CSE – 6324: Advanced Topics in Software Engineering

Support for Imports with Alias in Slither Tool

(Either Symbol Alias or Unit Alias)

Iteration 1

*Submitted By:*_**Team 4**

Preeti Singh	- 1002013566
Sripal Thorupunoori	- 1001969001
Mohit Singhi	- 1002004892
Siddhartha Reddy Sungomula	- 1001969005

TABLE OF CONTENT

1. Synopsis

The Ethereum blockchain lets anyone create a smart contract and run it over a distributed network. Smart contracts are self-executing programs or code that are established on blockchain networks. Essentially, it is a computer protocol that binds, validates, or enacts a contract between two or more parties.

Contracts of this type cannot be altered, and they are automatically executed if certain conditions are met. A vulnerability in the code can cause significant financial losses or other damage if attackers exploit it. To identify potential security vulnerabilities, flaws, and bugs in smart contracts, Slither (static analysis framework) is implemented. A few of the major vulnerabilities Slither can detect are reentrancy, access control, arithmetic operations, integer overflow and underflow, uninitialized variables, gas limit and gas price, and code quality.

2. Architectural Design

The Slither modular architecture is adaptable, easy-going, and extensible, allowing developers to add their own analysis modules and customize the analysis process. Smart contract code is tokenized and parsed into an Abstract Syntax Tree (AST) by Slither's Lexer and Parser components. Then, the AST is analyzed by Slither for Data Flow Analysis to determine how data is flowing through the smart contract code. It helps to detect issues like reentrancy attacks, uninitialized variables, and loops that are unbounded.

Using Slither's Control Flow Analysis component, the Smart Contract Code is analyzed for its Control Flow Graph (CFG). CFG identifies issues like unreachable code, dead code, and function visibility. There is a set of Detector Modules that perform specific checks on the smart contract code in Slither. Several security issues can be detected using these modules using AST, CFG, and Data Flow Analysis. These include integer overflows, underflows, wrong access control, and unchecked call results. After all security issues have been identified in the smart contract code, Slither generates a report. Furthermore, the report outlines where the issue is located, the severity of the issue, and suggestions for fixing it.

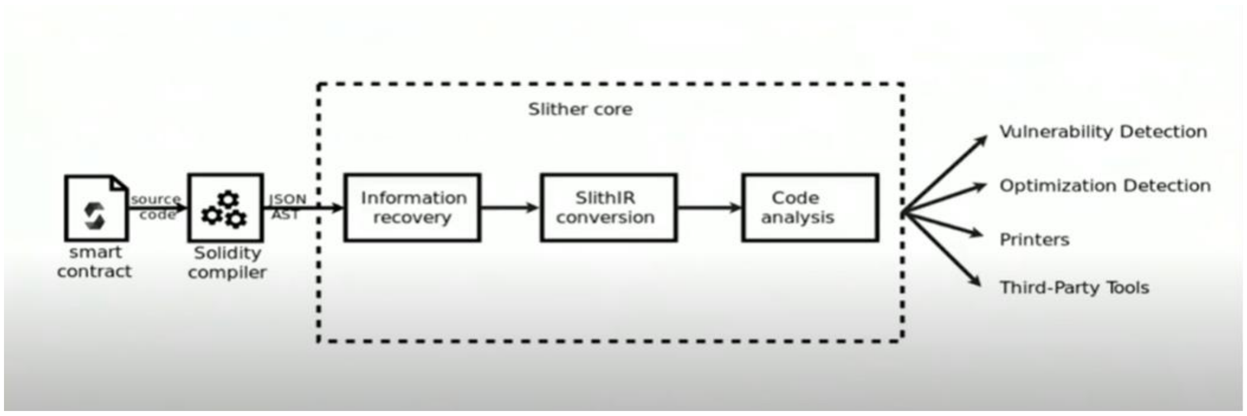


Figure 1.1: Slither Overview [1]

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.	
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 examines and finds vulnerabilities in Solidity programming using a variety of data structures. There are several important data structures in Slither, including:

Abstract Syntax Tree (AST): Slither converts Solidity code into an Abstract Syntax Tree (AST), a tree-like structure that symbolizes the syntactic structure. It is used to extract information about the functions, variables, and control flow instructions that make up the code structure.

