THE UNIVERSITY OF TEXAS AT ARLINGTON

Advanced Topics in Software Engineering

CSE 6324 – Section 004

Team 10

FINAL ITERATION

(Written Deliverable)

Vanjari, Vaishali Sunil - 1001956614

Solanki, Siddhrajsinh Pradumansinh - 1001957988

Parimi, Taraka Naga Nikhil - 1001985955

Komireddy, Sannihith Reddy – 1001982437

I. Project Proposal

The Slither analysis framework will be expanded by the addition of two detectors. The open-source static analysis framework for Solidity called Slither[1] can be used to identify issues within smart contracts. Slither uses the colors red, yellow, and green to represent high, medium, low, and informative vulnerabilities, respectively, in terms of impact and severity[2]. The project consists of adding two detectors as follows:

i) Detector to detect unencrypted private data on-chain [3]
ii) Detector to detect incorrect constructor name [11]

The main goal of the project plan is to propose results after adding the above two detectors. Contrary to general opinion, private data variables can be read. Attackers can discover the status of the contract by examining contract transactions even when the contract is not made public. It must be encrypted to be stored privately, either on-chain or off-chain[3]. As an alternative, private data can be altered in a pure private function and called from a public function. These mitigations are tested in the proposed detector for CWE-767. The point to be noted is that Slither does not detect this vulnerability before adding the CWE-767 detector.

In the final iteration, the detector to detect vulnerability CWE-767 is completed where the detector is checking if private data is encrypted or not and if it is being modified in a public function. The project checks every assignment to private data and validates if it is encrypted or not. If data is encrypted, then it can be modified in a public function. Data encryption is checked using regular expressions. As Slither has very few inbuilt cryptographic functions[13], the regEx expression uses external cryptographic functions as well. To check if private data is being modified or not in a public function, the project proposed the solution of checking the visibility of the function and if it is public then accessing all the state variables written in that public function. The point to note is local variables cannot be private in solidity language.

The idea of adding another detector to find incorrect constructor names[11] was proposed in the previous iteration. Constructors are unique functions that are only used once when creating contracts. They frequently carry out crucial, exclusive tasks including identifying the contract's owner. The only way to define a constructor in Solidity before version 0.4.24 was to write a function with the same name as the contract class that contained it. If a function's name differs slightly from the contract name, it becomes a regular, callable function and is no longer intended to serve as a constructor. When smart contract code is utilized under a new name without changing the name of the constructor function, this practice might occasionally cause security problems.

In the final iteration, the detector to detect incorrect constructor names related to CWE-665 is completed where the detector differentiates in a regular callback function and constructor using the special characteristics of the constructor such as the constructor is used to initialize state variables, the visibility of the constructor should be public, and constructor does not have a return type. The project also proposed for similarity matrix to check the similarity[14] between the constructor declared using the function and contract name and set the threshold above 0.5. The proposed solution faced issues when there is a setter function that does not have any return type, visibility is public which are almost similar characteristics as the constructor and used to assign values to state variables, and the contract has an incorrect/previous constructor name that does not meet the threshold value 0.5 for the similarity between the contract name and the constructor’s name.

II. Competitor detectors in Slither

i) Detector to detect CWE-767

There is no competitor detector in Slither to detect CWE-767 vulnerability.

