THE UNIVERSITY OF TEXAS AT ARLINGTON

# Advanced Topics in Software Engineering

## CSE 6324 – Section 001

## Team 9

## Final Iteration

## (Written Deliverable)

**Abhishek Wadhwani – 10020352719**

**Mounika Kottapalli– 1002085510**

**Nitin Raj Thumma– 1002080555**

**Sai Raghu Rami Reddy Dontireddy – 1002014523**

**<u>TABLE OF CONTENT</u>**

## I. Project Proposal:

The proposal aims to enhance the Slither Framework Analysis by incorporating functionality to handle nested structs. Conducting a thorough examination of Solidity language specifications enables a focused exploration of how nested structs are structured and utilized. This examination facilitates the development of a profound understanding of the logic and patterns associated with nested structures within Solidity contracts. [1][2]

To achieve this, it is imperative to establish connections between the detection algorithm and Slither's existing architecture. Formulating a detection algorithm capable of efficiently parsing through the AST involves analyzing variable declarations to identify instances of nested structs. This enhancement would fortify Slither's effectiveness, augmenting its thoroughness and precision in evaluating Solidity code.

During the final iteration, when implementing a detector for nested structs, two issues were encountered:

1. The "slither-read-storage" program was unable to find a slot for the struct variable. This issue stemmed from the fact that the "slither.core.solidity_types.user_defined_type.UserDefinedType" class was not implemented in the "find_struct_var_slot" function. Changes were made to the "read_storage.py" file, resolving this error.

2. The second issue involved an inability to find the slots of the nested structs. Even though the code executed without errors, the result indicated "person" with information about the person and a slot count of 0 in the mapping. However, no information was provided for the nested variables, and the count was not accurate.

These challenges highlight the importance of addressing the issues with the "slither-read-storage" program and refining the handling of nested structs to ensure accurate and comprehensive analysis of Solidity smart contracts within the Slither framework.

GitHub Issue - https://github.com/crytic/slither/issues/2077

## II. Competitors:

**Detector to detect nested structured slots in Slither:**

There is no competitor detector in Slither to detect nested structured slots. The issue required an update to calculate the slot for UserDefinedType.

Although there is no direct competitor, we sought guidance from another user (dokzai) who attempted to address a similar issue. His code is designed to take the elements of a struct, the starting slot for the search, and the struct variable to locate as parameters [12].

The code operates by recursively traversing the struct's elements, starting from the designated slot. If it finds the specified struct variable, it returns the corresponding slot number; otherwise, it returns None. Unfortunately, this approach did not lead to a resolution for the current issue. We used this code for experimental purposes, hoping to build upon it and find a solution.

## III. Project Plan:

| **S.no** | **Task Description** | **Status** |
|---|---|---|
| **1.** | Installation<br>a) Python v3.6+<br>b) Hardhat<br>c) Ubuntu OS<br>d) Slither | Completed |
| **2.** | Decide the detector to start working on | Completed |
| **3.** | a) Find the sample solidity contract to analyze it using Slither<br>b) Work on reviews from Inception | Completed |
| **4.** | a) Integrate Slither and Smart Contract with Hardhat.<br>b) Codebase and Folder Structure Analysis. | Completed |

| | | |
|---|---|---|
| **5.** | a) Load the solidity smart contract.<br><br>b) Set up the solidity compiler version based on the sample solidity smart contract.<br><br>c) Work on reviews from Iteration 1 | Completed |
| **6.** | a) Create a .sol solidity file.<br><br>b) Installing dotenv<br><br>c) Create dotenv file<br><br>d) Contract deployment | Completed |
| **7.** | a) Write a detector to detect vulnerability.<br><br>b) Analyze the solidity contract with nested structs. | Completed |
| **8.** | a) Development of the detector<br><br>b) Work on review from Iteration 2 | Completed |
| **9.** | a) Improvising the detector by fixing errors<br><br>b) Testing and analysis<br><br>c) Generate reports and documentation for Final Iteration | Incomplete<br><br>Incomplete<br><br>Completed |

