabilities

  1. Code snippet injection
  2. Code transformation
  3. Security weakening

- Injecting bugs the tools claim to detect
- Playing the role of developers rather attackers

SolidiFI evaluation

- 6 static analysis tools

  (*Oyente*, *Securify*, *Mythril*, *Smartcheck*, *Manticore*, *Slither*)
- 50 Smart Contracts representative of Etherscan
- Injected 9,369 distinct bugs (belong to 7 bug classes)

RQ1: False negatives of the evaluated tools?

RQ2: False positives of the evaluated tools?

RQ3: Injected bugs can be activated?

SolidiFI artifact:

https://github.com/DependableSystemsLab/SolidiFI-benchmark

Related work

MadMaX [OOPSLA, 2018]

- Uses pre-specified code patterns and rules
- Fails to detect variations in the patterns; results in high false-positives

```
for(uint i=0; i< orders[game].length; i++){
```

MadMax's rule

Fails on nested arrays

```
.decl PossibleArrayIterator(loop: Block, resVar:Variable, arrayId:Value)
// A loop, looping through an array
// Firstly, the loop has to be dynamically bound by some storage var(resVar)
// And this must be the array's size variable.
PossibleArrayIterator(loop, resVar, arrayId) :-
  StorageDynamicBound(loop, resVar),
  PossibleArraySizeVariable(resVar, arrayId).
```

Basic idea

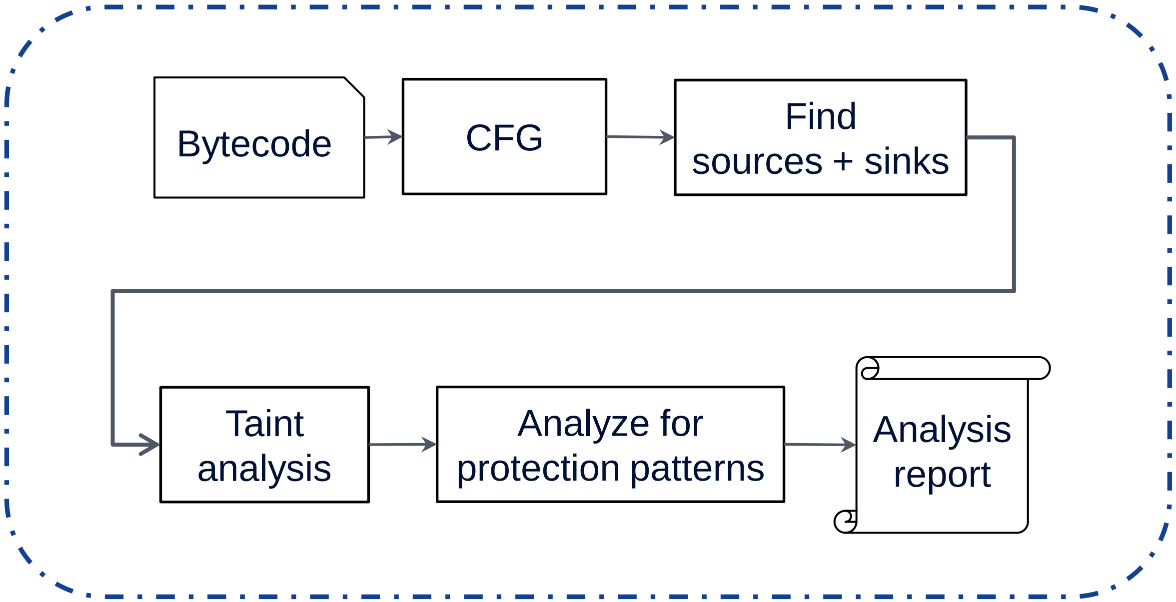
Observation:

- Gas-related vulnerabilities:
  - Caused by dependency on **user data sources manipulated by users**
  - Can be discovered by **tracking taints** without any pre-existing rules

An approach for **detecting** smart contract **gas-related vulnerabilities using static taint analysis**

eTainter

EVM Tainter



eTainter evaluation

RQ1: Effectiveness of eTainter compared to prior work (MadMax)?

RQ2: Performance of eTainter?

RQ3: Prevalence of gas-related vulnerabilities in the wild?