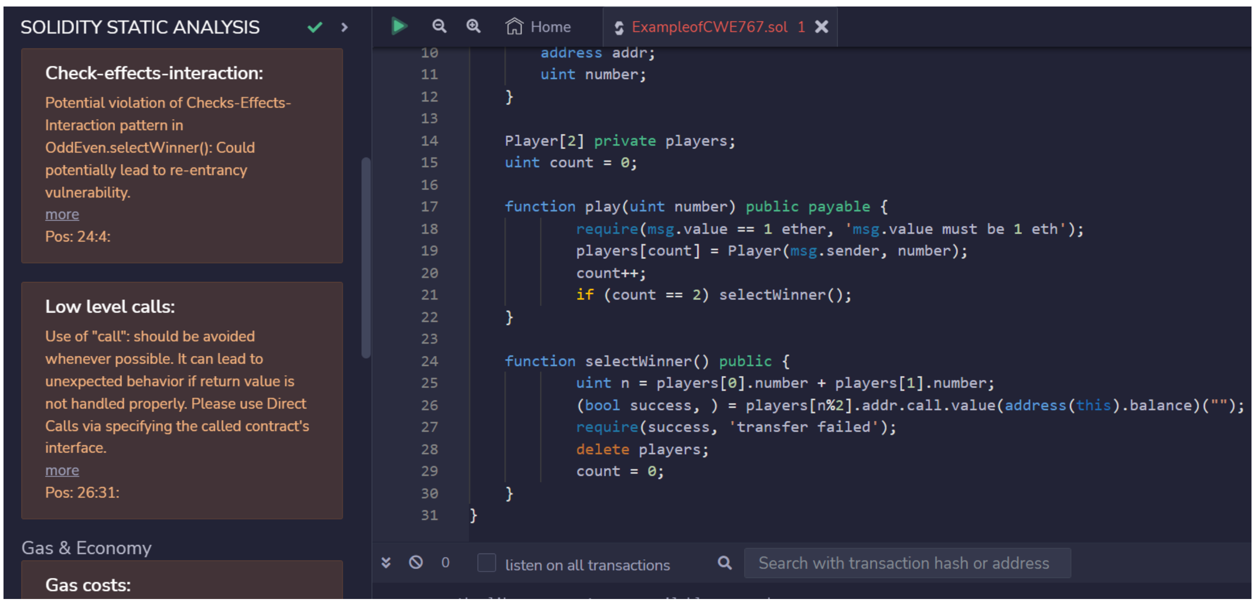
ii) Detector to detect CWE-665

There is no competitor detector in Slither to detect CWE-665 vulnerability. However, there is a naming-convention detector that can be used to follow the naming convention, but it does not provide static analysis if anyone follows the naming convention and uses an incorrect constructor name. For example, the contract name is Missing, and the constructor declared using the function is Misingg. Over here naming-convention detector will provide analysis as the first letter of the function name should be lowercase but will not provide that the constructor's name is incorrect.

III. Competitor – Remix IDE

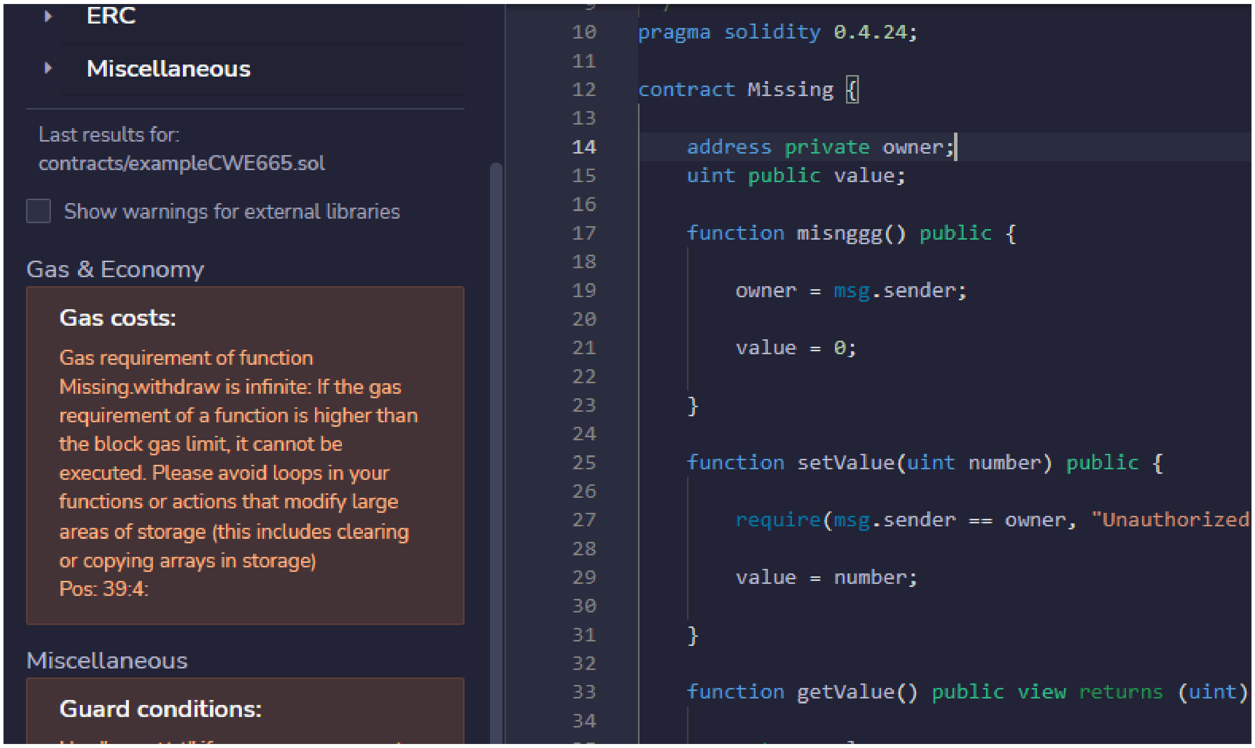
i) Detector to detect CWE-767:

Before adding a new detector in Slither to find the vulnerability CWE-767, we checked the static analysis currently being performed by Remix IDE but were unable to find the indicated vulnerability analysis on several smart contracts. Remix IDE performed static analysis, which resulted in an analysis of several security issues and vulnerabilities of various severity, including reentrancy, low-level calls, gas pricing, and guard conditions.



ii) Detector to detect the incorrect constructor's name:

We investigated the static analysis presently being performed by Remix IDE before adding a second detector to find the erroneous constructor's name, but we were unable to locate the required vulnerability analysis. Static analysis was performed by Remix IDE, leading to an investigation of security flaws or issues of varying importance, such as guard conditions and gas costs.



IV. <u>Project Plan</u>

#	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 (Incomplete)	Completed
6.	a) Gather information about the 2nd detector b) Find the sample solidity contract to analyze it using Slither. c) Work on reviews from Iteration 1	03/11/2023 (Followed)	03/25/2023 (Followed)	Completed
7.	a) Load the solidity smart contract b) Set up the solidity compiler version based on the sample solidity smart contract	03/26/2023 (Followed)	03/30/2023 (Followed)	Completed
8.	a) Create a detector Python file b)Add the detector to all_detectors.py using the import [5]	04/01/2023 (Followed)	04/05/2023 (Followed)	Completed
9.	a)Start development of 2nd detector. b)Analyze the solidity contract with the new detector. c) Work on reviews from iteration 2 d) Mitigate the vulnerabilities detected such as CWE767, and CWE665 in solidity smart contracts using best security practices.	04/06/2023 (Followed)	04/24/2023 (Followed)	Completed

V. Risk factors and Mitigation plan

#	Risk	Description	Mitigation Plan	Risk Exposure
1.	The Evolution of Solidity Language	Every year, the solidity language changes, therefore detector must work with it.	Keeping eye on the changing solidity pragma version and maintaining the detector accordingly.	Risk impact: 2 weeks The probability that risk will materialize: 30% Risk Exposure: 3 weeks approx.
2.	Technical issue – problem while finding cryptographic function for CWE-767 detector	To develop a CWE767 detector, it is important to verify whether private data is encrypted.	Need to update the cryptographic function as Solidity evolves.	Risk impact: 3 weeks The probability that risk will materialize: 90% Risk Exposure: 4 weeks approx.

3.	Technical issue – inexperience with Solidity	The team's uneven lack of basic language understanding could delay the detector's development.	While we move ahead, study solidity language.	Risk impact: 7 weeks The probability that risk will materialize: 94% Risk Exposure: 8 weeks approx.
4.	Technical issue – unable to differentiate between setter function and constructor in CWE-665 detector	The setter function has almost similar characteristics as the constructor and is unable to handle the case where the setter and incorrect/previous constructor’s name not meeting threshold 0.5 for the similarity name matrix.	Need to find a solution for this H-case.	Risk impact: 1 week The probability that risk will materialize: 50% Risk Exposure: 1 week approx.

