CSE 6324 - 004 - Advance Topic in Software Engineering

Project Title: Smart Contract Analysis tool - Slither and its defect fixing

Team 5

Team Members -

1. Shubham Arun Malankar (1002031033)
2. Nageshwar Ramkumar Jaiswal (1002033432)
3. Ravi Prakasha (1002026832)
4. Navyashree Budhihal Mutt (1001965572)
5. Rushikesh Mahesh Bhagat (1001911486)

Project Plan:

Our team's initial objective was to comprehend how Slither works and understand the scope for optimization within slither. We figured out the scope for optimization with the help of one of the open issues available on GitHub related to a similar variable detector. Furthermore, we used performance measuring tools like Snakeviz to accurately calculate the execution time to check the differences between them using execution time as a parameter. The tool also gave us a graphical representation of the total runtime and the measurements. Our primary aim was to gain a thorough understanding of Slither's workings to pinpoint the precise code or file that required optimization. We utilized performance measuring tools to calculate the execution time and precisely quantify any discrepancies. [2] [6]

Our second objective was to improve the performance of the Slither codebase by optimizing the detector responsible for identifying similar variable vulnerabilities. To achieve this goal, we explored several similar libraries related to difflib such as Levenshtein, Jellyfish, Fuzzywuzzy, and RapidFuzz, and assessed their impact on runtime. Additionally, we restructured the code to decrease its time complexity. We used iterator tools which is an inbuilt library within Python to reduce the time complexity as a replacement for the for loop. [9] [8] [5]

Our final goal is to obtain Smart Contracts that are compatible with Slither. The team aimed to procure Smart Contracts that could be analyzed using Slither as part of their third feature plan. The team struggled to find suitable smart contracts for this purpose. Eventually, we came across a repository on GitHub that contained practical contracts of adequate size to assess the detector's efficacy. [7]

Specification and design:

Inputs:

The following command consisting of input file and few stated parameters:

python -m cProfile -o slithe_results -s tottime -m slither GovernorSettings.sol

- Above stated parameters are elaborated as follows

python	This command is used to run the interpreter for python
-m	This tells Python to run a module as a script.
cProfile	The profiling of the code is done by cProfile for Python.
-o	This is used to specify the output file or destination for a process's results.
Slithe_results	The output file name where the result of the profiling is saved
-s tottime	This is used to infrom cProfile to order the profiling results according to the overall amount of time spent on each function.
-m slither GovernorSettings.sol	This is used to run the solidity file along with slither module.

- The input to the Slither tool is the GovernorMock.sol solidity file which imports variables from GovernorProposalThreshold.sol, GovernorSettings.sol, GovernorCountingSimple.sol and GovernorVotesQuorumFraction.sol into the smart contract. [7]
- Overall, the above command will run the slither module with the specified arguments.

Output:

Before running the command, **first we need to have the repository cloned from** https://github.com/trailofbits/slither into a folder need to have all the below python modules and packages installed under slither folder [3] :

1. Python version - 3.9.7 [11]
2. Cbor2 - 5.4.6
3. crytic-compile - 0.3.0 [3]
4. Jellyfish - 0.9.0 [5]
5. Packaging - 23.0
6. Prettytable - 3.6.0
7. Pycryptodome - 3.17
8. slither-analyzer==0.9.2 [3]
9. snakeviz==2.1.1 [6]
10. solc-select==1.0.3 [3]
11. tornado==6.2
12. wcwidth==0.2.6

After the execution of the above command, it generates profiling results sorted by the total time taken in each function, which will be saved in the slithe_results file as shown in below screenshots.

The below screenshot shows the overall runtime of the functions present in similar_variable.py with the help of snakewiz in a graphical manner.

