THE UNIVERSITY OF TEXAS
AT ARLINGTON

Advanced Topics in Software Engineering

CSE 6324 – Section 004

Team 10

ITERATION 1

(Written Deliverable)

Vanjari, Vaishali Sunil - 1001956614

Solanki, Siddhrajsinh Pradumansinh - 1001957988

Parimi, Taraka Naga Nikhil - 1001985955

Komireddy, Sannihith Reddy – 1001982437

I. Project Proposal

The idea is to add two detectors to Slither analysis framework. The Slither [1] is an open-source Solidity static analysis framework used to detect vulnerabilities in solidity smart contracts. Slither uses different colors to indicate the impact of severity of vulnerabilities such as red, yellow, and green for high, medium, low, and informational respectively [2]. The project consists of adding two detectors as follows:

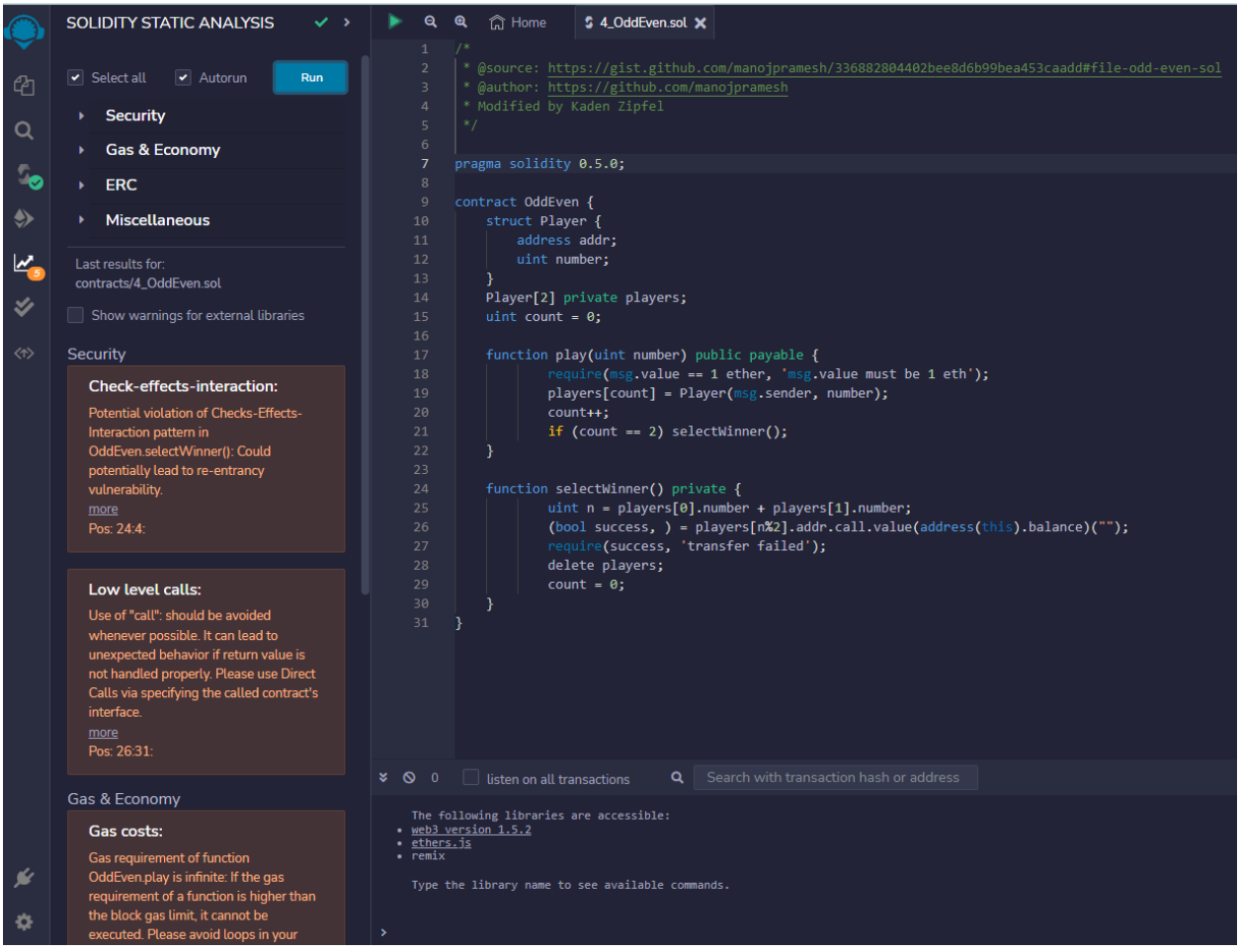
i) Detector to detect unencrypted private data on-chain [3]

ii) Compliance checker

In this iteration, the project plan is focused on adding a detector to detect unencrypted private data on-chain. There is a commonly held misunderstanding that private data variables cannot be read. Attackers can find information on the state of the contract by examining contract transactions, although the contract is not published [3]. Hence, private data must be encrypted to store on-chain or off-chain.