VI. <u>Specification and Design</u>

i) Input and Output for the detector to detect CWE-767:

Solidity smart contracts, which can be used as analysis input, are created using .sol extension. The outcome will be an analysis of the uploaded solidity smart contract. Odd_even.sol, a sample Solidity smart contract[3], is used as the detector's input. Line numbers 18, and 26, where a public function modifies the unencrypted private data, contains the vulnerable code. Line number 18 modifies the private data players by assigning a number in the public function payable() and line number 26 deletes the private data players in the public function selectWinner(), and both cases are vulnerable to attack related to CWE-767.

```
6 pragma solidity 0.5.0;
7 contract OddEven {
8     struct Player {
9         address addr;
10        uint number;
11    }
12
13    Player[2] private players;
14    uint count = 0;
15
16    function play(uint number) public payable {
17            require(msg.value == 1 ether, 'msg.value must be 1 eth');
18            players[count] = Player(msg.sender, number);
19            count++;
20            if (count == 2) selectWinner();
21    }
22    function selectWinner() public {
23            uint n = players[0].number + players[1].number;
24            (bool success, ) = players[n%2].addr.call.value(address(this).balance)("");
25            require(success, 'transfer failed');
26            delete players;
27            count = 0;
28    }
29 }
```

Analysis of the above odd_even.sol solidity contract without adding a detector to detect CWE-767 as output. As seen below, Slither does not detect that private data has been modified in the public visibility function.

```
vaish@vaish:~$ slither odd_even.sol
INFO:Detectors:
Reentrancy in OddEven.selectWinner() (odd_even.sol#22-28):
        External calls:
        - (success) = players[n % 2].addr.call.value(address(this).balance)() (odd_even.sol#24)
        State variables written after the call(s):
        - delete players (odd_even.sol#26)
        OddEven.players (odd_even.sol#13) can be used in cross function reentrancies:
        - OddEven.play(uint256) (odd_even.sol#16-21)
        - OddEven.selectWinner() (odd_even.sol#22-28)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
Reentrancy in OddEven.selectWinner() (odd_even.sol#22-28):
        External calls:
        - (success) = players[n % 2].addr.call.value(address(this).balance)() (odd_even.sol#24)
        State variables written after the call(s):
        - count = 0 (odd_even.sol#27)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Pragma version0.5.0 (odd_even.sol#6) 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
INFO:Detectors:
Low level call in OddEven.selectWinner() (odd_even.sol#22-28):
        - (success) = players[n % 2].addr.call.value(address(this).balance)() (odd_even.sol#24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Slither:odd_even.sol analyzed (1 contracts with 85 detectors), 5 result(s) found
vaish@vaish:~$
```

The following output after adding the CWE-767 detector can be considered as the final output, and it displayed vulnerabilities on line numbers 18, and 26 are detected as follows:

```
vaish@vaish:~$ slither odd_even.sol
INFO:Detectors:
CWE-767:Access to Critical Private Variable in public method in OddEven.play(uint256) (odd_even.sol#16-21)
        -Illegal modification:
        -players[count] = Player(msg.sender,number)(odd_even.sol#18)
CWE-767:Access to Critical Private Variable in public method in OddEven.selectWinner() (odd_even.sol#22-28)
        -Illegal modification:
        -delete players(odd_even.sol#26)
Reference: https://cwe.mitre.org/data/definitions/767.html
INFO:Detectors:
Reentrancy in OddEven.selectWinner() (odd_even.sol#22-28):
        External calls:
        - (success) = players[n % 2].addr.call.value(address(this).balance)() (odd_even.sol#24)
        State variables written after the call(s):
        - delete players (odd_even.sol#26)
        OddEven.players (odd_even.sol#13) can be used in cross function reentrancies:
        - OddEven.play(uint256) (odd_even.sol#16-21)
        - OddEven.selectWinner() (odd_even.sol#22-28)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
Reentrancy in OddEven.selectWinner() (odd_even.sol#22-28):
        External calls:
        - (success) = players[n % 2].addr.call.value(address(this).balance)() (odd_even.sol#24)
        State variables written after the call(s):
        - count = 0 (odd_even.sol#27)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Pragma version0.5.0 (odd_even.sol#6) allows old versions
solc-0.5.0 is not recommended for deployment
```

The detector determines whether encrypted private data is being updated in a public function. The project examines each assignment involving private data and determines whether it has been encrypted. Data can be changed in a public function even if it is encrypted. Regular expressions are used to verify data encryption. The regEx statement also employs external cryptographic functions because Slither has very few built-in cryptographic methods[13]. The project provided a method of assessing the function's visibility and, if it is public, then accessing all the state variables written in that public function to determine whether private data is being modified in a public function.