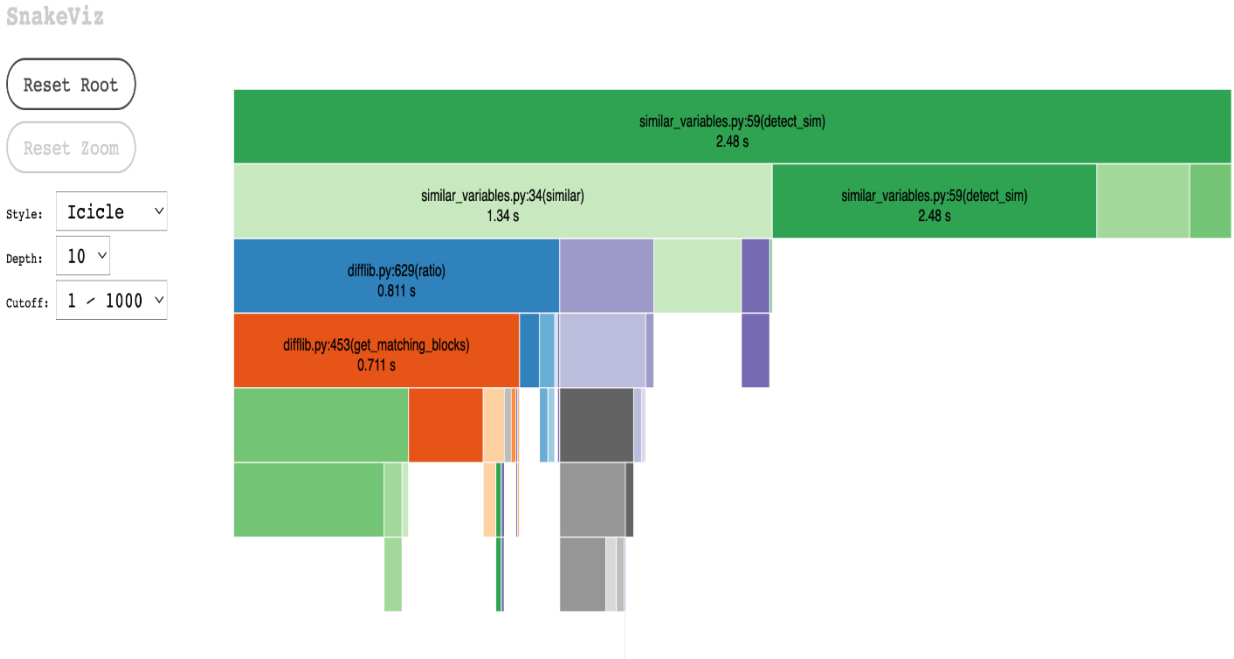

Figure.1 Output before optimizing similar_variable.py [6]

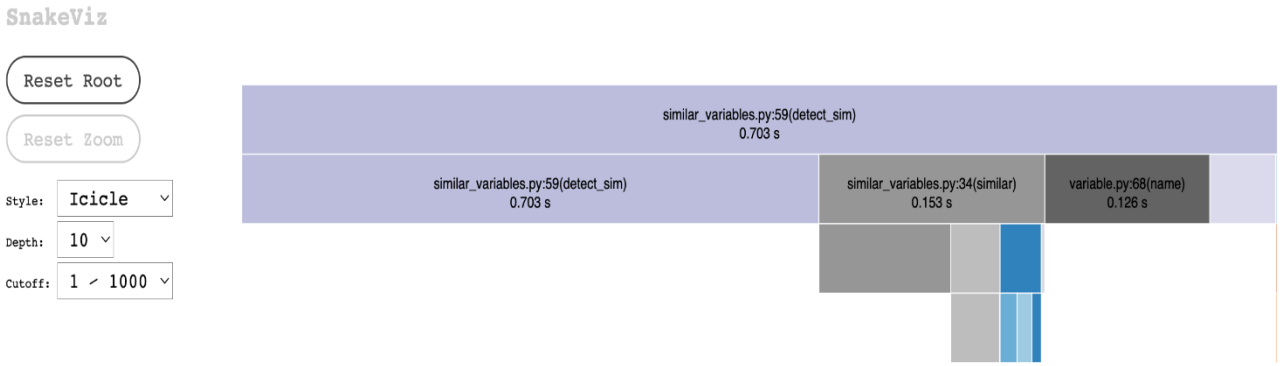

Figure. 2 Output after optimizing similar_variable.py [6]

The below screenshots show the overall runtime of the complete solidity file.

