CSE – 6324: Advanced Topics in Software Engineering

Support for Imports with Alias in Slither Tool

(Either Symbol Alias or Unit Alias)

Final Iteration

Submitted By: Team 4

Preeti Singh - 1002013566

Sripal Thorupunoori - 1001969001

Mohit Singhi - 1002004892

Siddhartha Reddy Sungomula – 1001969005

TABLE OF CONTENT

1. Synopsis [1]

The Ethereum blockchain enables anyone to create a smart contract and carry it out via a decentralized method. Smart contracts are self-executing algorithms or pieces of data that run on publicly accessible blockchains. It essentially uses a software protocol to create, verify or implement a contract among any number of parties.

Such agreements take action once certain conditions are met and are everlasting. A code error could cause significant financial losses or other harm if a criminal uses it. In order to identify potential security flaws, errors, and deficiencies in smart contracts, the static analysis framework Slither is employed. The most important flaws that Slither may find include reentrancy, access control, arithmetic operations, integer overflow and underflow, uninitialized variables, gas limit, and gas pricing.

2. Architectural Design [1]

The adaptable, straightforward, and extensible Slither structure allows programmers to add their own analysis modules and change the analysis methodologies. Smart contract code is categorized and parsed onto an AST using the Slither Lexer and Parser modules. The AST is subsequently looked at by Slither for Data Flow Analysis to determine where info is being transferred via smart contract code. Problems like reentrancy attacks, uninitialized variables, and unbounded loops are made easier to spot with its assistance.

Through the use of Slither's Control Flow Analysis tool (CFG), the Smart Contract Code is investigated for its Control Flow Graph. CFG identifies issues such as unachievable code, wasteful code, and functional visibility.

A group of scanner elements in Slither expose the smart contract code to various specialized examinations. Utilizing AST, CFG, and Data Flow Evaluation, these tools are capable of detecting a variety of protection flaws. Unregulated operation conclusions, poor access control, integer overflows, and underflows are a few of them. Once all flaws in the smart contract code have been discovered, Slither delivers an evaluation. Along with describing the problem's location and severity, the study offers suggestions for fixing it.

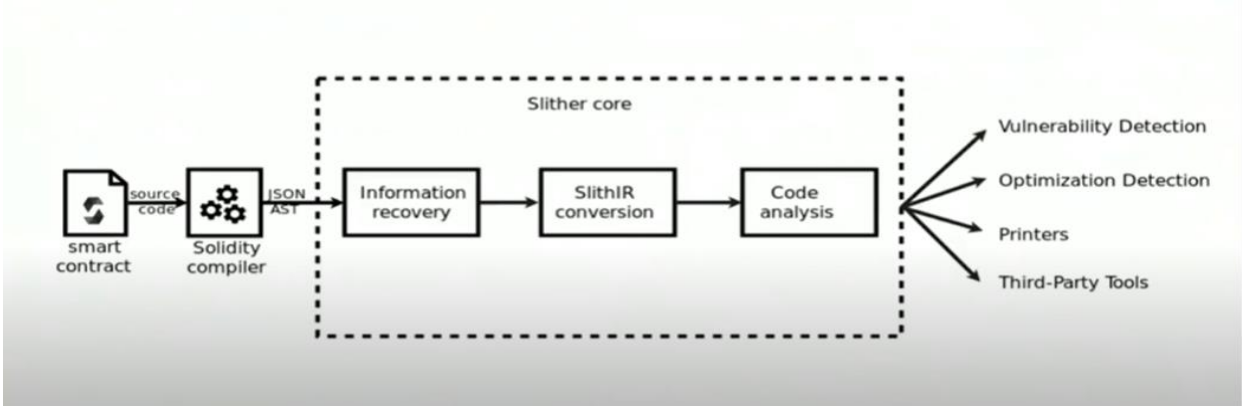


Figure 1.1: Slither Overview [1]