The mitigation for CWE-767 is using encryption on line #24 and modifying data in a pure private function on line #28 is applied as follows:

```
19    constructor() public{
20        owner = msg.sender;
21    }
22    function play(bytes32 number) public payable {
23        require(msg.value == 1 ether, 'msg.value must be 1 eth');
24        players[count] = Player(keccak256(abi.encodePacked(msg.sender)), keccak256(abi.encodePacked(number)));
25        count++;
26        if (count == 2) selectWinner();
27    }
28    function selectWinner() private {
29        require(msg.sender == owner, 'unauthorized');
30        msg.sender.transfer(address(this).balance);
31        delete players;
32        count = 0;
33    }
34    function setKey(bytes32 _key) public {
35        key = keccak256(abi.encodePacked(_key));
36    }
37    function decryptPlayer(uint index) public view returns (address, uint) {
38        bytes32 addr = players[index].encryptedAddr;
39        bytes32 number = players[index].encryptedNumber;
40        require(keccak256(abi.encodePacked(msg.sender, key)) == addr, 'unauthorized');
41        return (address(bytes20(addr)), uint(number));
42    }
```

ii) Input and Output for the detector to detect incorrect constructor names:

Solidity smart contracts, which can be used as analysis input, are created using the .sol extension. The outcome will be an analysis of the provided solidity smart contract. Sample solidity smart contract[11] incorrect_constructor.sol as input for the mentioned detector. The vulnerable code can be found on line no. 18 where the constructor's name is incorrect, and the attacker can take advantage of that.

```
7 pragma solidity 0.4.24;
8
9 contract Missing {
10     address private owner;
11     uint public value;
12
13     //constructor() public{
14     //    value = 0;
15     //}
16
17     //constructor
18     function misnggg() public {
19         owner = msg.sender;
20         value = 0;
21     }
22
23     function setValue(uint number) public {
24         require(msg.sender == owner, "Unauthorized");
25         value = number;
26     }
27
28     function getValue() public view returns (uint) {
29         return value;
30     }
31
```

Analysis of the above incorrect_constructor.sol solidity contract without adding detector to detect incorrect constructor name as output. As seen below, Slither does not detect the incorrect constructor's name from the smart contract uploaded.

```
vaish@vaish:~$ slither incorrect_constructor.sol
INFO:Detectors:
Missing.setValue(uint256) (incorrect_constructor.sol#23-26) should emit an event for:
        - value = number (incorrect_constructor.sol#25)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-ari
thmetic
INFO:Detectors:
Pragma version0.4.24 (incorrect_constructor.sol#7) allows old versions
solc-0.4.24 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions
-of-solidity
INFO:Slither:incorrect_constructor.sol analyzed (1 contracts with 85 detectors), 3 result(s
) found
```

The final output, which may be considered to include the CWE-665 detector, showed vulnerabilities on line number 18 that were discovered as follows:

```
INFO:Detectors:
Missing.setValue(uint256) (incorrect_constructor.sol#23-26) should emit an event for:
        - value = number (incorrect_constructor.sol#25)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-ari
thmetic
INFO:Detectors:
CWE-665:Improper Intialization of constructor in Missing.misnggg() (incorrect_constructor.s
ol#18-21)
        -Improper initizalition of constructor(incorrect_constructor.sol#18)
Reference: https://cwe.mitre.org/data/definitions/665.html
INFO:Detectors:
Pragma version0.4.24 (incorrect_constructor.sol#7) allows old versions
solc-0.4.24 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions
-of-solidity
INFO:Slither:incorrect_constructor.sol analyzed (1 contracts with 87 detectors), 5 result(s
) found
```

The completed constructor name error detector for CWE-665 distinguishes between a regular callback function and a constructor based on the constructor's unique properties, such as the fact that it is used to initialize state variables, that its visibility should be public, and that it lacks a return type. The project also suggested using a similarity matrix to determine whether the constructor defined using a function and contract name are similar[14] and to establish a threshold higher than 0.5.