Control Flow Graph: Slither creates a Control Flow Graph (CFG) from the AST, which shows how the code is controlled. It is used to find potential security holes, such as reentrancy and gas-related problems, by examining the execution routes of the code.

Call Graph: Slither creates a call graph using the CFG, which represents the function calls in the code. Using the call graph, we can analyze the communication between functions and identify potential weaknesses, such as unchecked call return values and unsigned storage.

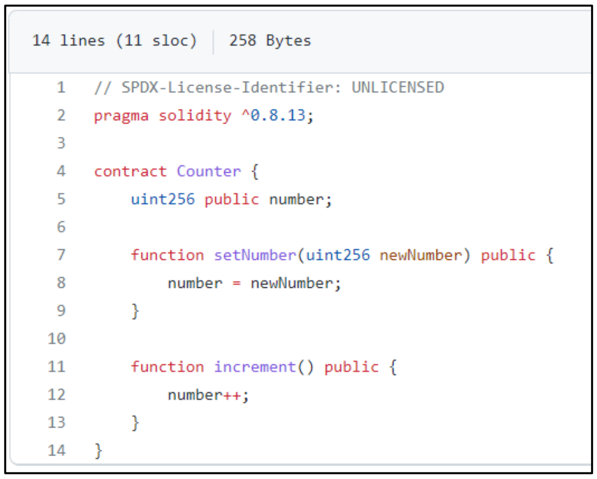
Data Flow Diagram (DFG): From the AST, Slither automatically generates a Data Flow Diagram (DFG), showing how data flows in the code. With the DFG, a security hole, such as an integer overflow or underflow, is identified by examining the code's data dependencies.

Symbol Table: In Slither, variables are listed in a symbol table that lists their kinds and names. It is used to identify data types in code and find the possible flaws, like type inconsistencies and uninitialized variables.

5. Inputs, Outputs and Transition Graph for the issue.[4]

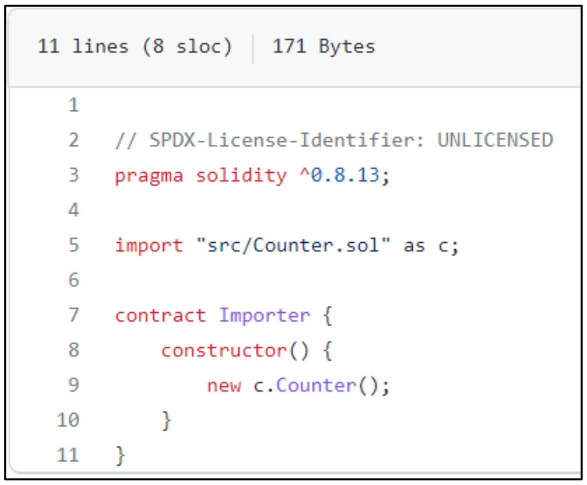
Inputs: The inputs in which the issue will arise are 1) A module in solidity which is imported and aliased into another module. Here, an example of such solidity module is Counter.sol given below. 2) A module which imports another solidity module and aliases it before using it. Here, an example of such solidity module is Importer.sol.

Counter.sol



```
14 lines (11 sloc) | 258 Bytes
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

contract Counter {
    uint256 public number;

    function setNumber(uint256 newNumber) public {
        number = newNumber;
    }

    function increment() public {
        number++;
    }
}
```

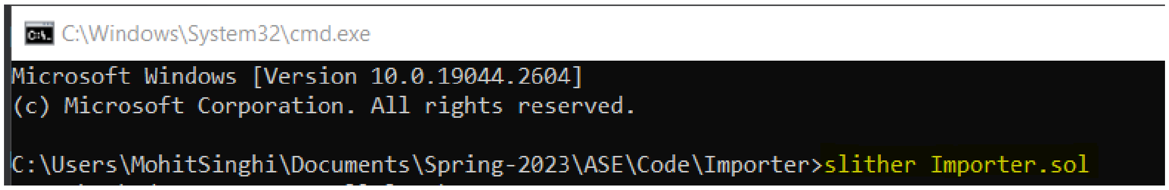
Importer.sol



```
11 lines (8 sloc) | 171 Bytes

// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "src/Counter.sol" as c;

contract Importer {
    constructor() {
        new c.Counter();
    }
}
```

