THE UNIVERSITY OF TEXAS
AT ARLINGTON

Advanced Topics in Software Engineering
CSE 6324 – Section 004
Team 10

ITERATION 2
(Written Deliverable)

Vanjari, Vaishali Sunil - 1001956614
Solanki, Siddhrajsinh Pradumansinh - 1001957988
Parimi, Taraka Naga Nikhil - 1001985955
Komireddy, Sannihith Reddy – 1001982437

I. Project Proposal

The plan is to extend the Slither analysis framework by adding two detectors. Slither[1] is the open-source static analysis framework for Solidity that may be used to find holes in smart contracts. Red, yellow, and green are used by Slither to denote the impact and severity of vulnerabilities, correspondingly, for high, medium, low, and informational[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]

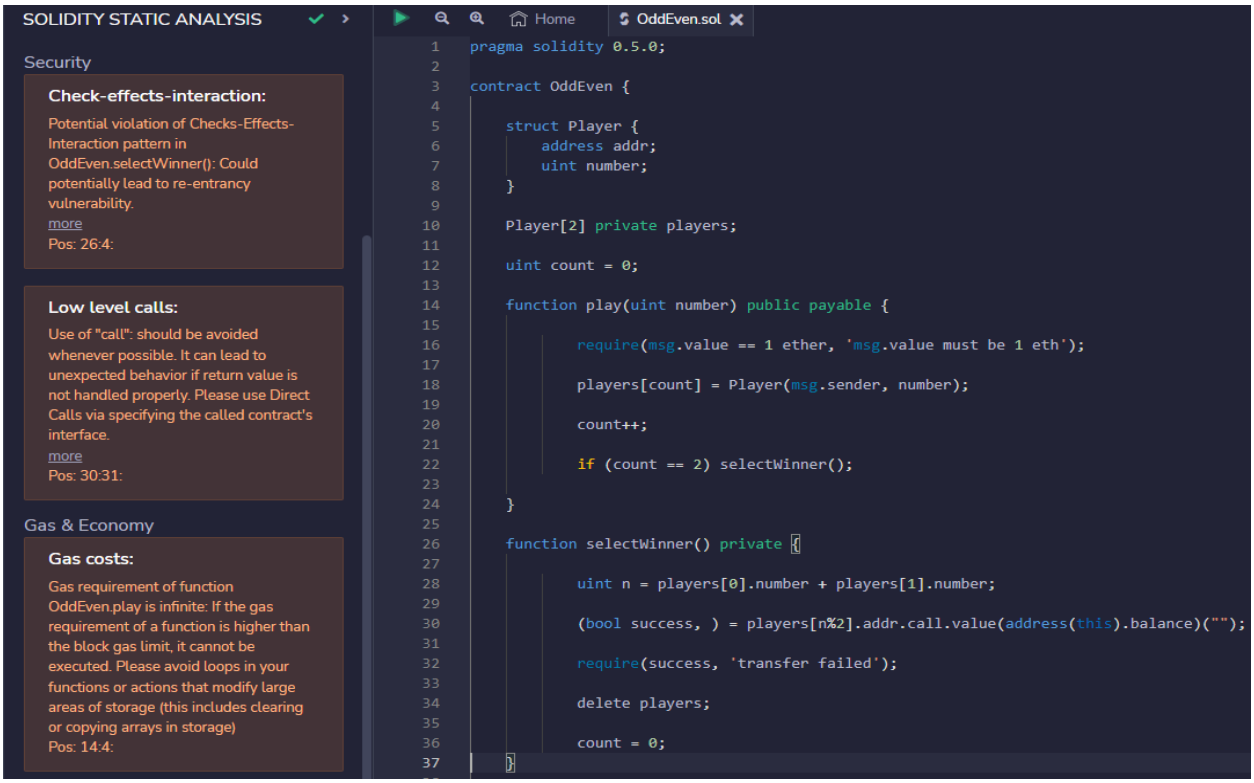
Adding a detector to find unencrypted private data on-chain is the main goal of the project plan for this iteration. Often held misconception: Private data variables cannot be read. Although the contract is not disclosed, attackers can learn about its status by looking at contract transactions. To store private data on-chain or off-chain, it must be encrypted[3]. Alternatively, private data can be modified in pure private function and that private function can be called in public function.