## IV. <u>Risk factors and Mitigation plan:</u>

| **Risk Factor** | **Mitigation Plan** | **Risk Exposure** |
|---|---|---|
| Maintaining and compatibility | Sticking with a fixed version of solidity in the detector. | Trying to maintain the same version and implement the code trying to make it compatible. |
| Thorough Testing and Validation | Create a detailed list of tests for the nested structs. | Incomplete. |
| Technical issue - Inexperience with Python | Learning python via tutorials. | With minimal knowledge of programming Language, it is difficult to work on the code. |
| Technical issue Inexperience with Solidity | Learning solidity language concepts by tutorials. | With no knowledge of Solidity, it is very difficult to work on Nested Struct and it takes time to get the hand of it. |
| Technical issue Complexity Building the Nested Struct and using in a Function | Finding out how to build nested structs and write them in the smart contracts | Analyzing existing solidity open-source code and going through Solidity Documentation and relevant work. |
| Not receiving the anticipated output | Rechecking and writing the code and find where it went wrong | Analyzing the complete code and trying to enhance the code to get a better outcome. |

## V. Specification and Design:

Tools such as Hardhat, Python, Pip3, and Slither, establishing a solid foundation were installed for this project. We deployed a Smart Contract on the Polygon Mumbai blockchain and worked to enhance Slither's functionality for analyzing contracts that incorporate nested structs.

Installation on Ubuntu OS:

- Hardhat[3]

```
npm install -save-dev hardhat
```

```
npx hardhat init
npm install --save-dev @nomicfoundation/hardhat-toolbox@^3.0.0
```

- Python

```
sudo apt update
 sudo apt install python3
sudo apt-get -y install python3-pip
```

- Slither [4]

```
pip3 install slither-analyzer
```

### A. Figuring out blockchains onto which we can deploy on Smart Contract

The Slither tool can only analyze storage slots in smart contracts that are deployed on a blockchain. We chose Polygon Mumbai because it's efficient, scalable, and friendly to developers. Additionally, it provides a great environment for testing and refining smart contracts without incurring high transaction costs (gas fees).

### B. Nested Struct Smart Contracts

There are 2 structs in it which are home address and person.

**Struct Definitions:**

- *HomeAddress* is a struct representing a person's home address with fields for street, city, and country. [5]
- *Person* is another struct, but it's currently empty. It should include fields for the person's name, age, and *HomeAddress*.

**State Variables:**

- name and age are state variables of the contract to store the name and age of the contract creator.
- *location* is an instance of the *HomeAddress* struct to store the creator's home address.

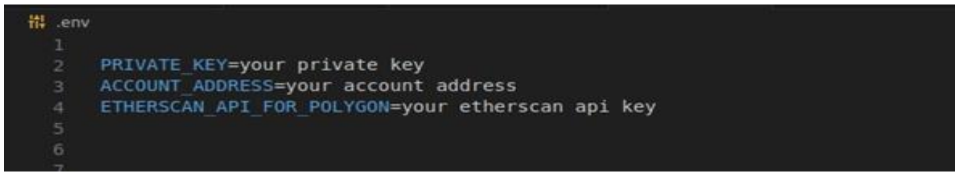
- people is a mapping that associates Ethereum addresses with Person structs.

**Addperson function:**

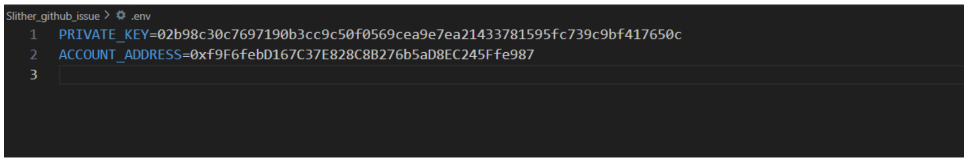
- With the help of parameters, it sets the new properties.
- It creates a *homeaddress* struct *addr* with the provided street, city, and country.
- It then tries to create a person struct *newperson* using name, age and *addr*.

**C. Installing and creating dotenv file:**

dotenv is used to store sensitive or configuration-related information separately from the code by using environment variables. These variables are then accessed within the code to provide security and flexibility.