II. Competitor – Remix IDE

Before adding a new detector to detect the vulnerability CWE-767, we checked the existing static analysis carried out by Remix IDE but could not find the proposed vulnerability detection. Remix IDE performed the static analysis and generated an analysis on 5 security issues/ vulnerabilities with different severities [4] such as reentrancy, low-level calls, gas costs, and guard conditions.



III. Project Plan

#	Task Description	Start Date (Anticipated/Followed)	End Date (Anticipated/Followed)	Status (Completed/Incomplete)
1.	Installation a) Python v3.6+ b) solc compiler c) solc-select d) Slither	02/16/2023 (Followed)	02/22/2023 (Followed)	Completed
2.	a) Decide the detector to start working on b) Find the sample solidity contract to analyze it using Slither c) Work on reviews from Inception	02/23/2023 (Followed)	02/26/2023 (Followed)	Completed
3.	a) Load the solidity smart contract b) Set up the solidity compiler version based on the sample solidity smart contract	02/27/2023 (Followed)	03/01/2023 (Followed)	Completed
4.	a) Create a detector python file b) Add the detector to all_detectors.py using the import [5]	03/02/2023 (Followed)	03/02/2023 (Followed)	Completed
5.	a) Write a detector to detect vulnerability for CWE-767 b) Analyze the solidity contract with the new detector	03/03/2023 (Followed)	03/10/2023	In-Progress
6.	a) Gather information about the 2nd detector b) Set up the compliance standard c) Work on reviews from Iteration 1	03/11/2023	03/25/2023	Incomplete
7.	a) Define the compliance method b) Find the sample solidity contract to analyze it using Slither	03/26/2023	03/30/2023	Incomplete
8.	a) Create a detector python file b) Add the detector to all_detectors.py using the import [5]	04/01/2023	04/01/2023	Incomplete

9.	a) Write a detector for the Compliance checker b) Analyze the solidity contract with the new detector	04/02/2023	04/06/2023	Incomplete
10.	a) Generate reports b) Work on reviews from iteration 2 c)if all detectors have been implemented, search for a new detector	04/07/2023	04/24/2023	Incomplete

IV. Risk factors and Mitigation plan

#	Risk	Description	Mitigation Plan	Risk Exposure
1.	Evolving Solidity Language	Solidity is evolving each year and needs to make sure of compatibility with the detector.	Sticking with a fixed version of solidity in the detector.	Risk impact: 2 weeks Probability that risk will materialize: 92% Risk Exposure: 1 week approx.
2.	Technical issue – inexperience with Python	One of teammates has no experience working with Python, and Slither is python compatible tool.	Spend time learning python.	Risk impact: 5 weeks Probability that risk will materialize: 96% Risk Exposure: 3 weeks approx.
3.	Technical issue – inexperience with Solidity	All of teammates have no experience working with solidity smart contracts which can impact the development of the detector.	Spend time learning solidity language	Risk impact: 4 weeks Probability that risk will materialize: 90% Risk Exposure: 4 weeks approx.
4.	False-positive analysis	Slither can give false-positive analysis which needs to handle	Need to research the vulnerabilities to get an understanding of false positives.	Risk impact: 8 weeks Probability that risk will materialize: 91% Risk Exposure: 5 weeks approx.

V. Specification and Design

i) Input and Output:

Solidity smart contract will be the input that will be created with the .sol extension and can be used to do analysis. The output will be the analysis of the uploaded solidity smart contract.

Sample solidity smart contract[3] odd_even.sol as input is as follows:

```
pragma solidity 0.5.0;

contract OddEven {
    struct Player {
        address addr;
        uint number;
    }

    Player[2] private players;
    uint count = 0;

    function play(uint number) public payable {
            require(msg.value == 1 ether, 'msg.value must be 1 eth');
            players[count] = Player(msg.sender, number);
            count++;
            if (count == 2) selectWinner();
    }

    function selectWinner() private {
            uint n = players[0].number + players[1].number;
            (bool success, ) = players[n%2].addr.call.value(address(this).balance)("");
            require(success, 'transfer failed');
            delete players;
            count = 0;
    }
}
```