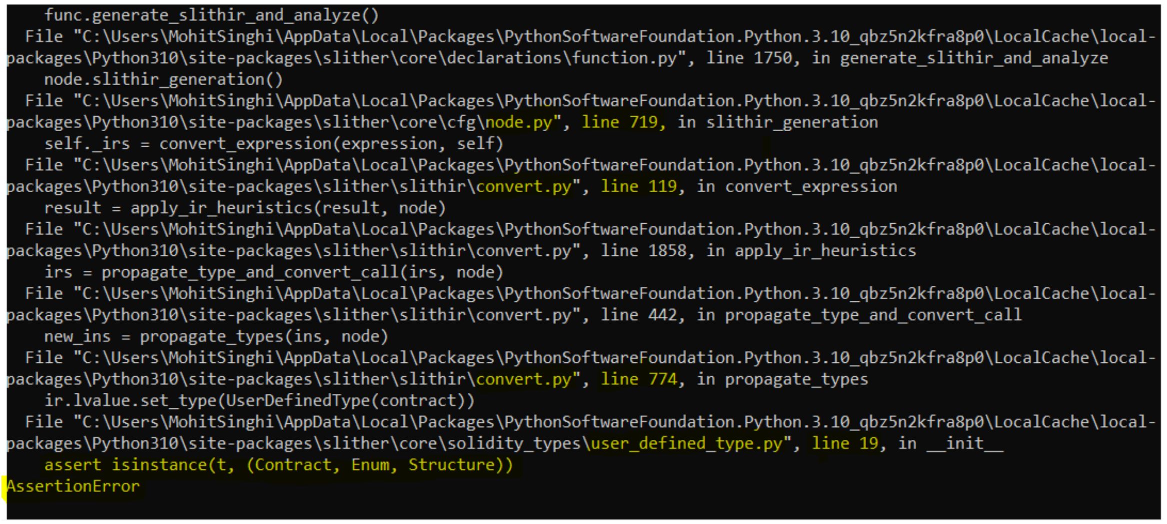
Output:

Re-created the Alias issue with Slither: To recreate this issue run slither analyzer on the above Importer.sol as can be seen in below, screenshot.



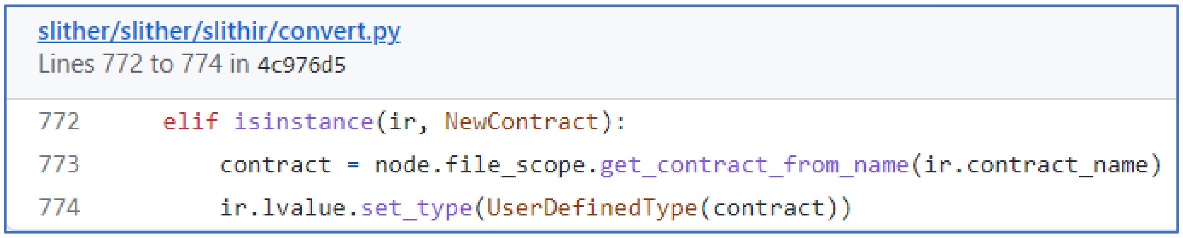
```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.2604]
(c) Microsoft Corporation. All rights reserved.

C:\Users\MohitSinghi\Documents\Spring-2023\ASE\Code\Importer>slither Importer.sol
```

Error stack trace: Below is the issue stack trace with AssertionError. The stack suggests issue with either of the three slither core files **user_defined_python.py** or **convert.py** or **node.py**



```
    func.generate_slithir_and_analyze()
  File "C:\Users\MohitSinghi\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-
packages\Python310\site-packages\slither\core\declarations\function.py", line 1750, in generate_slithir_and_analyze
    node.slithir_generation()
  File "C:\Users\MohitSinghi\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-
packages\Python310\site-packages\slither\core\cfg\node.py", line 719, in slithir_generation
    self._irs = convert_expression(expression, self)
  File "C:\Users\MohitSinghi\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-
packages\Python310\site-packages\slither\slithir\convert.py", line 119, in convert_expression
    result = apply_ir_heuristics(result, node)
  File "C:\Users\MohitSinghi\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-
packages\Python310\site-packages\slither\slithir\convert.py", line 1858, in apply_ir_heuristics
    irs = propagate_type_and_convert_call(irs, node)
  File "C:\Users\MohitSinghi\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-
packages\Python310\site-packages\slither\slithir\convert.py", line 442, in propagate_type_and_convert_call
    new_ins = propagate_types(ins, node)
  File "C:\Users\MohitSinghi\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-
packages\Python310\site-packages\slither\slithir\convert.py", line 774, in propagate_types
    ir.lvalue.set_type(UserDefinedType(contract))
  File "C:\Users\MohitSinghi\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-
packages\Python310\site-packages\slither\core\solidity_types\user_defined_type.py", line 19, in __init__
    assert isinstance(t, (Contract, Enum, Structure))
AssertionError
```