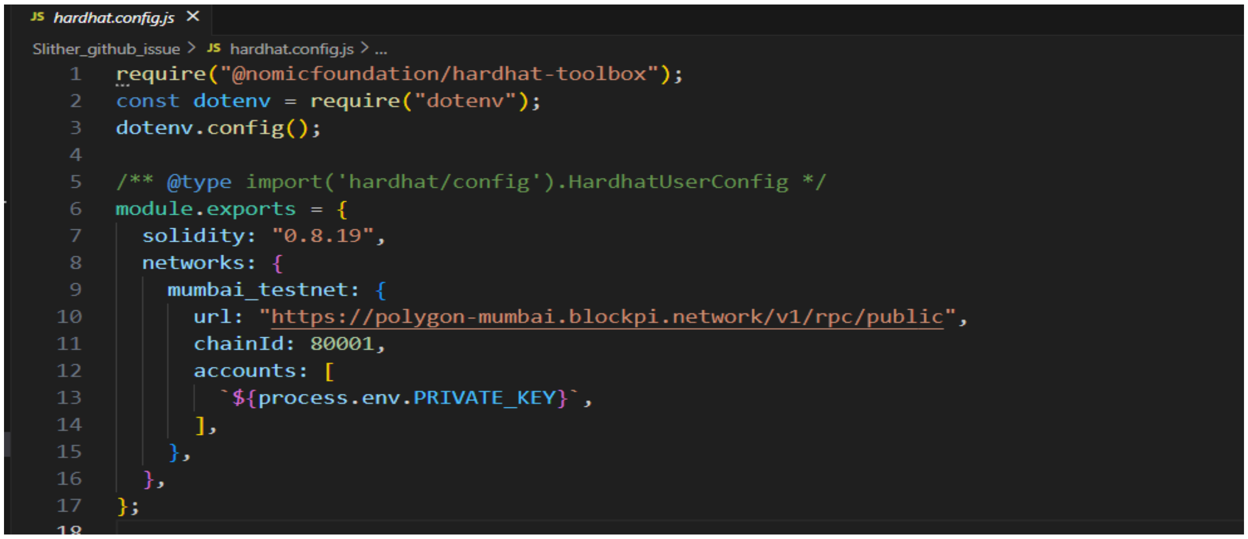
```
.env

PRIVATE_KEY=your private key
ACCOUNT_ADDRESS=your account address
ETHERSCAN_API_FOR_POLYGON=your etherscan api key


```



```
Slither_github_issue > .env
PRIVATE_KEY=02b98c30c7697190b3cc9c50f0569cea9e7ea21433781595fc739c9bf417650c
ACCOUNT_ADDRESS=0xf9F6febD167C37E828C8B276b5aD8EC245Ffe987

```

**D. Deployment scripts for Hardhat tool.**

• Install the Hardhat Toolbox.
• Import the dotenv Package.
• Load the Environment Variables from the. env File into the Current Process.
• Specify the RPC URL of the Mumbai Testnet.
• Specify the chainId Field (Chain ID) of the Mumbai Testnet.
• Specify the Private Keys of the Account for Deployment and Testing.


hardhat.config.js ✕

Slither_github_issue > hardhat.config.js > ...

```
require("@nomicfoundation/hardhat-toolbox");
const dotenv = require("dotenv");
dotenv.config();

/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  solidity: "0.8.19",
  networks: {
    mumbai_testnet: {
      url: "https://polygon-mumbai.blockpi.network/v1/rpc/public",
      chainId: 80001,
      accounts: [
        `${process.env.PRIVATE_KEY}`,
      ],
    },
  },
};
```