Analysis of the above odd_even.sol solidity contract as output is as follows:

```
vaish@vaish:~$ slither odd_even.sol

Reentrancy in OddEven.selectWinner() (odd_even.sol#19-25):
        External calls:
        - (success) = players[n % 2].addr.call.value(address(this).balance)() (odd_even.sol#21)
        State variables written after the call(s):
        - delete players (odd_even.sol#23)
        OddEven.players (odd_even.sol#9) can be used in cross function reentrancies:
        - OddEven.play(uint256) (odd_even.sol#12-17)
        - OddEven.selectWinner() (odd_even.sol#19-25)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities

Reentrancy in OddEven.selectWinner() (odd_even.sol#19-25):
        External calls:
        - (success) = players[n % 2].addr.call.value(address(this).balance)() (odd_even.sol#21)
        State variables written after the call(s):
        - count = 0 (odd_even.sol#24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Pragma version0.5.0 (odd_even.sol#1) allows old versions
solc-0.5.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in OddEven.selectWinner() (odd_even.sol#19-25):
        - (success) = players[n % 2].addr.call.value(address(this).balance)() (odd_even.sol#21)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
odd_even.sol analyzed (1 contracts with 84 detectors), 5 result(s) found
vaish@vaish:~$
```

ii) Installation:

a) Install Python v3.6+

b) Install solc compiler[6]

```
sudo add-apt-repository ppa:ethereum/ethereum
sudo apt-get update
sudo apt-get install solc
```

c) Install solc-select[7] which helps to switch between solidity compilers versions

```
pip3 install solc-select
```

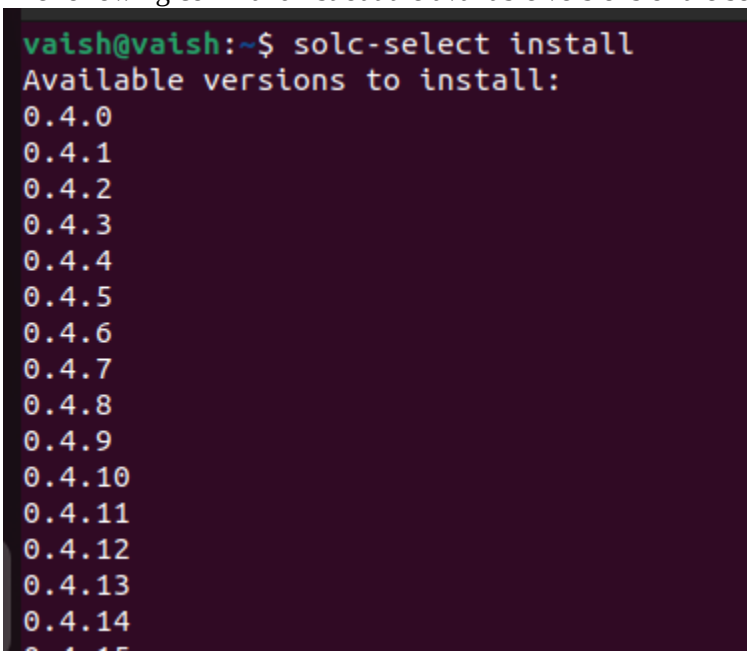
d) Install Slither[8] using pip

```
pip3 install slither-analyzer
```

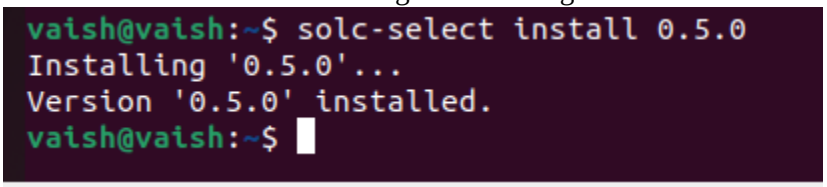
VI. Code and Tests

i) Screenshots:

a) The following command list out the available versions of the solc compiler as follows:



b) Solc version can be installed using the following command:



c) The following command can be used to use the installed solc compiler version:

```
vaish@vaish:~$ solc-select use 0.5.0
Switched global version to 0.5.0
vaish@vaish:~$
```

d) The solidity smart contract can be analyzed using the following command:

```
vaish@vaish:~$ slither odd_even.sol

Reentrancy in OddEven.selectWinner() (odd_even.sol#19-25):
        External calls:
        - (success) = players[n % 2].addr.call.value(address(this).balance)() (odd_even.sol#21)
        State variables written after the call(s):
        - delete players (odd_even.sol#23)
        OddEven.players (odd_even.sol#9) can be used in cross function reentrancies:
        - OddEven.play(uint256) (odd_even.sol#12-17)
        - OddEven.selectWinner() (odd_even.sol#19-25)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities

Reentrancy in OddEven.selectWinner() (odd_even.sol#19-25):
        External calls:
        - (success) = players[n % 2].addr.call.value(address(this).balance)() (odd_even.sol#21)
        State variables written after the call(s):
        - count = 0 (odd_even.sol#24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Pragma version0.5.0 (odd_even.sol#1) allows old versions
solc-0.5.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in OddEven.selectWinner() (odd_even.sol#19-25):
        - (success) = players[n % 2].addr.call.value(address(this).balance)() (odd_even.sol#21)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
odd_even.sol analyzed (1 contracts with 84 detectors), 5 result(s) found
vaish@vaish:~$
```