3. An Outline of Implementation

Iteration	Goals Targeted	Goals Achieved
1	▪ Install Slither and understand the tool. ▪ Analyze the most closely related tools. ▪ Investigate/understand the issue in detail and find out what inputs, outputs, and data structures were used in detail. ▪ Identify and provide the necessary test cases for the important code elements. ▪ Assess the major risks and devise realistic plans to mitigate them. ▪ Examine how the project's agenda impacts the user.	▪ Installed the Slither and understood the tool. ▪ Analyzed the competitors like Mythril, Manticore. ▪ Investigated the alias issue in detail and found out what inputs, outputs, transition graph, and data structures. ▪ Identified the important code elements and written the necessary test cases for the same. ▪ Assessed the major risks and made necessary realistic plans to mitigate them. ▪ Examined how the support to import with alias will impact the user.

2	▪ Start implementing "support for alias import". ▪ Reevaluate the risks. ▪ Prepare a risk mitigation plan. ▪ Conduct user testing and obtain appropriate user feedback.	▪ Studied various issues related with current issue. ▪ Analyzed the code and performed code changes. ▪ Overcame assertion issue and able to run the smart contract
3	▪ Ensure that all deliverables are completed on time. ▪ Test the important code elements with the necessary test cases. ▪ Get feedback from users in the final stage.	

4. Key Data Structures used in Slither: [2]

Slither analyzes and pinpoints software bugs in Solidity using a variety of datatypes. Slither contains a variety of important data types, including:

- **AST:** Once Slither converts Solidity code, an Abstract Syntax Tree (AST), that mimics a collection of branches and embodies the syntax hierarchy, is produced. It is used to gather information on the steps, settings, and directives that make up the code framework's execution flow.
- **Control Flow Graph:** Slither creates a Control Flow Graph (CFG) from the AST, showing the way efficiently the program's logic is controlled. In order to find potential weaknesses in security, such as reentrancy and gas-related problems, the algorithm's pathways are examined.
- **Data Flow Diagram:** In a nutshell, Slither uses the AST to produce a Data Flow Diagram (DFG) which demonstrates how information moves within the program's code. To discover software flaws like integer

overflows or underflows, the DFG analyzes the duplicated information contained in the programming language.

- **Symbol Table:** In Slither, attributes are listed in a symbol table with details on what they are and identities. It is used to identify byte formats in programs with the goal to find possible issues such type inconsistencies and uninitialized elements.

5. Implementation of the issue.[3][4]

- **Environment Setup**

Instructions to Compile and Run (Platform: Mac)

Compile:

1. Install python3 →

   - Update the package list with the following command →
     `Sudo apt -get update`
   - Install Python by typing the following command →
     `sudo apt-get install python`
   - You can verify that Python has been installed by typing the following command →
     `python —version`

     We have used version 3.10.5 for python.

2. Install Slither analyzer (using below commands) →

   `pip3 install slither-analyzer`

3. Install the solc-select package and its dependencies →
   `pip3 install solc-select`

4. To install solc version 0.4.25, type the following command →
   `solc-select install 0.4.25`

5. Use solc version 0.4.25, type the following command →
   `solc-select use 0.4.25`

6. Create a copy of a Git repository, including all its files and version history, on your local machine →
   `git clone http://github.com/crytic/slither`

7. To change to the appropriate directory, use the command →
   `cd slither`

8. Install the slither package that you cloned from GitHub using the git clone command →
   `Python3 setup.py install`

Run:

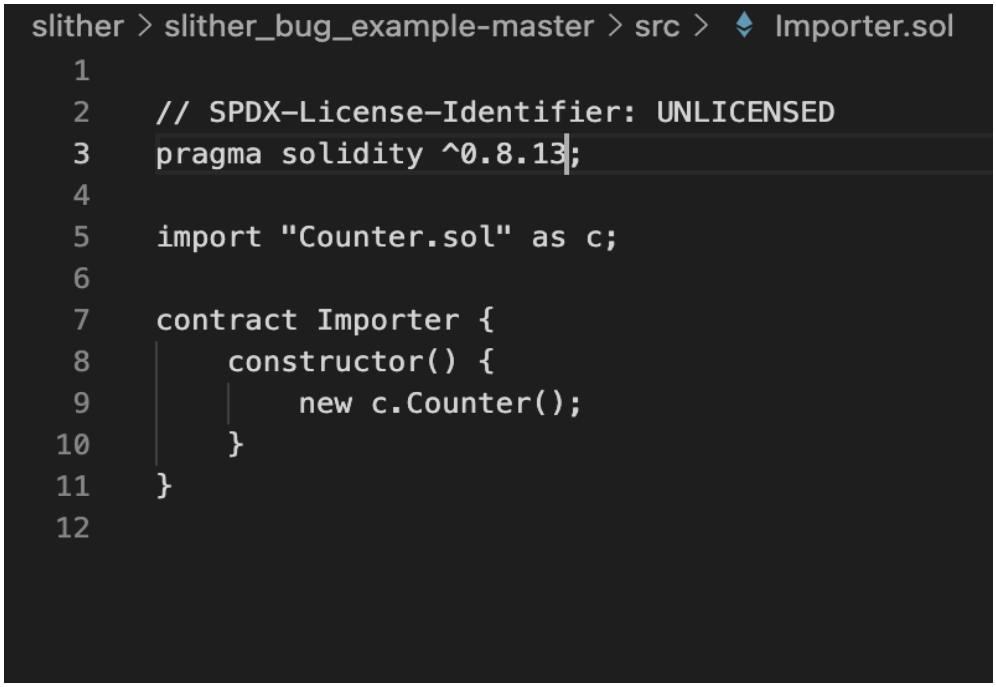
1. After changes to the code, navigate to the directory where the cloned repository of Slither is saved and build and install the changes on your system using the command →
   `pip install .`

2. Run the Slither command with the Solidity filename i.e Importer.sol →
   `slither Importer.sol`

- **Issue Recreation and Error Trace**

  Recreated the Issue:

  We recreated the issue by executing the smart contract "Counter.sol" and then importing it as c in another contract "Importer.sol".



A Solidity smart contract "Importer.sol" imports the "Counter.sol" contract using the import statement and aliases it as "c". The Importer contract's constructor then creates an instance of the Counter contract.

slither > slither_bug_example-master > src > Counter.sol

```
1   // SPDX-License-Identifier: UNLICENSED
2   pragma solidity ^0.8.13;
3
4   contract Counter {
5       uint256 public number;
6
7       function setNumber(uint256 newNumber) public {
8           number = newNumber;
9       }
10
11      function increment() public {
12          number++;
13      }
14  }
15
```

In the above code, a basic counter is defined using Solidity. It contains a state variable **number** of type uint256 that represents the current count. The contract has two functions that modify the state variable, **setNumber()** and **increment()**. The setNumber() function takes a new number as an argument and updates the number variable accordingly. A number variable is incremented by one using the increment() function.

Traceback error:

```
    result = apply_ir_heuristics(result, node)
  File "/Users/preetisingh/anaconda3/lib/python3.9/site-packages/slither/slithir/convert.py", line 1925, i
apply_ir_heuristics
    irs = propagate_type_and_convert_call(irs, node)
  File "/Users/preetisingh/anaconda3/lib/python3.9/site-packages/slither/slithir/convert.py", line 463, in
propagate_type_and_convert_call
    new_ins = propagate_types(ins, node)
  File "/Users/preetisingh/anaconda3/lib/python3.9/site-packages/slither/slithir/convert.py", line 795, in
propagate_types
    ir.lvalue.set_type(UserDefinedType(contract))
  File "/Users/preetisingh/anaconda3/lib/python3.9/site-packages/slither/core/solidity_types/user_defined_
ype.py", line 20, in __init__
    assert isinstance(t, (Contract, Enum, Structure))
AssertionError

(base) Preetis-MacBook-Air:src preetisingh$
```