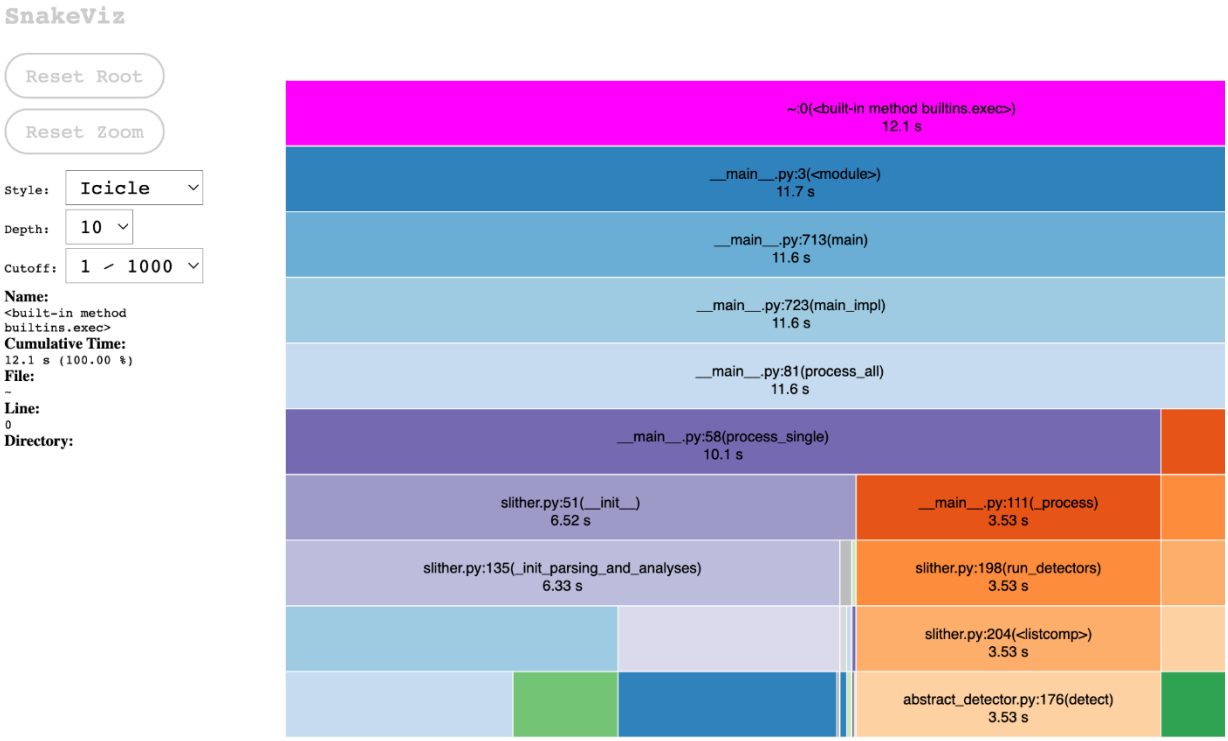

Figure. 3 Output of overall runtime before optimizing [6]