Analysis: After recreating the issue as per our analysis, the stack trace suggests issue in file **convert.py** at the below code block else-if condition. Further analysis is needed to understand why the aliased contract object cannot be type casted to a Solidity Contract type, which is why it gives as assertion error. [5]



slither/slither/slithir/convert.py
Lines 772 to 774 in 4c976d5

```
772        elif isinstance(ir, NewContract):
773            contract = node.file_scope.get_contract_from_name(ir.contract_name)
774            ir.lvalue.set_type(UserDefinedType(contract))
```

Instructions to Compile and Run (Platform: Windows)

Compile:

1. Install python3 and Solidity Compiler.
2. Install Slither (using below commands)

   ```
   pip3 install slither-analyzer
   ```

3. Set path to slither executable in Environment variables.

4. Verify if both tools are installed correctly by running below version commands:

   ```
   slither --version
   ```

   ```
   solc --version
   ```

5. Download the files <filenames> from GitHub to recreate the issue Compile files to check no issues with the compiler or code.

   ```
   solc --bin <file_name>.sol -o .
   ```

Run:

Run Slither analyzer on main issue file Importer.sol

```
slither Importer.sol
```

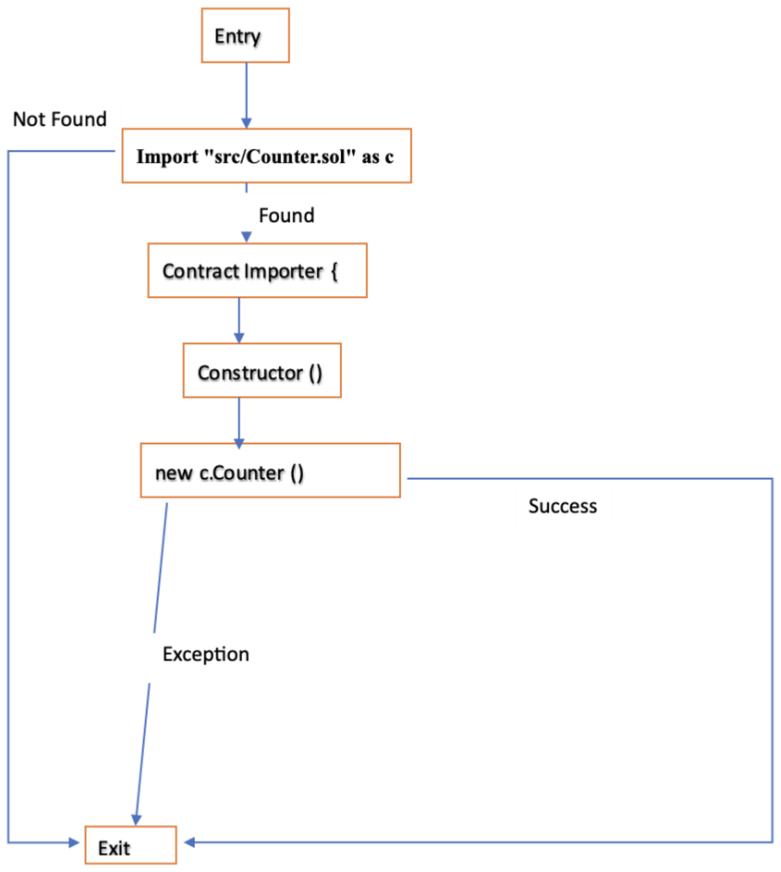
Output: There are two possible outcomes which we have in scope of our analysis

1. Slither analysis runs successfully or gives us possible vulnerabilities in code.

2. An error occurs and error stack trace is printed.

6. Transition Graph

A transition graph represents a program's flow of control. In the below transition graph, the code in "importer.sol" attempts to import counter.sol using an alias, which generates an assertion error. In the absence of the "alias", the contract Importer block defines a Solidity contract named "Importer". In its constructor, a new instance of the Counter contract is created, i.e., new c.Counter(). The code will be executed without any errors if counter.sol does not generate any run-time errors.