The message of error suggests that there was an assertion issue in the code. An assertion fault was reported in **user_defined_type.py's __init__ method** for the **UserDefinedType** class. This error was fixed in the previous iteration by declaring an alias_check method that checks if there is an ast.Alias object with kind set to 'contract' and name set to 'c'. A contract instance with the same name will be returned using the **get_contract** method if such an object is found. It will return None if the contract cannot be found. In this way, the **UserDefinedType** object will be created with the correct contract object and can be used properly.

```
@staticmethod
def alias_check(alias, module):
    if alias is None:
        return None

    if isinstance(alias, ast.Alias) and alias.kind == 'contract':
        instance = module.get_contract(alias.name)
    elif isinstance(alias, ast.Alias) and alias.kind == 'enum':
        instance = module.get_enum(alias.name)
    elif isinstance(alias, ast.Alias) and alias.kind == 'structure':
        instance = module.get_structure(alias.name)
    else:
        return None

    if instance is None:
        return None

    if isinstance(instance, ast.Alias):
        return UserDefinedType.alias_check(instance.alias, module)

    return instance.name
```

- **Implementation & Solutio**
  Input for Iteration:

Based on the previous iteration output, we modified **solc_parsing/declarations/contract.py** to resolve the error. Below is the code for the function **parse_contract**, which extracts an abstract syntax tree (AST) of a Solidity contract. The **import_node** is checked if it has an alias attribute with the if-else block. In this case, we add the alias to the **references** set. Otherwise, we add the path to

the references set. For contracts with import statements with aliases, this should ensure the references attribute is set correctly.

```
110     def parse_contract(self, contract):
111         contract_node = ast.AST.Contract()
112         contract_node.name = contract['name']
113         contract_node.type = 'contract'
114         contract_node.is_abstract = contract['is_abstract']
115         contract_node.is_interface = contract['is_interface']
116 
117         for stmt in contract['body']:
118             if stmt['type'] == 'EnumDefinition':
119                 enum_node = self.parse_enum(stmt)
120                 contract_node.enums.append(enum_node)
121             elif stmt['type'] == 'StructDefinition':
122                 struct_node = self.parse_struct(stmt)
123                 contract_node.structs.append(struct_node)
124 
125             elif stmt['type'] == 'ImportDirective':
126                 import_node = self.parse_import(stmt)
127                 contract_node.imports.append(import_node)
128                 if import_node.alias:
129                     contract_node.references.add(import_node.alias.alias)
130                 else:
131                     contract_node.references.add(import_node.path)
132             elif stmt['type'] == 'PragmaDirective':
133                 contract_node.pragmas.append(self.parse_pragma(stmt))
134 
135         return contract_node
```

In the below modified code, a new function is added to convert.py to handle "import ... as ..." syntax. Also, the **parse_import** function now checks for "as" keywords and calls the new function if they are found.

```
153     # Provide an additional function for handling 'import ... as ...'
154     def parse_import_alias(self, stmt):
155         import_alias_node = ast.AST.ImportAlias()
156         import_alias_node.name = stmt['name']
157         import_alias_node.alias = stmt['alias']
158         return import_alias_node
159 
```

The above function parses an import statement with an alias and creates an abstract syntax tree (AST) node that represents the import statement. A new instance of the class **ast.AST.ImportAlias()** is created and assigned to the variable **import_alias_node**. It then sets the name attribute of **import_alias_node** to the value of the name key in the stmt dictionary. Afterwards, **import_alias_node**'s alias attribute is set to the alias key in the **stmt** dictionary. In the end, it returns an object named **import_alias_node**.

```
160     # Use the new function parse_import to check for the 'as' keyword
161     def parse_import(self, stmt):
162         import_node = ast.AST.Import()
163         import_node.path = stmt['path']
164         if 'as' in stmt:
165             import_node.alias = self.parse_import_alias(stmt['as'])
166         return import_node
```

As part of the import statement, if an alias is included, the function **parse_import** creates a node for the alias and adds it to the import node.