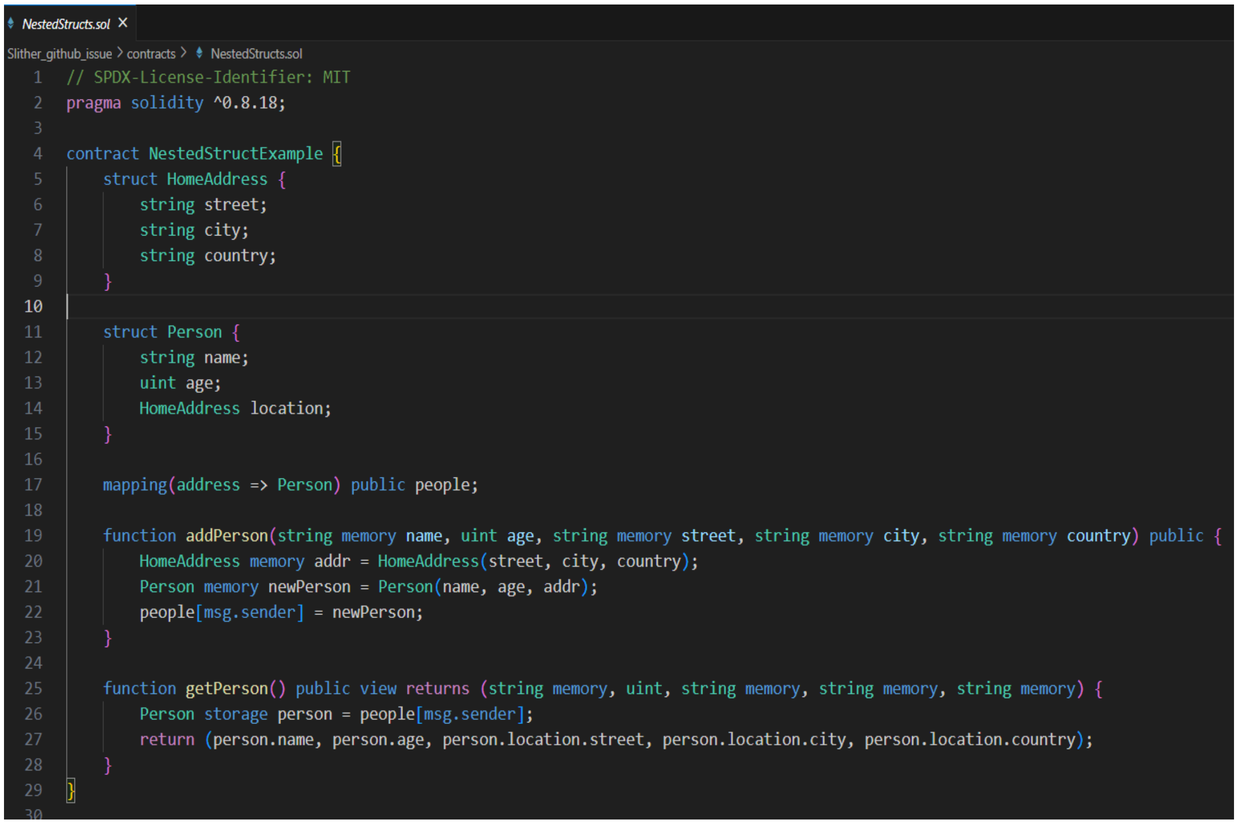
## E. Deploying the contract on Polygon blockchain

The smart contract is deployed on polygon mumbai testnet with an address of 0xB091F9c43b4D58C6040F65841E4DF8Be27726BC2

https://mumbai.polygonscan.com/address/0xB091F9c43b4D58C6040F65841E4DF8Be27726BC2