Adding another detector to find incorrect constructor name can make the smart contract exploitable to attacks. Constructors were formerly identified in Solidity v0.4.22 as functions with the same name as the contract they were contained in. If the contract name gets changed in development, then declaring constructor using function name will act as normal callable function. Especially if the constructor is conducting privileged activities, this could have disastrous results [12].

II. Competitor – Remix IDE

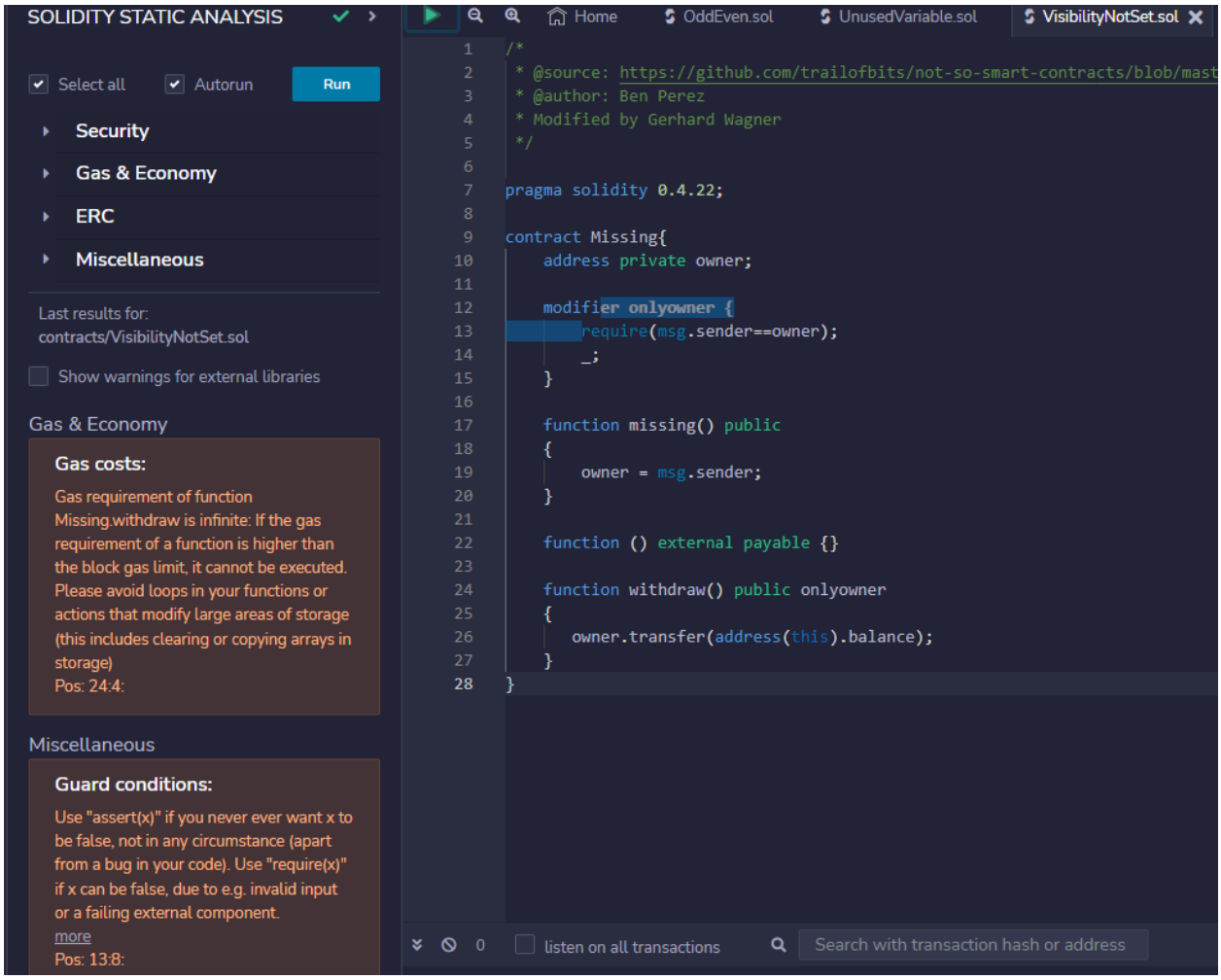
i) Detector to detect CWE-767:

We reviewed the current static analysis performed by Remix IDE before implementing an additional detector to detect the vulnerability CWE-767 but were unable to locate the suggested vulnerability analysis. Static analysis was done by Remix IDE, which produced an analysis on five security problems or vulnerabilities of varying seriousness, including reentrancy, low-level calls, gas prices, and guard circumstances.



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

Before adding a second detector to find the incorrect constructor's name, we checked the static analysis currently being done by Remix IDE but were unable to find the indicated vulnerability analysis. Remix IDE performed static analysis, which resulted in an examination of security issues or vulnerabilities of varied significance, such as 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	02/23/2023 (Followed)	02/26/2023 (Followed)	Completed

	b) Find the sample solidity contract to analyze it using Slither c) Work on reviews from Inception			
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)	04/01/2023 (Followed)	Completed
9.	a)Write a detector to detect the incorrect constructor name. b)Analyze the solidity contract with the new detector	04/02/2023	04/06/2023	In Progress
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 must ensure that it is compatible with the detector as it changes every year.	Maintaining a constant interpretation of solidity in the detector.	Risk impact: 3 weeks Probability that risk will materialize: 45% Risk Exposure: 2 weeks approx.
2.	Technical issue – inexperience with Python	One of the teammates lacks Python programming experience, and Slither is a Python-compatible tool.	Take the time to learn Python.	Risk impact: 5 weeks Probability that risk will materialize: 94% Risk Exposure: 4 weeks approx.
3.	Technical issue – inexperience with Solidity	The lack of solidity smart contract knowledge shared by all the teammates could hinder the development of the detector.	Understand solidity language as you progress.	Risk impact: 5 weeks Probability that risk will materialize: 96% Risk Exposure: 5 weeks approx.
4.	False-positive analysis	Slither can produce false-positive results, which should be handled and improved.	Need to research the vulnerabilities to get an understanding of false positives.	Risk impact: 5 weeks Probability that risk will materialize: 89% Risk Exposure: 3 weeks approx.

V. Specification and Design

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

The .sol extension will be used to build Solidity smart contracts, which can then be used as input for analysis. The analysis of the uploaded solidity smart contract will be the result. Sample solidity smart contract[3] odd_even.sol as input for the mentioned detector. Vulnerable code can be found in line no. 14 where public function modifying the private data which is unencrypted.

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

Analysis of the above odd_even.sol solidity contract without adding 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

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:~$
```

The following output after adding CWE-767 detector can be considered as final output:

```
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#18-23)
        -Illegal modification:
        -players[count] = Player(msg.sender,number)(odd_even.sol#20)
Reference: https://cwe.mitre.org/data/definitions/767.html
INFO:Detectors:
Reentrancy in OddEven.selectWinner() (odd_even.sol#25-31):
        External calls:
        - (success) = players[n % 2].addr.call.value(address(this).balance)() (odd_even.sol#27)
        State variables written after the call(s):
        - delete players (odd_even.sol#29)
        OddEven.players (odd_even.sol#15) can be used in cross function reentrancies:
        - OddEven.play(uint256) (odd_even.sol#18-23)
        - OddEven.selectWinner() (odd_even.sol#25-31)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
Reentrancy in OddEven.selectWinner() (odd_even.sol#25-31):
        External calls:
        - (success) = players[n % 2].addr.call.value(address(this).balance)() (odd_even.sol#27)
        State variables written after the call(s):
        - count = 0 (odd_even.sol#30)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
```

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

Solidity smart contracts, which can be utilized as input for analysis, 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 at line no. 17 where constructor name is incorrect, and attacker can take advantage of that.