Error -

```
(base) Preetis-MacBook-Air:slither_bug_example-master preetisingh$ cd src/
(base) Preetis-MacBook-Air:src preetisingh$ slither Importer.sol
ERROR:ContractSolcParsing:Impossible to generate IR for Importer.constructor (Importer.sol#9-12):
 'NoneType' object has no attribute 'references'
INFO:Detectors:
Pragma version^0.8.9 (Counter.sol#2) allows old versions
Pragma version^0.8.9 (Importer.sol#3) allows old versions
solc-0.8.19 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Slither:Importer.sol analyzed (2 contracts with 84 detectors), 3 result(s) found
(base) Preetis-MacBook-Air:src preetisingh$
```

Explanation - The error message indicates that the **Importer.constructor** function in the **Importer.sol** contract still is not able to generate an intermediate representation (IR). Solidity's parser failed to parse some part of the code due to the **NoneType** object error. Additional details about detected issues are provided in the **INFO** section of the message, including the pragma version and warnings about using a particular version of Solidity. There is no connection between these warnings and the NoneType object error.

6. Test Cases

Test Case 1: The below test case is for smart contract Importer.sol, which uses "Import …. As …" Syntax

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.9;

import "./Counter.sol" as c;

contract TestImport_Alias {
    c.Counter public cnt;

    constructor() {
        cnt = new c.Counter();
    }

    function TestCaseAlias() public view {
        bool exists = ContractAddress(address(cnt));
        assert(exists);
    }

    function ContractAddress(address ad) private view returns (bool) {
        uint size;
        assembly { size := extcodesize(ad) }
        return size > 0;
    }
}
```

Explanation – The **TestImport_Alias** imports the Counter contract and tests its existence using **TestCaseAlias()** and **ContractAddress()**. By using the as keyword, the contract aliases Counter.sol as "c". As a result, the **TestImporter** contract can refer to Counter using the alias "c" instead of its original name.

In the **TestImport_Alias** contract, there is a public variable named **cnt**, which is of type **c.Counter**. An instance of the Counter contract is stored in this variable. A new instance of the Counter contract is created and stored in the counter variable by the constructor of the **TestImport_Alias** contract. **TestCaseAlias()** checks for the existence of an imported Counter contract using a public function. The function calls a private function called **ContractAddress()**, which takes an address as input and returns a boolean value indicating whether a contract exists at that address.

Output for Test Case 1 :