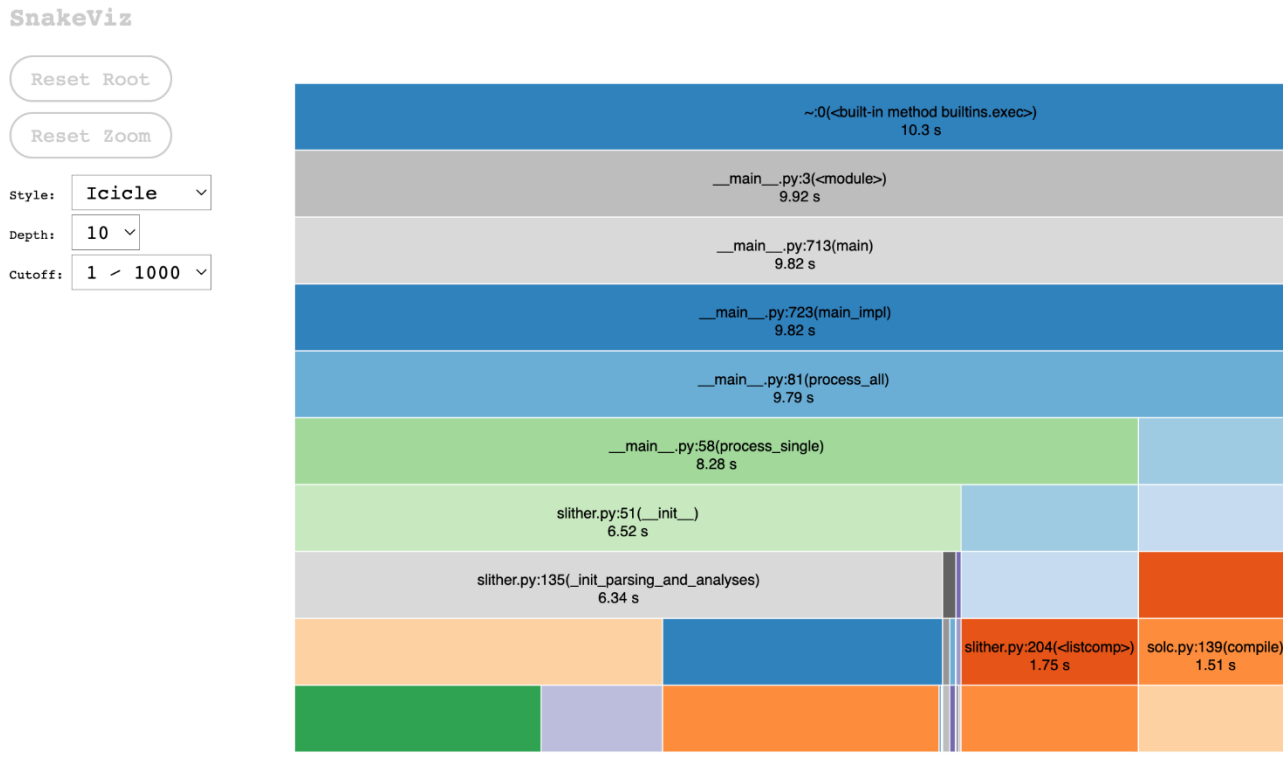

Figure. 4 Output of overall runtime after optimizing [6]

Data structures used:

- Set: All variables found inside contracts are appended inside set data structure.
- List: Lists were used to store variables found inside solidity functions and contracts and after detection of similar variables.

Implemented Feature:

- The similar variable vulnerability used difflib sequence matcher to compare the similarity variables within the variables. The existing method used by slither is a ratio that calculates the distance between two strings by computing the longest common subsequence (LCS) between the two variables by providing a similarity score between 0 and 1.[1] We also compared the other methods provided by Sequence Matcher real_quick_ratio and quick_ratio but they did not show a significant difference in the execution time.
- To reduce the overall runtime, we used Jellyfish, a python library which Jaro-Winkler distance to find the similarity between the two variables. The Jaro-Winkler measures the similarity using the number of similar and the position of the similar characters in the variables.[5] Furthermore, we used itertools a python library as an alternative for for-loop which reduced the time complexity from exponential to linear in detect_sim method within similar_variable.py file which is responsible for finding similar variables within the contact. As an alternative to the for-loop,

itertools is a Python module that changed the time complexity from exponential to linear. [10]

Code and Tests:

The code changes done as part of this iteration are available through our repository:

Repository link - https://github.com/shubhammalankar/ASE-CSE-6324-Team-5-Slither

Slither code changes performed:

- In the file similar_variables.py, Jellyfish, a python library is utilized in order to achieve better execution time. [5]
- The jellyfish utilizes Jaro_Winkler function to find the similarity between the variables.
- As an alternative to the for-loop, itertools a Python module is used to change the time complexity from exponential to linear. [10]

```
@staticmethod
def similar(seq1, seq2):
    """Test the name similarity
    Two name are similar if difflib.SequenceMatcher on the lowercase
    version of the name is greater than 0.90
    See: https://docs.python.org/2/library/difflib.html
    Args:
        seq1 (str): first name
        seq2 (str): second name
    Returns:
        bool: true if names are similar
    """
    #check if length of both variables.
    if len(seq1) != len(seq2):
        return False
    #convert variable into lowercase.
    a=seq1.lower()
    b=seq2.lower()
    #Jaro-Winkler measures the similarity using the number of
    #similar and the position of the similar characters in
    #the variables
    val = jellyfish.jaro_winkler(a,b)
    #comparing the similartiy ratio with a constant
    #value 0.90 if greater than 0.90 return True
    ret = val > 0.90
    return ret # retrun the boolean value
```

Figure. 5 Jellyfish(jaro_winkler) [5]

```python
@staticmethod
def detect_sim(contract):
    """Detect variables with similar name
    Returns:
        bool: true if variables have similar name
    """
    #extract all variables from contracts functions.
    all_var = [x.variables for x in contract.functions]
    all_var = [x for l in all_var for x in l]
    #extract all global variables from contacts.
    contract_var = contract.variables
    #append all variables and typecast into set so that it removes duplicate.
    all_var = set(all_var + contract_var)
    ret = []
    # Generates all possible combinations of given iterable list of a given length
    for v1, v2 in itertools.combinations(all_var, 2):
        if v1.name.lower() != v2.name.lower(): # check if variables are not same
                #call similar function which finds similar variables.
                if SimilarVarsDetection.similar(v1.name, v2.name):
                    if (v2, v1) not in ret:
                        ret.append((v1, v2)) #append the variables to the result list i.e ret
    return set(ret) #remove the duplicates by typecasting list to set
```

Figure. 6 itertools [10]

```solidity
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "../../governance/extensions/GovernorProposalThreshold.sol";
import "../../governance/extensions/GovernorSettings.sol";
import "../../governance/extensions/GovernorCountingSimple.sol";
import "../../governance/extensions/GovernorVotesQuorumFraction.sol";

abstract contract GovernorMock is
    GovernorProposalThreshold,
    GovernorSettings,
    GovernorVotesQuorumFraction,
    GovernorCountingSimple
{
    function proposalThreshold() public view override(Governor, GovernorSettings)
    returns (uint256) {
        return super.proposalThreshold();
    }

    function propose(
        address[] memory targets,
        uint256[] memory values,
        bytes[] memory calldatas,
        string memory description
    ) public override(Governor, GovernorProposalThreshold) returns (uint256) {
        return super.propose(targets, values, calldatas, description);
    }
}
```

Figure. 7 Test Contract (GovernorMock.sol) [7]

The steps required to run and test the code would be as follows:

1. Please install the latest version of Python by following the steps from https://www.python.org/downloads/ . [11]
2. Install the git by using the following https://git-scm.com/download/. [12]
3. Next, clone the repository https://github.com/trailofbits/slither and extract the file folder by using the following command. [13]
4. Create a virtual environment inside the folder where the slither code resides by following the below command in the terminal
   python3 -m venv env [14]
5. To activate virtual environment in the folder:
   On Windows: .\env\Scripts\activate.bat [14]
   On macOS/Linux: source env/bin/activate [14]
6. Install the following dependencies using the pip command from the terminal.
   - Python version - 3.9.7 [11]
   - Cbor2 - 5.4.6
   - crytic-compile - 0.3.0 [3]
   - Jellyfish - 0.9.0 [5]
   - Packaging - 23.0
   - Prettytable - 3.6.0
   - Pycryptodome - 3.17
   - slither-analyzer==0.9.2 [3]
   - snakeviz==2.1.1 [6]
   - solc-select==1.0.3 [3]
   - tornado==6.2
   - wcwidth==0.2.6
7. After installing the dependencies navigate to the slither folder inside the folder from the terminal
   On Windows: cd slither
   On macOS/Linux: cd/slither
8. Run the below command
   python -m cProfile -o slithe_results -s tottime -m slither GovernorSettings.sol .
9. After running the above command then run below the command
   snakeviz slithe_results
10. The graphical results will automatically open in the browser available on the computer

