a place of mind
THE UNIVERSITY OF BRITISH COLUMBIA

Towards Effective Static Analysis Approaches for Security Vulnerabilities in Smart Contracts

Asem Ghaleb

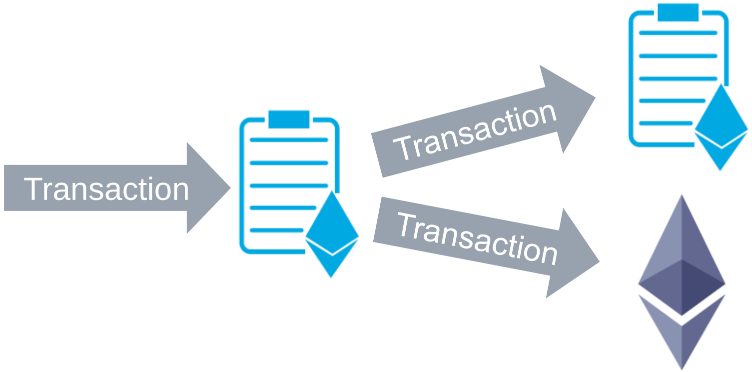
PhD Candidate @ UBC, Canada

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;

     }

    function governanceRecoverUnsupported external onlyOperator{
       //omitted code
     }