Dataset	Contract Num.	Used for
Annotated dataset	28	RQ1
Ethereum dataset	60,612	RQ2 & RQ3
Popular–contracts dataset	3,000	RQ3

eTainter artifact:

https://github.com/DependableSystemsLab/eTainter

Related work

Ethainter [PLDI, 2020]

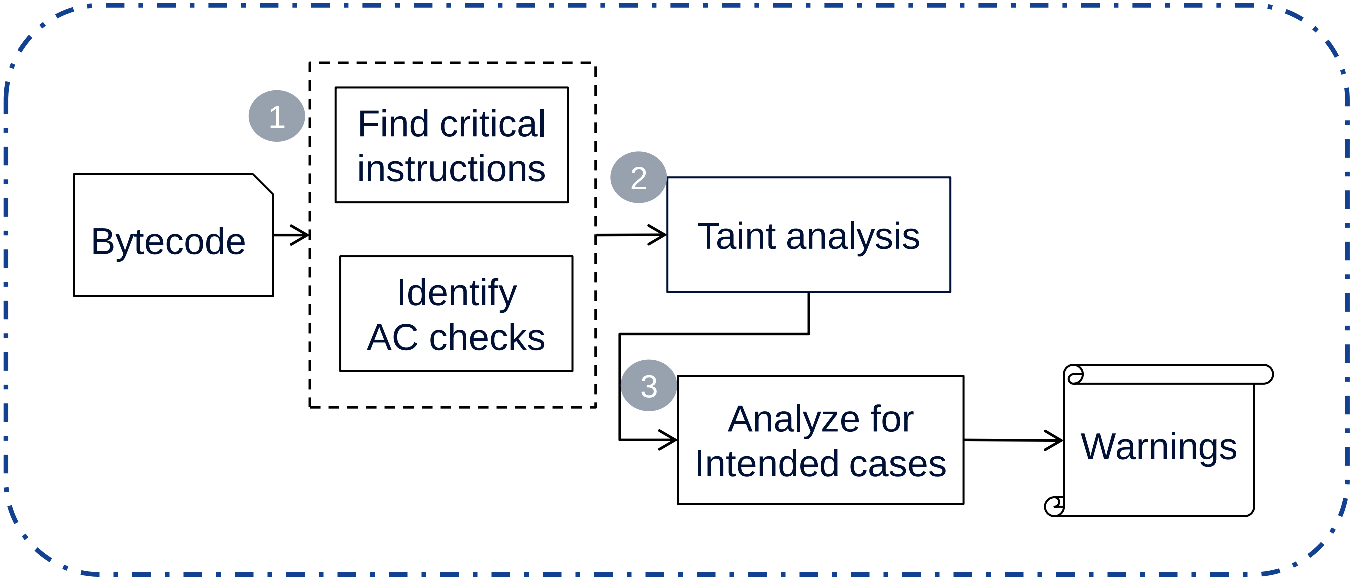
- Relies on pre-specified code patterns for access control checks
- Over-approximates access control checks

SPCon [ISSTA, 2022]

- Relies on historical transactions to extract access control rules
  - Lack of transactions for several functions
- Assumes transactions are benign and done by authorized users
  - Not guaranteed, especially for vulnerable contracts

AChecker

Access Control Checker



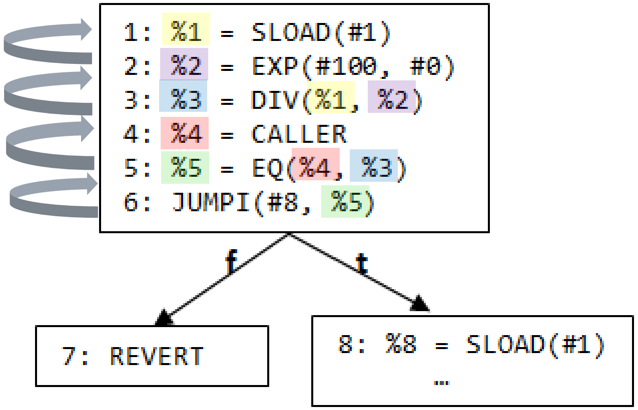
Identifying access control checks

```
contract Wallet{
  address owner = msg.sender;
  modifier onlyOwner {
    require (owner == msg.sender);
      _;
    }

  function owenr () public {
    owner =msg.sender;
    }

  function withdraw(uint256 amount) onlyOwner public{
    //some code
       msg.sender.transfer(amount);
    }

 }
```



withdraw's EVM bytecode (SSA form)

eTainter approach: Example

Intended bahaviors

```
uint256 constant howMuchToBecomeOwner = 1000 ether;

 function changeOwner (address _newOwner) payable external {
   if(msg.value >= howMuchToBecomeOwner) {
    owner.transfer(msg.value);
    owner = _newOwner;

  }
```

$$!msg.value \geq howMuchToBecomeOwner \wedge howMuchToBecomeOwner = 1000$$

AChecker evaluation

RQ1: Effectiveness of AChecker compared to prior work?

RQ2: Precise of AChecker to infer intended behaviors?

RQ3: Performance of AChecker?

RQ4: Prevalence of access control vulnerabilities?

Dataset	Contract Num.	Used for
CVE dataset	15	RQ1
SmartBugs dataset	47,518	RQ1, RQ2, RQ3 & RQ4
Popular–contracts dataset	3,000	RQ3 & RQ4