7. Test Cases

7.1 **Test Case 1:** Make a new contract called Vulnerable.sol that has a vulnerability in it, such an unchecked call return value problem. Replace the Counter.sol contract in the Importer.sol contract with this Vulnerable.sol contract.

```
// Vulnerable.sol
pragma solidity ^0.8.0;                    //version of solidity compiler

contract Vulnerable {                      //a contract named Vulnerable is defined here
  function transferTokens(address to, uint256 amount) external { //function named
'transferTokens' which have two arguments 'to' & 'amount' is used to make amount
transaction
      (bool success, ) = to.call{value: amount}("");  //unchecked call to 'to' address to
transact mentioned amount
      require(success, "Transfer failed");  //checks whether the transaction is successful
or not
```

```
    }
}

// Importer.sol
pragma solidity ^0.8.13;
import "src/Vulnerable.sol"; //importing vulnerable from src directory

contract Importer { //defines solidity named Importer
    constructor() {    //constructor declaration
        new Vulnerable();
    }
}
```

Explanation

When the Vulnerable.sol contract is imported instead of the expected Counter.sol contract, slither must notice a problem and deliver a warning sign.

7.2 **Test Case 2** ***:*** To check if the Counter contract's increment function is working correctly.

```
pragma solidity ^0.8.13;
import "truffle/Assert.sol";
import "../contracts/Importer.sol";
import "../contracts/Counter.sol";
contract TC1_Counter {
    function testIncrement() public {
        Counter counter = new Counter();
        counter.increment();
        counter.increment();
        counter.increment()
        Assert.equal(counter.number(), 3, "Number not incremented correctly"); } }
```

Explanation

In this test case, we create an instance of the smart contract Counter and call its increment function three times. Using the Assert.equal function, it checks whether the value returned by the "number" function of the "Counter" contract is equal to 3. The test will fail if the value is not equal to 3 and it will print the message "Number not incremented correctly".

7.3 **Test Case 3*:*** To check if the setNumber function of Counter contract is working correctly.

```
pragma solidity ^0.8.13;
import "truffle/Assert.sol";
import "../contracts/Importer.sol";
import "../contracts/Counter.sol";
contract TC0_Counter {
   function testSetNumber() public {
      Counter counter = new Counter();
      counter.setNumber(10);
      Assert.equal(counter.number(), 10, "Number not set correctly");
   }
}
```

Explanation

The setNumber() function of a smart contract called Counter is tested in this Solidity test case. First, it creates an instance of the Counter contract, then it calls the setNumber() function with the value 50. Finally, it checks whether the number() function of the Counter contract returns the expected value of 50. If the returned value does not equal 50, the test case will fail with the message "Number not set correctly".

8. An Overview of the Closer Ties

Slither has several prospective advantages compared to other tools –

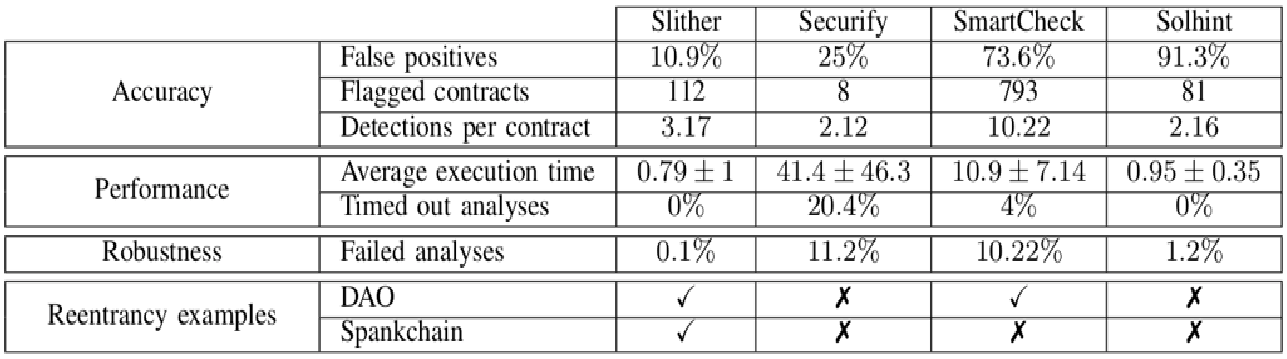
- ❖ Slither's Intermediate Representation (IR) represents Solidity code in a more abstract and simplified manner, making it easier to analyze for security flaws. IRs (Intermediate Representation) used by Slither abstract away some low-level details of Solidity code, and instead focus on high-level structures. In this way, the analysis engine can reason about the code and identify potential security issues more easily. As Slither analyzes Solidity code using an IR, it also does so more quickly and efficiently than some tools that analyze source code directly.