Risk:

Risk 1: Inadequate Documentation for Slither

Document of slither is unclear, because of which it is difficult to understand how to use the tool effectively. This can lead to errors or incorrect usage of the tool and resulting in inaccurate results. There is a chance of misinterpretation of the result. [3]

Risk Exposure: When the documentation for a tool is inadequate, it increases the risk of security vulnerabilities going undetected. If Slither users do not have access to clear and comprehensive documentation, they may miss critical information that could help them identify and address potential security risks in their smart contracts.

Mitigation: Since changing Slither documentation is not our goal, we looked at other resources like StackOverflow and other GitHub issues which were having same issues.

Risk 2: Slither producing different results on Mac and Windows Operating Systems

While running slither tools on different Operating Systems like macOs and Windows, it produced subtle differences in the output produced and impact because of differences in the compiler and computing power. This could happen, due to different file systems and the limitations of each Operating System.

Risk Exposure: If Slither produces different results on Mac and Windows operating systems, it could lead to inaccurate analysis of smart contracts in real time. This could result in potential security risks going undetected, leading to vulnerabilities that could be exploited by attackers. This could lead to reduced reliability and misleading results.

Mitigation: To minimize the impact of different system environments on Slither's analysis, it can be standardizing the environment and use cross-platform, in which they run the tool. This could involve using a consistent set of system configurations, such as the same version of the Solidity compiler and running the slither on docker, to ensure that the analysis is consistent across different operating systems.

Risk 3: Searching for Smart Contracts that are compatible

Smart contracts are easily available on the internet but finding a smart contract specific to our issue is arduous. They were not complex enough to resolve our vulnerability.

Risk Exposure - It was time consuming to design them after studying different issue.

Risk Mitigation - After extensive research, we were able to locate a repository containing real-time contracts of sufficient size to evaluate the detector's performance. [7]

Risk 4: Loss of Functionality while optimizing the detector

Use of different libraries affects the number of vulnerabilities detected. It can reduce the efficiency of existing detector. [8]

Risk Exposure – Caused extra variables while testing for smart contract.

Mitigation – Used one library to test various bugs and resolve them in smart contract.

Customers and Users:

- We reached out to the publisher (customer), who has posted an issue in Github: To gain deep understanding about the issue, what exactly publisher has faced problem, when and where it occurred, we contacted the publisher via email. They suggested the probability of fixing our issue and asked them to suggest smart contracts which have similar variable issues.
- We have tested new changes on smart contracts which are related to governance jobs. Given test contract is counting votes of members present inside parliament while passing a proposal. [7]

References:

[1] https://stackoverflow.com/questions/50487058/python-difflibs-ratio-quick-ratio-and-real-quick-ratio

[2] https://github.com/crytic/slither/issues/1630

[3] https://github.com/crytic/slither/wiki/Developer-installation

[4] https://icons8.com/icons

[5] https://pypi.org/project/jellyfish/

[6] https://jiffyclub.github.io/snakeviz/

[7] https://github.com/OpenZeppelin/openzeppelin-contracts

[8] https://stackoverflow.com/questions/682367/good-python-modules-for-fuzzy-string-comparison

[9] https://pypi.org/project/rapidfuzz/

[10] https://docs.python.org/3/library/itertools.html

[11] https://www.python.org/

[12] https://git-scm.com/downloads

[13] https://github.com/crytic/slither

[14] https://realpython.com/python-virtual-environments-a-primer/