## F. Input and Output for the detector:

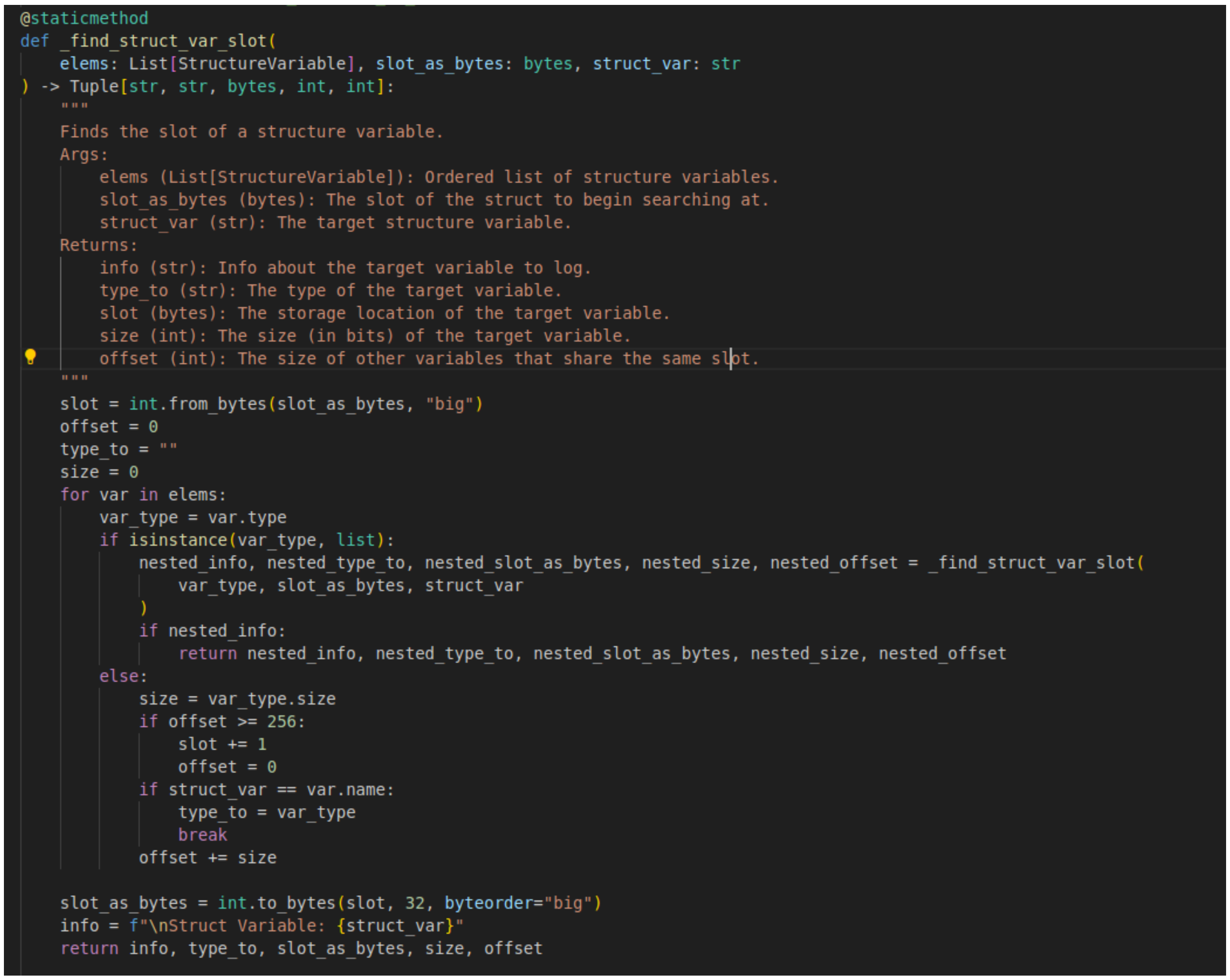
This Solidity smart contract defines a struct called Person to store the information of a person, including their name, age, and address. It also defines a mapping called people to map Ethereum addresses to Person structs. The contract provides two functions: addPerson() and getPerson(). The addPerson() function allows users to add a new person to the people mapping. The getPerson() function allows users to get the information of the person who is calling the function.


NestedStructs.sol ×

Slither_github_issue > contracts > NestedStructs.sol

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;

contract NestedStructExample {
    struct HomeAddress {
        string street;
        string city;
        string country;
    }

    struct Person {
        string name;
        uint age;
        HomeAddress location;
    }

    mapping(address => Person) public people;

    function addPerson(string memory name, uint age, string memory street, string memory city, string memory country) public {
        HomeAddress memory addr = HomeAddress(street, city, country);
        Person memory newPerson = Person(name, age, addr);
        people[msg.sender] = newPerson;
    }

    function getPerson() public view returns (string memory, uint, string memory, string memory, string memory) {
        Person storage person = people[msg.sender];
        return (person.name, person.age, person.location.street, person.location.city, person.location.country);
    }
}
```

## G. About the Detector



```
@staticmethod
def _find_struct_var_slot(
    elems: List[StructureVariable], slot_as_bytes: bytes, struct_var: str
) -> Tuple[str, str, bytes, int, int]:
    """
    Finds the slot of a structure variable.
    Args:
        elems (List[StructureVariable]): Ordered list of structure variables.
        slot_as_bytes (bytes): The slot of the struct to begin searching at.
        struct_var (str): The target structure variable.
    Returns:
        info (str): Info about the target variable to log.
        type_to (str): The type of the target variable.
        slot (bytes): The storage location of the target variable.
        size (int): The size (in bits) of the target variable.
        offset (int): The size of other variables that share the same slot.
    """
    slot = int.from_bytes(slot_as_bytes, "big")
    offset = 0
    type_to = ""
    size = 0
    for var in elems:
        var_type = var.type
        if isinstance(var_type, list):
            nested_info, nested_type_to, nested_slot_as_bytes, nested_size, nested_offset = _find_struct_var_slot(
                var_type, slot_as_bytes, struct_var
            )
            if nested_info:
                return nested_info, nested_type_to, nested_slot_as_bytes, nested_size, nested_offset
        else:
            size = var_type.size
            if offset >= 256:
                slot += 1
                offset = 0
            if struct_var == var.name:
                type_to = var_type
                break
            offset += size