   }
```

```
initialized = true;
```

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

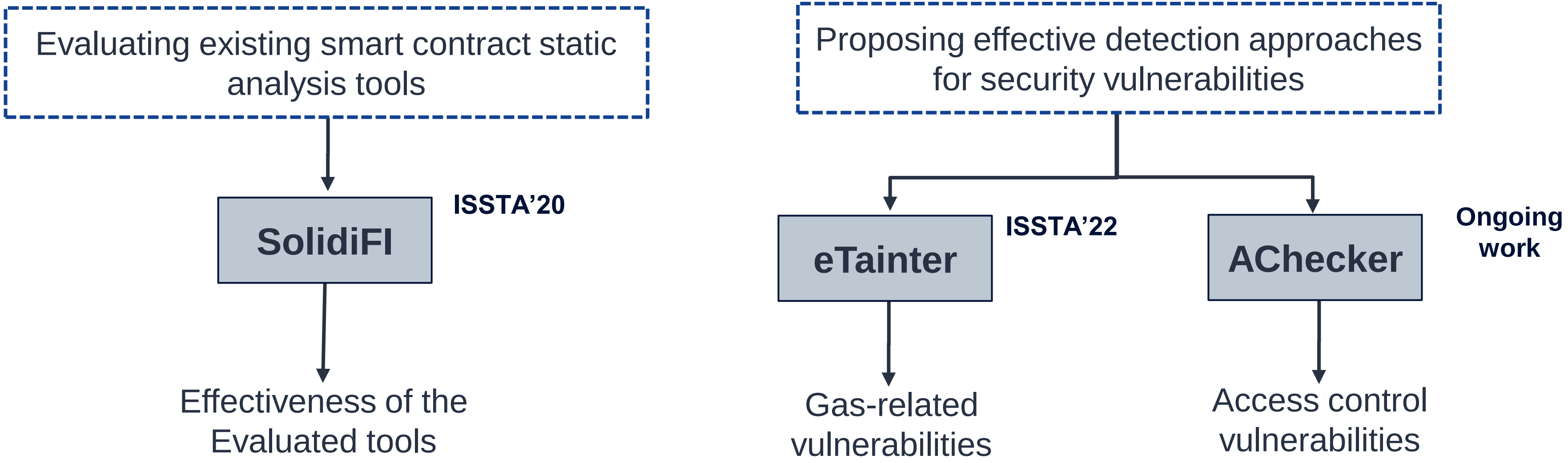
Build effective and efficient static analysis approaches for detecting security vulnerabilities in smart contracts

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

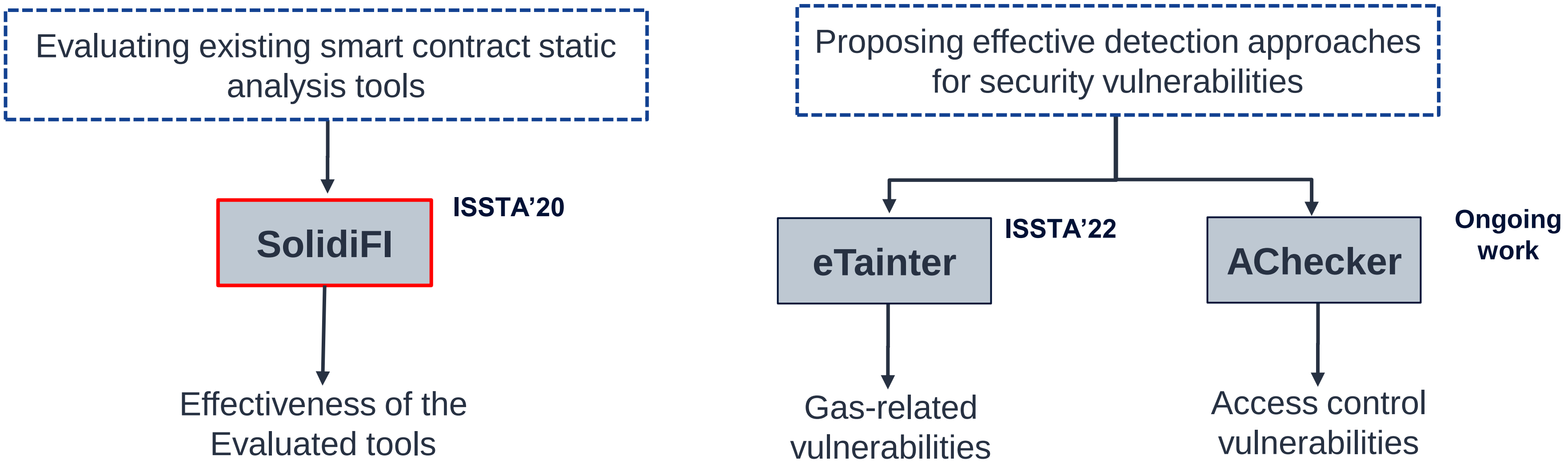
Solution insight

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

Oyente SECURIFY Mythril SmartCheck SLITHER

- 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

- **Key Idea:** inject bugs into the source code of smart contracts

Findings summary

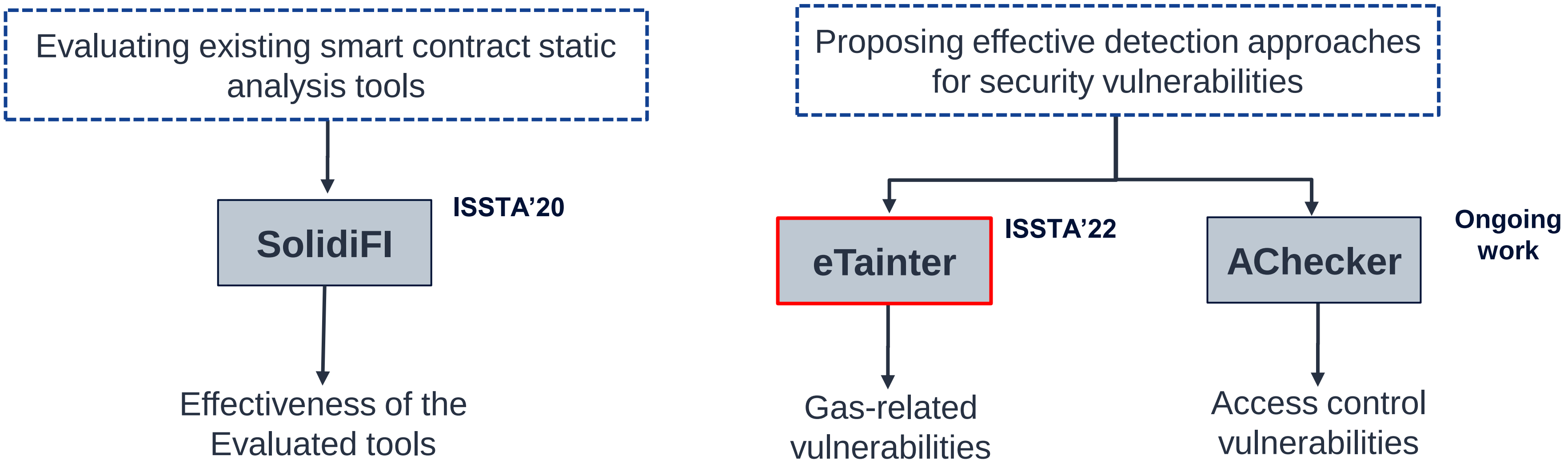
- All tools have many undetected cases
- All tools reported false positives
- Tools with low false negatives reported high false positives