The mitigation for CWE-665 is using a newer pragma version on line #7 and declaring the constructor using the constructor() keyword on line #13 applied as below:

```
 7 pragma solidity 0.8.24;
 8
 9 contract Missing {
10     address private owner;
11     uint public value;
12
13     constructor() public{
14         value = 0;
15     }
16
17     function setValue(uint number) public {
18         require(msg.sender == owner, "Unauthorized");
19         value = number;
20     }
21
22     function getValue() public view returns (uint) {
23         return value;
24     }
25
26     function withdraw() public {
27         require(msg.sender == owner, "Unauthorized");
28         msg.sender.transfer(address(this).balance);
29     }
30 }
```

The H-case which is not working is when the setter function with almost similar characteristics as the constructor such as no return type, visibility is public, and assigning values to state variables. Consider the following example where mycontract on line #13 is the constructor's name and Missing is the contract name. The mycontract and Missing are unable to provide a similarity matrix ratio above the threshold of 0.5. In addition to that setValue() is a setter function that has almost similar characteristics to the constructor.

```
 7 pragma solidity 0.4.24;
 8
 9 contract Missing {
10     address private owner;
11     uint public value;
12
13     function mycontract() public {
14         owner = msg.sender;
15         value = 0;
16     }
17
18     function setValue(uint number) public {
19         require(msg.sender == owner, "Unauthorized");
20         value = number;
21     }
22
23     function getValue() public view returns (uint) {
24         return value;
25     }
26
27     function withdraw() public {
28         require(msg.sender == owner, "Unauthorized");
29         msg.sender.transfer(address(this).balance);
30     }
31 }
```

Analysis of the above solidity smart contract is as below:

```
vaish@vaish:~$ slither incorrect_constructor.sol
INFO:Detectors:
CWE-767:Access to Critical Private Variable in public method in Missing.mycontract() (incorrect_constructor.sol#13-16)
        -Illegal modification:
        -owner = msg.sender(incorrect_constructor.sol#14)
Reference: https://cwe.mitre.org/data/definitions/767.html
INFO:Detectors:
Missing.setValue(uint256) (incorrect_constructor.sol#18-21) should emit an event for:
        - value = number (incorrect_constructor.sol#20)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
Pragma version0.4.24 (incorrect_constructor.sol#7) allows old versions
solc-0.4.24 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Slither:incorrect_constructor.sol analyzed (1 contracts with 87 detectors), 4 result(s) found
vaish@vaish:~$
```

iii) 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
```

VII. <u>Code and Tests</u>

i) Screenshots:

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

```
vaish@vaish:~$ solc-select install
Available versions to install:
0.4.0
0.4.1
```

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

```
vaish@vaish:~$ solc-select install 0.5.0
Installing '0.5.0'...
Version '0.5.0' installed.
vaish@vaish:~$
```

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
```

e) Create a Python file and add a detector skeleton[2] to it to add the detector. After that, append that file to the all_detectors.py file, which can be found 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 to detect CWE-767 is detecting CWE767)[8]:

```
from slither.detectors.abstract_detector import AbstractDetector, DetectorClassification
from slither.core.solidity_types.elementary_type import ElementaryType
import re
from Crypto.Cipher import AES

class CWE767detector(AbstractDetector):
    ARGUMENT = "CWE767Detector"  # slither will launch the detector with slither.py --mydetector
    HELP = "CWE767:Critical private access in public method"
    IMPACT = DetectorClassification.HIGH
    CONFIDENCE = DetectorClassification.HIGH
    WIKI = "https://cwe.mitre.org/data/definitions/767.html"
    WIKI_TITLE = "CWE767 Vulnerability example"
    WIKI_DESCRIPTION = "Detector example"
    WIKI_EXPLOIT_SCENARIO = "Adding a detector to find unencrypted private data on-chain is the main goal of the project plan. Often held misconception: Private data variables cannot be read. Although the contract is not disclosed, attackers can learn about its status by looking at contract transactions."
    WIKI_RECOMMENDATION = "To store private data on-chain or off-chain, it must be encrypted. Alternatively, private data can be modified in pure private function and that private function can be called in public function."

    def _detect(self):
        results = []
        encryption_regex = re.compile(r'AES|RSA|DES|3DES|Blowfish|Twofish|MD5|sha256|sha3|bcrypt|scrypt|keccak256|ripemd160|SHA3|ecrecover|secp256k1')
        for contract in self.slither.contracts:
            for function in contract.functions:
                for node in function.nodes:
```