```
(base) Preetis-MacBook-Air:src preetisingh$ slither TestWithAlias.sol
ERROR:ContractSolcParsing:Impossible to generate IR for TestCounter.constructo
 'NoneType' object has no attribute 'references'
INFO:Detectors:
TestCounter.counter (TestWithAlias.sol#9) is never initialized. It is used in:
        - TestCounter.testSetNumber() (TestWithAlias.sol#15-18)
        - TestCounter.testIncrement() (TestWithAlias.sol#20-24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninit
INFO:Detectors:
Pragma version^0.8.9 (Counter.sol#3) allows old versions
Pragma version^0.8.9 (TestWithAlias.sol#3) allows old versions
solc-0.8.19 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incor
INFO:Detectors:
TestCounter.counter (TestWithAlias.sol#9) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-
INFO:Slither:TestWithAlias.sol analyzed (2 contracts with 84 detectors), 5 resu
(base) Preetis-MacBook-Air:src preetisingh$
```

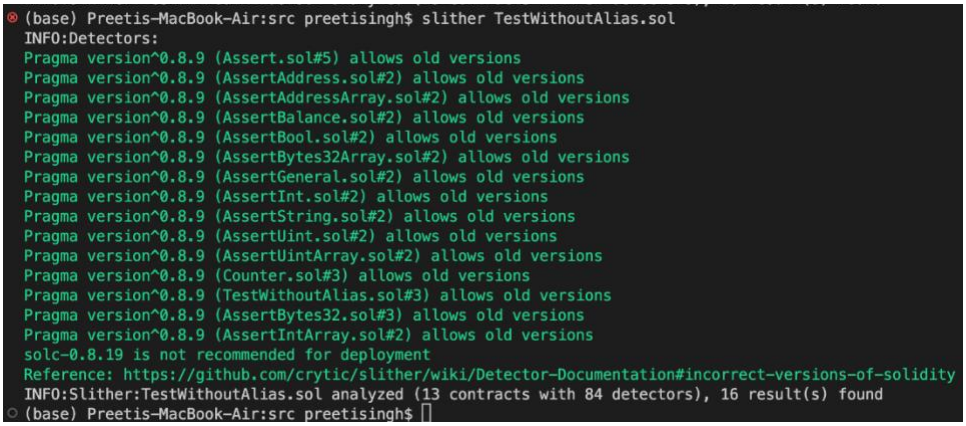
Explanation - This error message indicates that there is an issue with generating the Intermediate Representation (IR) for **TestImport_Alias.constructor** in **TestWithAlias.sol**. Counter.sol aliased as "c" cannot be imported by the **TestImporter**, resulting in the **TestImporter.counter** variable not being initialized.

Test Case 2 : The below test case is for smart contract Counter.sol, which does not uses any “Import …. As …" Syntax