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

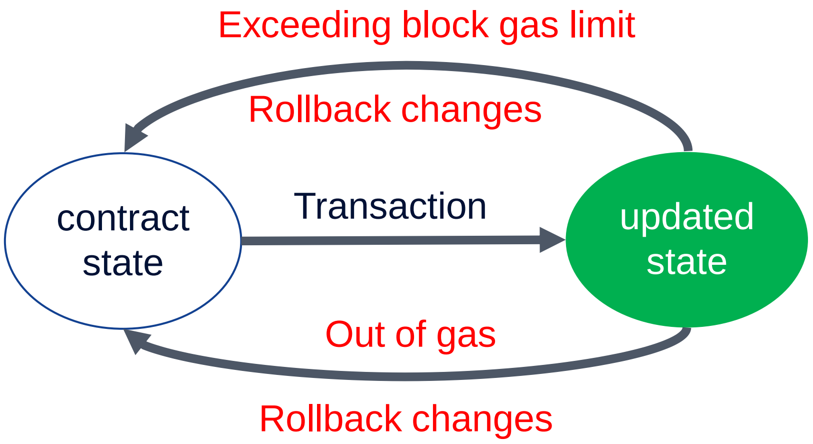
- Executing contract costs gas
- Gas cost for every EVM low-level instruction (opcode)
- Contract’s users pay the gas cost

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 tracking

Sink: i< orders[game].length

Sources:

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

Storage sink: orders[game] tainted

Taint reaches sink (loop exit condition)

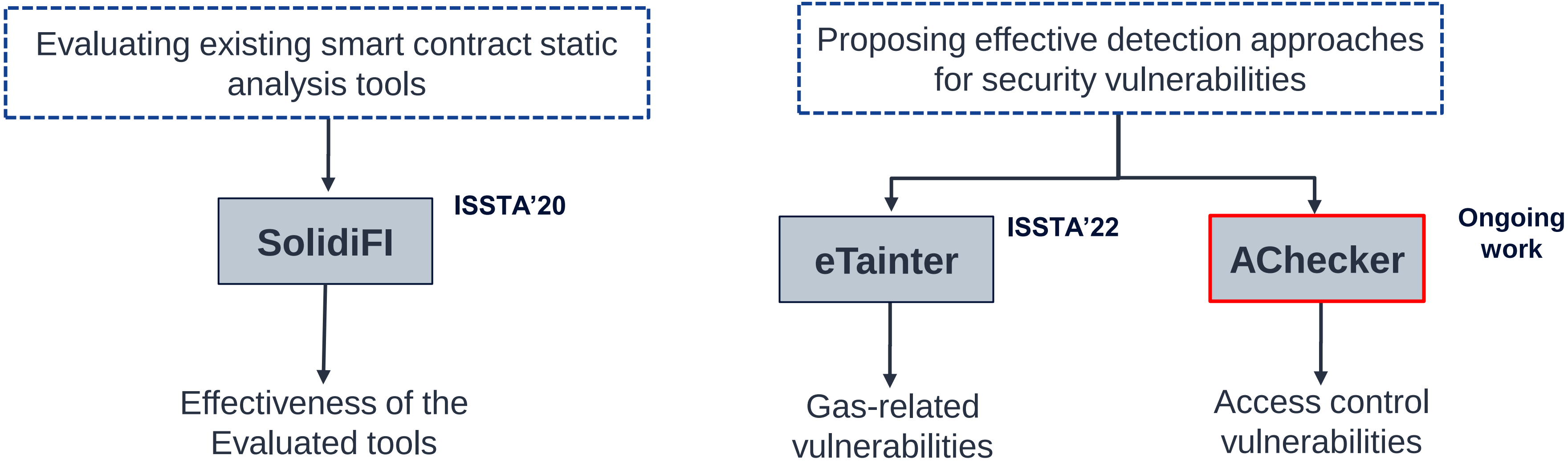
Findings summary

- 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

AC data items: owner

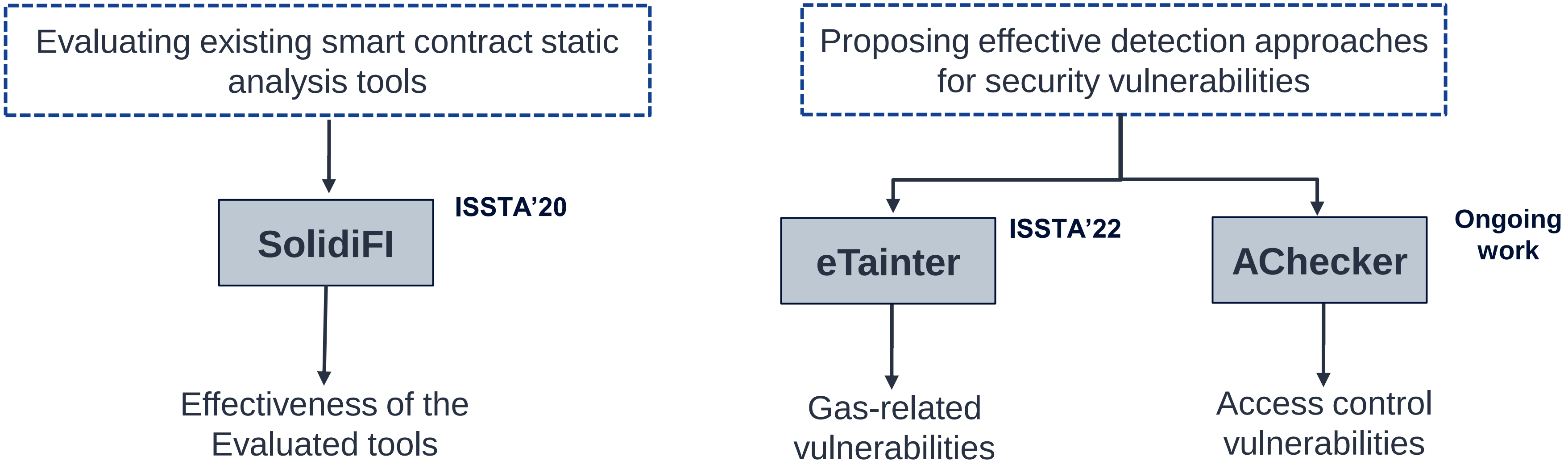
Step 2:Taint analysis to detect AC vulnerabilities

Sinks: 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