```
 7 pragma solidity 0.4.22;
 8
 9 contract Missing{
10     address private owner;
11
12     modifier onlyowner {
13         require(msg.sender==owner);
14         _;
15     }
16
17     function missing() public
18     {
19         owner = msg.sender;
20     }
21
22     function () external payable {}
23
24     function withdraw() public onlyowner
25     {
26        owner.transfer(address(this).balance);
27     }
28 }
```

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 IncorrectConstructorName.sol

Pragma version0.4.22 (IncorrectConstructorName.sol#7) is known to contain se
vere issues (https://solidity.readthedocs.io/en/latest/bugs.html)
solc-0.4.22 is known to contain severe issues (https://solidity.readthedocs.
io/en/latest/bugs.html)Reference: https://github.com/crytic/slither/wiki/Det
ector-Documentation#incorrect-versions-of-solidity
IncorrectConstructorName.sol analyzed (1 contracts with 84 detectors), 2 res
ult(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
```

VI. Code and Tests

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 IncorrectConstructorName.sol
```

e) Create a Python file and add 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 .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]:

```
 7    IMPACT = DetectorClassification.HIGH
 8    CONFIDENCE = DetectorClassification.HIGH
 9    WIKI = "https://cwe.mitre.org/data/definitions/767.html"
10    WIKI_TITLE = "CWE767 example"
11    WIKI_DESCRIPTION = "Detector example"
12    WIKI_EXPLOIT_SCENARIO = ".."
13    WIKI_RECOMMENDATION = ".."
14
15    def _detect(self):
16        results = []
17
18        for contract in self.slither.contracts:
19            for function in contract.functions:
20                # for variable in function.variables:
21                # for opr in function.variables_written:
22                    # print("Opr ", "func:",function.name, " ",opr," ",opr.visibility, " ")
23                for node in function.nodes:
24                    for variable in node.state_variables_written:
25                        if function.visibility == "public" and variable.visibility == "private":
26                            info=["CWE-767:Access to Critical Private Variable in public method in ", function, "\n",
  "\t-","Illegal modification:","\n", "\t-", node.expression.__str__(),"(",node.source_mapping.__str__(),")\n"]
27                            # info+= ["\tIllegal modification:\n", "\t", node.expression,"\n"]
28                            res = self.generate_result(info)
29                            results.append(res)
30        return results
```

g) Analyze the smart contract again with newly added detector to get the expected result. Generate the report on analysis got after adding the new detector.

```
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#18-23)
        -Illegal modification:
        -players[count] = Player(msg.sender,number)(odd_even.sol#20)
Reference: https://cwe.mitre.org/data/definitions/767.html
INFO:Detectors:
Reentrancy in OddEven.selectWinner() (odd_even.sol#25-31):
        External calls:
        - (success) = players[n % 2].addr.call.value(address(this).balance)() (odd_even.sol#27)
        State variables written after the call(s):
        - delete players (odd_even.sol#29)
        OddEven.players (odd_even.sol#15) can be used in cross function reentrancies:
        - OddEven.play(uint256) (odd_even.sol#18-23)
        - OddEven.selectWinner() (odd_even.sol#25-31)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
Reentrancy in OddEven.selectWinner() (odd_even.sol#25-31):
        External calls:
        - (success) = players[n % 2].addr.call.value(address(this).balance)() (odd_even.sol#27)
        State variables written after the call(s):
        - count = 0 (odd_even.sol#30)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
```

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
3.	Running the solidity smart contract with Slither without adding the detector	Analysis of solidity smart contract without detecting incorrect constructor name.
4.	Running the solidity smart contract with Slither without adding the detector	Analysis of solidity smart contract with detecting incorrect constructor name.

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

VII. Customers and Users

#	Customer	Feedback/Suggestions
1.	Shubham Rathi (Solidity Beginner)	a) More sample solidity contracts must be tested for the recently introduced detector.
2.	Ashwini Shenvi (Solidity Enthusiast)	a) Simple to grasp the concept. b) Would like to know how such detector will identify an incorrect constructor name
3.	Darshan Patil (CSE 6324-Team 2)	a) Need to work on false positive analysis if private function is modifying private data and calling it through public function.

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
10. a-sample-contract-visibility-and-getters
11. Incorrect-constructor-name
12. Blog-sigmaprime-io/solidity-security#constructors