THE UNIVERSITY OF TEXAS AT ARLINGTON

Advanced Topics in Software Engineering
CSE 6324 – Section 004

Team 1:

SAICHARAN PAGIDIMUNTHALA – 1002006773

RAMYA MADDINENI - 1001965818

BHARGAV SUNKARI - 1002028016

DILEEP KUMAR NAIDU RAVI - 1002023397

ITERATION-2

GitHub Link: https://github.com/saicharan1248/cse6324_team1_project.git

I. OVERVIEW

Our objective is to address a specific issue that has been raised in Slither[1], an open-source project that provides static analysis for Solidity smart contracts. The issue involves a bug that has been identified and reported by the Slither community or external contributors.

Issue : Slither-flat ignores custom errors defined outside of the contract. #1410

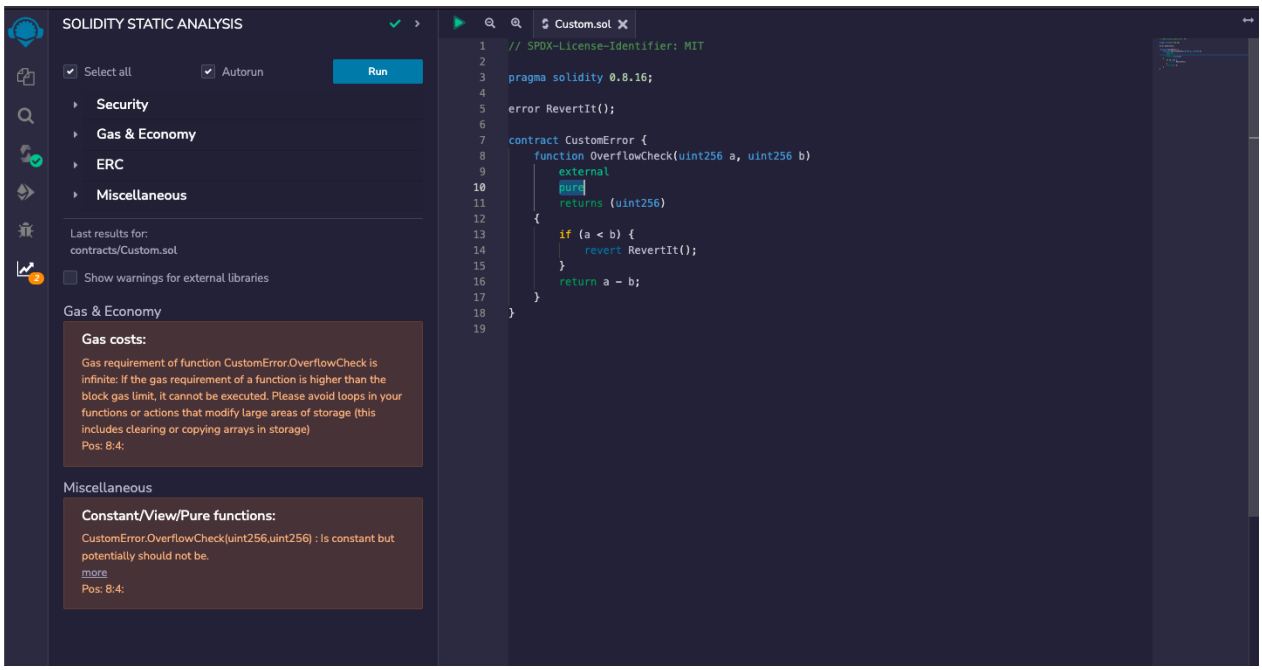
Slither uses a list of tools for their analysis,

- slither-check-upgradeability : Review delegate call-based upgradeability.
- slither-prop : Automatic unit test and property generation
- slither-flat : Flatten a codebase.
- vslither-check-erc : Check the ERC's conformance.
- slither-format : Automatic patch generation
- slither-read-storage : Read storage values from contracts.

In this iteration, the project plan is focused on analysing Custom Errors using Slither. There is this issue that a Custom Error cannot be read. slither-flat is responsible for flattening Solidity contract imports in one. This returns a flattened contract without errors defined and results in irrelevant and bad analysis. Hence custom errors should be read during flattening and analysing the contract.

II. COMPETITOR - REMIX IDE

We checked the possible vulnerability that can possibly be caused by that issue, and this is what we found on the analysis report generated by Solidity Static Analysis extension[1]. The storage warning is irrelevant as the custom error is defined outside the contract which is misread by the slither-flat tool.