e) To add the detector, a python file needs to be created, and write detector skeleton[2] in that file. Then, add that file in all_detectors.py file which can be traced in the .local folder in Ubuntu. Import the detector as shown below:

```
  GNU nano 6.2
# pylint: disable=unused-import,relative-beyond-top-level
from .examples.backdoor import Backdoor
#from .examples.CWE767 import CWE767Detector
from .variables.uninitialized_state_variables import UninitializedStateVarsDetection
from .variables.uninitialized_storage_variables import UninitializedStorageVars
from .variables.uninitialized_local_variables import UninitializedLocalVars
from .variables.var_read_using_this import VarReadUsingThis
from .attributes.constant_pragma import ConstantPragma
from .attributes.incorrect_solc import IncorrectSolc
from .attributes.locked_ether import LockedEther
from .functions.arbitrary_send_eth import ArbitrarySendEth
from .erc.erc20.arbitrary_send_erc20_no_permit import ArbitrarySendErc20NoPermit
from .erc.erc20.arbitrary_send_erc20_permit import ArbitrarySendErc20Permit
from .functions.suicidal import Suicidal

# from .functions.complex_function import ComplexFunction
from .reentrancy.reentrancy_benign import ReentrancyBenign
from .reentrancy.reentrancy_read_before_write import ReentrancyReadBeforeWritten
from .reentrancy.reentrancy_eth import ReentrancyEth
from .reentrancy.reentrancy_no_gas import ReentrancyNoGas
from .reentrancy.reentrancy_events import ReentrancyEvent
from .variables.unused_state_variables import UnusedStateVars
```

Path of all_detectors.py: /home/vaish/.local/lib/python3.10/site-packages/slither/detectors/all_detectors.py

f) Complete the detector. (The following detector code is not complete as throws an abstractDetector error)[8]

```python
from slither.core.solidity_types import Type
from slither.detectors.abstract_detector import AbstractDetector, DetectorClassification
from slither.core.variables import variable
from slither.core.declarations import Function

class CWE767Detector():

    ARGUMENT = "CWE767"  # slither will launch the detector with slither.py --mydetector
    IMPACT = DetectorClassification.HIGH
    CONFIDENCE = DetectorClassification.HIGH

    def __init__(self, contract, function):
        super().__init__(contract, function)

    def detect(self):
        # Check if the function is public and accesses a private state variable
        if self.function.visibility == "public":
            for variable in self.function.state_variables_read:
                if isinstance(variable.variable, variable) and variable.variable.visibility == "private":
                    self._issues.append("CWE-767: Access to critical private variable '{}' via public function '{}'".format(variable.variable.name, self.function.name))
            for variable in self.function.state_variables_written:
                if isinstance(variable.variable, variable) and variable.variable.visibility == "private":
                    self._issues.append("CWE-767: Access to critical private variable '{}' via public function '{}'".format(variable.variable.name, self.function.name))
```

g) Analyze the smart contract again to get the expected result.

ii) Test Case:

#	Test Case	Expected Output
1.	Running the solidity smart contract with Slither without adding the detector	Analysis of solidity smart contract without CWE-767 vulnerability
2.	Running the solidity smart contract with Slither after adding the detector	Analysis of solidity smart contract with CWE-767 vulnerability

iii) GitHub link: https://github.com/vaishalivanjari/CSE6324ASE

VII. Customers and Users

#	Customer	Feedback/Suggestions
1.	Shubham Rathi (Solidity Beginner)	a) Good Concept b) Need to make user-friendly UI
2.	Ashwini Shenvi (Solidity Enthusiast)	a) Easy to understand. b) Interested in knowing how vulnerability will be detected by the detector
3.	Darshan Patil (CSE 6324-Team 2)	a) Need to test a few more sample solidity contracts for the newly added detector

VIII. References

1. Slither, the Solidity source analyzer
2. Adding-a-new-detector-to Slither-skeleton
3. SWC-136-Unencrypted-private-data-on-chain
4. RemixIDE-solidity-static-analysis
5. Adding-a-new-detector-to-Slither-in-all_detectors.py
6. installing-solidity-solc-compiler
7. install-solc-select-slither
8. install-slither
9. Slither-Usage#detector-selection