- Slither can detect more types of vulnerabilities, such as reentrancy, floating point precision errors, and uninitialized storage variables, than some other tools.

- Detected issues are explained in a clear and concise manner, with recommendations on how to resolve them. As Slither analyzes Solidity smart contract code, it provides detailed information about any potential security vulnerabilities that it detects, including the type of vulnerability, where it occurs, and a description of the problem.

- A variety of formats are supported, including JSON, CSV, and plain text. This allows the tool to be integrated with other tools and workflows, such as other analysis tools.

- Slither is the most accurate tool with the lowest false positive rate of 10.9%. A summary of the accuracy results is shown in the attached image, which includes False positives, Flagged contracts, and Detections per contract rows.



		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]

9. Risk Management

9.1 Biggest risks:

1) Not able to achieve the iteration goals on time can be the biggest risk.

2) The tool may not run smoothly on all operating systems. This may result in delay of the work.

3) One of the risks is the possibility of unknown issues arising while solving the current issue. This can make it more complex to solve.

4) Not having sufficient knowledge of the specified tool can be biggest risk and could consume more time than expected.

5) There is always a chance for emergency for any member of the team. This can be a risk to the team as the workload increases on the other members in absence of any team member.

9.2 Impacts of the risks:

Sl no	**Description**	**Risk Exposure**
Risk 1:	Delay in completion of one iteration can have impact on the next iteration, which delays overall project.	P= 20% E=20; R.E= 4 hours
Risk 2:	Risks like this can impact because it becomes it consumes a lot of time in setting up virtual machine with compatible OS	P= 40% E=20; R.E= 8 hours
Risk 3:	These kinds of risks have a high impact as unknown error makes the problem solving more complex.	P= 20% E=40; R.E= 8 hours

Risk 4:	Having limited knowledge requires the team to spend additional hours to gain knowledge on the issue itself.	P= 40% E=40, R.E= 16 hours
Risk 5:	Team members absence does impact the progress and productivity	P= 5% E=10; R.E= 0.5 hours

Where; R.E is Risk Exposure and P is probability of the risk occurring.

9.3 Plans to mitigate the risks:

Risk 1: To deal with such cases is to have regular meetings on the work we have done.

Risk 2: To handle such risks is to set up a virtual machine on the operating system that can support the tool.

Risk 3: To mitigate risks from unknown errors we need to keep the tools up to date and follow best practices.

Risk 4: To avoid this risk we need to work extra hours to understand the tool. Seeking help from online forums and product documents.

Risk 5: In such emergency cases, the team needs to divide the work among the remaining teammates so that it does not affect the team's efficiency.

10. Actual value to users/customers

Because of the "Support for imports with aliases" feature for Slither, users are more likely to analyze Solidity code containing imports with aliases more accurately and efficiently. Due to a lack of accurate tracking of symbols imported using an alias, Slither could miss vulnerabilities or weaknesses in code.

Upon successful implementation of this feature, Slither can easily track aliases and better analyze results. Thus, smart contracts can be made more secure and exploitable vulnerabilities can be reduced in production. Furthermore, this feature can help automate the analysis of complex code structures to streamline the analysis process and save developers time and effort. A feature like this is particularly useful when analyzing large, complex Solidity projects with multiple imported symbols or units. Slither's "import with alias" feature improves accuracy, speed, and organization, making it more usable and effective for smart contract developers and auditors.

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