REF[1]

III. PROJECT PLAN

#	Task Description	Start Date (Anticipated/Followed)	End Date (Anticipated/Followed)	Status (Completed/Incomplete)
1.	Installation a) Python v3 b) solc compiler c) solc select d) Slither	02/16/2023 (Followed)	02/22/2023 (Followed)	Completed
2.	a) Research about the issue b) Write a sample contract to run analysis c) Gather information about slither tools utilised in analysing the contract	02/23/2023 (Followed)	02/26/2023 (Followed)	Completed
3.	a) Setup remix ide for development and load slither extension. b) Load and gather reports from the analysis on the sample contract.	02/27/2023 (Followed)	03/01/2023 (Followed)	Completed
4.	a) Understand the slither tools code base b) Review the scope of slither-flat.	02/27/2023 (Followed)	02/27/2023 (Followed)	Completed
5.	a) Find flattening.py code for the issue. b) Review compilation unit for top level custom errors in compilation_unit.py. c) Decide either to modify the existing code or write new.	02/27/2023 (Followed)	02/27/2023	In-Progress
6.	a)Add error_top_level to compilation unit. b) Update flattening.py to read the custom errors from top level.	03/11/2023	03/25/2023	In-Progress

7.	a) Test the compilance_unit for bugs and issues. b) Analyse the contract with the new compliance unit. c) Work on reviews from Iteration 2	03/11/2023	03/25/2023	Incomplete
8.	a) Run tests with multiple contracts. b) Generate reports and check for other issues in slither tools.	04/07/2023	04/24/2023	Incomplete

IV. RISK FACTORS AND MITIGATION PLAN

#	**Risk**	**Description**	**Mitigation Plan**	**Risk Exposure**
1.	Lack of understanding of Solidity language	Solidity is a relatively new programming language, and developers may not have a lot of experience in working with it.	Spend time understanding the language before starting the work and seek advice from experts through the developer community.	Risk impact: 2 weeks Probability that risk will materialise: 92% Risk Exposure:1 week
2.	Technical issue - inexperience with Python	All my teammates have no experience working with Python, Slither written in python.	Spend time learning python.	Risk impact: 5 weeks Probability that risk will materialise: 96% Risk Exposure:3 week
3.	Technical issue - inexperience with Solidity	All my teammates have no experience working with Solidity smart contracts which can impact the development and understanding of the issue.	Spend time learning solidity language.	Risk impact: 5 weeks Probability that risk will materialise: 96% Risk Exposure:3 week
4.	Technical issue - inexperience in static analysis	All my teammates have no experience in static analysis on a smart contract.	Spend time understanding Slither.	Risk impact: 5 weeks Probability that risk will materialise: 96% Risk Exposure:3 week

5.	Lack of Testing	A lack of testing results in undiscovered bugs and vulnerabilities.	To do some comprehensive testing plan and find bugs.	Risk impact: 5 weeks Probability that risk will materialise: 96% Risk Exposure:3 week

V. SPECIFICATION AND DESIGN

I) Input and Output:

The input will be a solidity smart contract with .sol extension and can be used to do analysis. The output will be the analysis report of the contract uploaded. Sample solidity smart contract OverUnderFlow.sol as input is as follows:

Sample solidity smart contract[2] OverUnderFlow.sol

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.16;

// Errors
error OverFlow();
error UnderFlow();

contract OverUnderFlow {
    uint8 public salt = 100;

    // Check for Overflow
    function UnderFlowCheck(uint256 x) external view {
        if (x > salt) revert UnderFlow();
    }

    // Check for Underflow
    function OverFlowCheck(uint256 x) external view {
        if (uint256(x + salt) < type(uint8).max) revert OverFlow();
    }
}

```

REF[2]

Analysis of the above OverUnderFlow.sol solidity contract as output[3] is as follows:

```
root@MacBookAir~$slither OverUnderFlow.sol

Function OverUnderFlow.UnderFlowCheck(uint256) (OverUnderFlow.sol#12-14) is not in mixedCase
Function OverUnderFlow.OverFlowCheck(uint256) (OverUnderFlow.sol#17-19) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conv
entions