g) Analyze the smart contract again with the newly added detector to get the expected result.

```
vaish@vaish:~$ slither odd_even.sol
INFO:Detectors:
CWE-767:Access to Critical Private Variable in public method in OddEven.play(uint256) (odd_even.sol#16-21)
        -Illegal modification:
        -players[count] = Player(msg.sender,number)(odd_even.sol#18)
CWE-767:Access to Critical Private Variable in public method in OddEven.selectWinner() (odd_even.sol#22-28)
        -Illegal modification:
        -delete players(odd_even.sol#26)
Reference: https://cwe.mitre.org/data/definitions/767.html
INFO:Detectors:
Reentrancy in OddEven.selectWinner() (odd_even.sol#22-28):
        External calls:
        - (success) = players[n % 2].addr.call.value(address(this).balance)() (odd_even.sol#24)
        State variables written after the call(s):
        - delete players (odd_even.sol#26)
        OddEven.players (odd_even.sol#13) can be used in cross function reentrancies:
        - OddEven.play(uint256) (odd_even.sol#16-21)
        - OddEven.selectWinner() (odd_even.sol#22-28)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
Reentrancy in OddEven.selectWinner() (odd_even.sol#22-28):
        External calls:
        - (success) = players[n % 2].addr.call.value(address(this).balance)() (odd_even.sol#24)
        State variables written after the call(s):
        - count = 0 (odd_even.sol#27)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Pragma version0.5.0 (odd_even.sol#6) allows old versions
```

h) Apply best security practices to mitigate the detected vulnerability such as the following mitigation for CWE-767.

```
    address internal owner;

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

    function play(bytes32 number) public payable {
        require(msg.value == 1 ether, 'msg.value must be 1 eth');
        players[count] = Player(keccak256(abi.encodePacked(msg.sender)), keccak256(abi.encodePacked(number)));
        count++;
        if (count == 2) selectWinner();
    }

    function selectWinner() private {
        require(msg.sender == owner, 'unauthorized');
        msg.sender.transfer(address(this).balance);
        delete players;
        count = 0;
    }

    function setKey(bytes32 _key) public {
        key = keccak256(abi.encodePacked(_key));
    }

    function decryptPlayer(uint index) public view returns (address, uint) {
        bytes32 addr = players[index].encryptedAddr;
```

ii) Test Case:

#	Test Case	Expected Output
1.	Running the solidity smart contract without adding the detector in Slither	Analysis of solidity smart contract without CWE-767 vulnerability
2.	Running the solidity smart contract after adding the detector in Slither	Analysis of solidity smart contract with CWE-767 vulnerability
3.	Running the solidity smart contract without adding the detector in Slither	Analysis of solidity smart contract without detecting incorrect constructor name.
4.	Running the solidity smart contract after adding the detector in Slither	Analysis of solidity smart contract with detecting incorrect constructor name.

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

VIII. Customers and Users

#	Customer	Feedback/Suggestions
1.	Shubham Rathi (Solidity Beginner)	Thank the team for being proactive in providing mitigation for the discovered issue.
2.	Ashwini Shenvi (Solidity Enthusiast)	The idea is easy for beginners to understand.
3.	Darshan Patil (CSE 6324-Team 2)	Looking forward to the professor's evaluation of the suggested CWE-767 solution.

IX. 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
10. a-sample-contract-visibility-and-getters
11. Incorrect-constructor-name
12. Blog-sigmaprime-io/solidity-security#constructors
13. slither-mathematical-and-cryptographic-functions
14. find-the-similarity-metric-between-two-strings