    slot_as_bytes = int.to_bytes(slot, 32, byteorder="big")
    info = f"\nStruct Variable: {struct_var}"
    return info, type_to, slot_as_bytes, size, offset
```

**1. Parameters**:

*elems*: List of StructureVariable objects representing ordered structure variables.
*slot_as_bytes*: Bytes representing the initial storage slot to start the search.
*struct_var*: The target structure variable to search for within the list.

**2. Return Values:**

*info*: Information about the target variable to log.
*type_to*: The type of the target variable.
*slot*: Bytes representing the storage location of the target variable.
*size*: The size (in bits) of the target variable.
*offset*: The size of other variables that share the same slot.

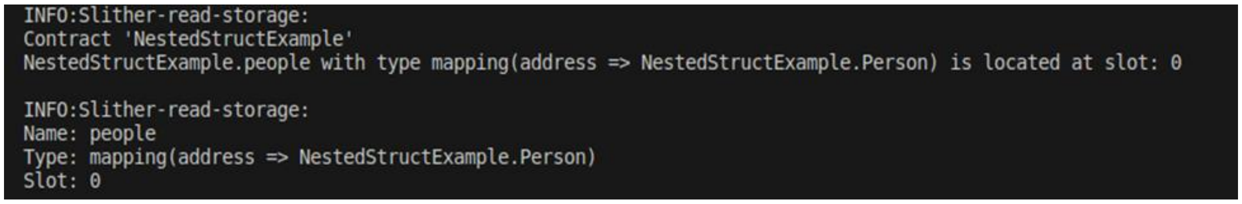
The function begins by initializing several variables: slot, offset, type_to, and size. It iterates through each StructureVariable in the provided list *elems*. For each variable, it checks if the

*var_type* is a list (indicating a nested structure). If it's a nested structure, it recursively calls the *_find_struct_var_slot* function to search within the nested structure.
If the nested search finds the target variable, it returns the corresponding information. If the *var_type* is not a nested structure, it calculates the size of the variable.

If the accumulated offset exceeds 256 bits (32 bytes), it increments the slot and resets the offset. It checks if the current variable matches the *struct_var*. If it does, it sets the *type_to* and breaks the loop. Otherwise, it increments the offset by the size of the current variable. Finally, it converts the slot into bytes format, constructs information about the target variable, and returns the gathered information (info, type_to, slot, size, and offset).

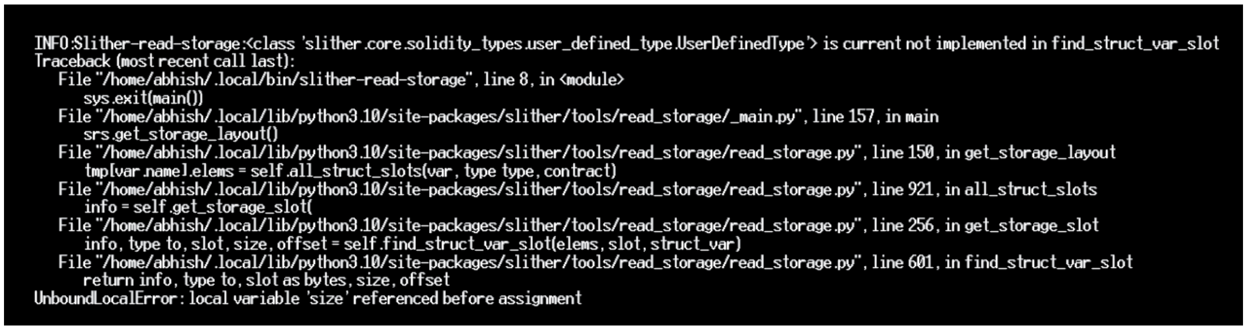
## H. Running the command for the Detector on the Deployed contract.

Need to mention mumbai and then the address as the contract is deployed on Polygon Mumbai Testnet [6]



```
INFO:Slither-read-storage:
Contract 'NestedStructExample'
NestedStructExample.people with type mapping(address => NestedStructExample.Person) is located at slot: 0

INFO:Slither-read-storage:
Name: people
Type: mapping(address => NestedStructExample.Person)
Slot: 0
```

This isn’t throwing any error for nested structs. It shows slot 0 of our contract as can be seen. We are trying to modify the detector so that it gives further details about how the slots for Person and HomeAddress could be managed.