OverUnderFlow.salt (OverUnderFlow.sol#9) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-decla
red-constant
OverUnderFlow.sol analyzed (1 contracts with 85 detectors), 3 result(s) found
root@MacBookAir~$slither-flat OverUnderFlow.sol
INFO:Slither:Export crytic-export/flattening/OverUnderFlow_6cfde6d9-6210-4654-a48a-3516b8bff6aa.sol
root@MacBookAir~$
root@MacBookAir~$
root@MacBookAir~$cat crytic-export/flattening/OverUnderFlow_6cfde6d9-6210-4654-a48a-3516b8bff6aa.sol
pragma solidity 0.8.16;
contract OverUnderFlow {
    uint8 public salt = 100;

    // Check for Overflow
    function UnderFlowCheck(uint256 x) external view {
        if (x > salt) revert UnderFlow();
    }

    // Check for Underflow
    function OverFlowCheck(uint256 x) external view {
        if (uint256(x + salt) < type(uint8).max) revert OverFlow();
    }
}
root@MacBookAir~$
```

REF[3]

ii) Installation:

a) Install Python v3.6+

b) Install solc compiler[4]

```
brew update
brew upgrade
brew tap ethereum/ethereum
brew install solidity
```

REF[4]

c) Install solc-select[5] which allows you to switch between solidity compiler versions.

```
pip3 install solc-select
```

REF[5]

d) Install Slither using git.

Using Git

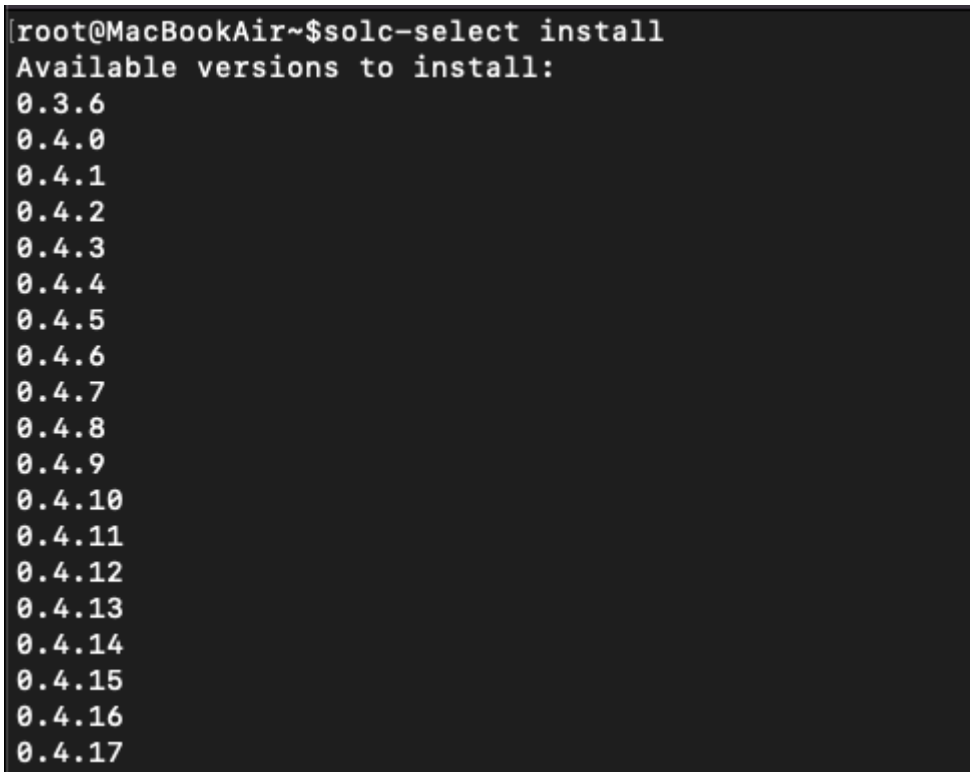
```
git clone https://github.com/crytic/slither.git && cd slither
python3 setup.py install
```

REF[6]

VI. CODE AND TESTS

i) Screenshots:

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



REF[7]

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

```
root@MacBookAir~$solc-select install 0.8.16
Installing '0.8.16'...
Version '0.8.16' installed.
root@MacBookAir~$
```

REF[8]

c) Run static analysis using the following command:

```
root@MacBookAir~$slither OverUnderFlow.sol

Function OverUnderFlow.UnderFlowCheck(uint256) (OverUnderFlow.sol#12-14)
 is not in mixedCase
Function OverUnderFlow.OverFlowCheck(uint256) (OverUnderFlow.sol#17-19)
is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
#conformance-to-solidity-naming-conventions

OverUnderFlow.salt (OverUnderFlow.sol#9) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
#state-variables-that-could-be-declared-constant
OverUnderFlow.sol analyzed (1 contracts with 85 detectors), 3 result(s)
found
root@MacBookAir~$
```

REF[9]

d) Flatten a contract using the following command:

```
root@MacBookAir~$slither-flat OverUnderFlow.sol
INFO:Slither:Export crytic-export/flattening/OverUnderFlow_6ef3b575-771a
-444b-8aee-c79ffa3b4ca5.sol
root@MacBookAir~$
```

REF[10]

e) To fix the issue with slither-flat, some functions need to be added in the slither/tools/flattening/flattening.py file which does the flattening during analysis. From L76:79 reads the top level imports and declarations.