```
3   pragma solidity ^0.8.9;
4
5   import "./Assert.sol";
6   import "./Counter.sol";
7
8   contract TestCounter {
9     Counter counter;
10
11    function beforeEach() public {
12      counter = new Counter();
13    }
14
15    function testSetNumber() public {
16      counter.setNumber(10);
17      bool success = Assert.equal(counter.number(), 10, "Number should be set to 10");
18      assert(success);
19    }
20
21    function testIncrement() public {
22      counter.setNumber(0);
23      counter.increment();
24      bool incremented = Assert.equal(counter.number(), 1, "Number should be incremented by 1");
25      require(incremented, "Number should be incremented by 1");
26    }
27  }
```

Explanation - An example test case for "**Counter.sol"** contract is defined in this Solidity smart contract. Firstly, it imports the **Assert.sol** and **Counter.sol** contracts. As you can see, we are just importing both smart contracts directly and are not using the "as" function.

Output for Test Case 2 :



Explanation – Although, the warnings that the pragma version being used in the imported files and the main file allow old versions of Solidity, that could be improved upon. The output from the **slither** command shows that the file **TestWithoutAlias.sol** was analyzed and that 17 results were found.

7. An Overview of the Closer Ties [1]

Compared to certain other tools, Slither may have a number of advantages, such as:

- ❖ Software flaws are easier to detect because Slither's Intermediate Representation (IR) abstracts and simplifies Solidity code representation. Slither hides some low-level Solidity program details with IRs (Intermediate Representations) so that it may focus on higher abstractions. By performing this, the examination and conclusion can reason about the code with greater ease and identify potential safety problems. Slither may evaluate Solidity scripts more rapidly and effectively than certain tools that merely look at source code since it utilizes an IR.

❖ In comparison to other tools, Slither has a greater range of issues, such as reentrancy, issues with floating point correctness, and uninitialized storage variables.

❖ The identified issues are succinctly and plainly presented, along with recommendations for how to fix them. When Slither analyzes Solidity smart contract code, it provides detailed information on every potential information leak it discovers, including the flaw's nature, its location, and a synopsis of the problem.

❖ There are numerous acceptable formats, include simple text, CSV, and JSON. The tool can then be integrated into procedures and other applications, such as other analysis tools, thanks to this.

❖ Slither is the most accurate tool, with a 10.9% false positive rate. A summary of the accuracy findings, including rows for False positives, flagged contracts, and Detections per contract, is shown in the accompanying image.

		Slither	Securify	SmartCheck	Solhint
Accuracy	False positives	10.9%	25%	73.6%	91.3%
	Flagged contracts	112	8	793	81
	Detections per contract	3.17	2.12	10.22	2.16
Performance	Average execution time	0.79 ± 1	41.4 ± 46.3	10.9 ± 7.14	0.95 ± 0.35
	Timed out analyses	0%	20.4%	4%	0%
Robustness	Failed analyses	0.1%	11.2%	10.22%	1.2%
Reentrancy examples	DAO	✓	✗	✓	✗
	Spankchain	✓	✗	✗	✗

Figure 1.2: Slither outperforms its competitors. [1]

8. Risk Management

8.1 Biggest Risks

- The main risk may be failing to meet the iteration targets on schedule.
- Not that every computer system will support the program flawlessly. Resolution could be delayed as an effect of this.

- Another concern is the potential for unrecognized problems to emerge when resolving the existing problem. This could make the problem harder to fix.
- The largest risk is also not possessing enough expertise of the designated instrument, which could take longer than anticipated.
- Anyone on the team could experience an urgent situation at any time. This poses a risk to the team since, in the event of absent teammates, the burden on the remaining people escalates.

8.2 Impact of the risks

Sr no	Risk	Risk Exposure
1	One iteration's late completion can affect the following one, delaying the project.	P= 20% E=10; R.E= 2 hours
2	Such vulnerabilities have a significant effect since an unanticipated inaccuracy makes conflict resolution more difficult.	P= 10% E=30; R.E=3 hours
3	Due to their lack of awareness, the group must investigate further to learn about the problem.	P= 30% E=40, R.E= 12 hours
4	Development and output are impacted when members of the team are absent.	P= 5% E=10; R.E= 0.5 hours
	Where R.E is Risk Exposure	

8.3 Plans to Mitigate risks:

- To handle such circumstances, we must hold regular meetings to review our work.
- We must keep our tools current and adhere to best practices to limit risks from unanticipated errors.
- To mitigate this risk, we will have to put in more time learning the tool. using product manuals and online forums as resources.
- In these kinds of cases of emergency, the group must distribute the workload amongst the rest of the members to maintain team productivity.

8.4 Mitigated risks

- We have mitigated the risk of setting up virtual machine which was mentioned in previous iteration.
- To mitigate the issue and reduce the risk exposure we have tried installing slither tool on all of the team members' machines.
- With this mitigation we were able to work on the issue simultaneously reducing the time consumption and risk exposure.
- Dividing the total issue into small chunks and working on them helped to reduce the risk exposure.

9. Actual value to users/customers [3][4]

The "Support for imports with aliases" tool for Slither enables users to more accurately and efficiently study Solidity code, including imports with aliases. If items imported using aliases are not adequately monitored, Slither may miss vulnerabilities or weaknesses in the software.

If this feature is properly built, Slither will be able to monitor aliases easily and increase the quality of its discoveries. As a consequence, executable faults may be removed, making smart contracts more dependable and safer. By automating the examination of challenging code topologies, this feature may help expedite the analysis process and save engineers time and effort. When analyzing complex Solidity programs with several imported symbols or units, this type of capabilities is extremely useful.

GitHub Link:

- https://github.com/preetisingh1121/Slither.git

References -

1. https://arxiv.org/pdf/1908.09878.pdf
2. https://blog.trailofbits.com/2019/05/27/slither-the-leading-static-analyzer-for-smart-contracts/
3. https://github.com/crytic/slither/issues/1452
4. https://github.com/SheldonHolmgren/slither_bug_example/tree/master/src

5. https://github.com/crytic/slither/issues/1594
6. https://remix.ethereum.org/