```
INFO:Slither-read-storage:<class 'slither.core.solidity_types.user_defined_type.UserDefinedType'> is current not implemented in find_struct_var_slot
Traceback (most recent call last):
  File "/home/abhish/.local/bin/slither-read-storage", line 8, in <module>
    sys.exit(main())
  File "/home/abhish/.local/lib/python3.10/site-packages/slither/tools/read_storage/_main.py", line 157, in main
    srs.get_storage_layout()
  File "/home/abhish/.local/lib/python3.10/site-packages/slither/tools/read_storage/read_storage.py", line 150, in get_storage_layout
    tmp[var.name].elems = self.all_struct_slots(var, type type, contract)
  File "/home/abhish/.local/lib/python3.10/site-packages/slither/tools/read_storage/read_storage.py", line 921, in all_struct_slots
    info = self.get_storage_slot(
  File "/home/abhish/.local/lib/python3.10/site-packages/slither/tools/read_storage/read_storage.py", line 256, in get_storage_slot
    info, type to, slot, size, offset = self.find_struct_var_slot(elems, slot, struct_var)
  File "/home/abhish/.local/lib/python3.10/site-packages/slither/tools/read_storage/read_storage.py", line 601, in find_struct_var_slot
    return info, type to, slot as bytes, size, offset
UnboundLocalError: local variable 'size' referenced before assignment
```

## I. Errors that have occured:

a. The implementation of the current version of the detector gives us the following error.
This error means that *slither-read-storage* program is not able to find a slot for the struct variable. This is because the *slither.core.solidity_types.user_defined_type.UserDefinedType* class is not currently implemented in the *find_struct_var_slot* function.

b. With changes to the read_storage.py
We managed to get rid of the above error, but we are still not able to find the slots of the nested structs. We still get slots are 0 in the mapping and nothing for the nested variables.

**Conclusion:** Over the past months, our team has made significant strides in advancing our understanding and learning in various aspects of blockchain development and security analysis.

To calculate the number of storage slots for a nested struct, you sum up the storage slots used by each variable within the structs, considering their individual slot sizes and packing them sequentially in storage. Understanding these principles aids in optimizing storage usage, minimizing gas costs, and efficiently organizing data within smart contracts.

In conclusion, despite efforts to address the initial error in the current version of the detector by modifying the read_storage.py file, the project faces a significant challenge with the inability to locate slots for nested structs. Unfortunately, this issue remains unresolved, presenting a roadblock in the completion of the slither project. Further exploration and debugging may be necessary to overcome this obstacle and achieve the desired functionality.

## VI. Target Users/Customers:

- CSE 6324-001 class
- Aditya Vichare (CSE 6324-001) - Project is interesting but incomplete due to errors, requiring further refinement.
- Smart Contract Enthusiast (Amosh Sapkota) - This concept is presented in a way that is easily grasped by beginners.

## VII. References:

1. Slither: https://github.com/crytic/slither
2. Issue: https://github.com/crytic/slither/issues/2077
3. Install Hardhat:hardhat.org/hardhat-runner/docs/getting-started#overview
4. Install Slither: https://github.com/crytic/slither#how-to-install
5. https://www.immunebytes.com/blog/slither-a-solidity-static-analyzer-for-smart-contracts/
6. https://www.alchemy.com/overviews/mumbai-testnet
7. https://hardhat.org/hardhat-runner/docs/guides/deploying
8. https://docs.chainstack.com/reference/polygon-getting-started
9. https://www.npmjs.com/package/dotenv
10. https://docs.arbitrum.io/stylus/how-tos/local-stylus-dev-node
11. https://stackoverflow.com/questions/70011797/return-nested-struct-inside-struct-in-solidity
12. https://github.com/dokzai/slither/commit/f42bbad5f8cc55584a42f514141265f95fecde98

GitHub Repository: https://github.com/Abhismoothie/Slither-Enhancementproject-team-9-CSE6324-001