```
72
73          for contract in compilation_unit.contracts:
74              self._get_source_code(contract)
75
76          self._get_source_code_top_level(compilation_unit.structures_top_level)
77          self._get_source_code_top_level(compilation_unit.enums_top_level)
78          self._get_source_code_top_level(compilation_unit.variables_top_level)
79          self._get_source_code_top_level(compilation_unit.functions_top_level)
80
81      def _get_source_code_top_level(self, elems: Sequence[TopLevel]) -> None:
82          for elem in elems:
83              self._source_codes_top_level[elem] = elem.source_mapping.content
84
```

REF[11]

f) From Line 81:95 We wrote this method to extracts the custom errors defined outside of a contract from **compilation_unit.custom_errors** dictionary and generates a string containing the custom error definitions in the proper.

```
71          self._check_abi_encoder_v2()
72
73          for contract in compilation_unit.contracts:
74              self._get_source_code(contract)
75
76          self._get_source_code_top_level(compilation_unit.structures_top_level)
77          self._get_source_code_top_level(compilation_unit.enums_top_level)
78          self._get_source_code_top_level(compilation_unit.variables_top_level)
79          self._get_source_code_top_level(compilation_unit.functions_top_level)
80
81      def _get_source_code_custom_errors(self, compilation_unit):
82
83          source_code = ""
84          for error_name in compilation_unit.custom_errors:
85              error_node = compilation_unit.custom_errors[error_name]
86              args = [f"{arg['name']}:{arg['type']}" for arg in error_node.arguments]
87              source_code += f"error {error_name}({','.join(args)});\n"
88
89          source_code += self._get_source_code_custom_errors(compilation_unit)
90          source_code += self._get_source_code_top_level(compilation_unit.structures_top_level)
91          source_code += self._get_source_code_top_level(compilation_unit.enums_top_level)
92          source_code += self._get_source_code_top_level(compilation_unit.variables_top_level)
93          source_code += self._get_source_code_top_level(compilation_unit.functions_top_level)
94
95          return source_code
96
97      def _get_source_code_top_level(self, elems: Sequence[TopLevel]) -> None:
98          for elem in elems:
99              self._source_codes_top_level[elem] = elem.source_mapping.content
100
```

ii) Test Case:

#	Test Case	Expected Output
1.	Running the solidity smart contract with Slither flat without top level errors added.	Flattened solidity smart contract with missing custom errors.
2.	Running the solidity smart contract with top level errors added.	Flattened solidity smart contract with custom errors.

VII. CUSTOMERS AND USERS

#	Customer	Feedback/Suggestions
1	Aishwarya Kalmangi (CSE 6324-Team 2)	More sample solidity contracts must be tested for This method to Work.
2	Sai Nikhil Kanchukatla (CSE 6324-Team8)	Simple method to implement But, still need some modification .

FURTHER ITERATIONS:

1. Iteration 1: Installing dependencies and reproducing the issue.
2. Iteration 2: Reviewing the existing code and analyse the smart-contract analysis, code, and flat module.
3. Iteration 3: Add top level analysis to core compilation unit and flat tool.

TOOL DIFFERENCE:

Tool	Truffle's Built-in Flattener	Solidity Flattener	Slither Flattening Tool
Language Support	Solidity only	Solidity only	Multiple languages
Installation Required	Yes	Yes	Yes
Output Format	Solidity source code	Solidity source code	Multiple formats
Customizability	Limited	None	Advanced

Security Features	Limited	None	Advanced
Integration with IDEs	Good	Poor	Good

I. REFERENCES

1. Slither, the Solidity source analyzer [1]
2. Contract Flattening in Slither flat [2]
3. Solc compiler [3]
4. Solc select [4]
5. Installing Solidity compiler [5]
6. Installing solc-select [6]
7. Install Slither [7]
8. Tools: slither-flat [8]
9. Compilation Unit [9]
10. RemixIDE - Static analysis [10]