Advanced Topics in Software Engineering

CSE 6324 – Section 001

Team 9

Iteration 2

(Written Deliverable)

Abhishek Wadhwani – 10020352719
Mounika Kottapalli– 1002085510
Nitin Raj Thumma– 1002080555
Sai Raghu Rami Reddy Dontireddy – 1002014523

TABLE OF CONTENT

I. Abstract:

The Proposal is to enhance the Slither Framework Analysis by adding a functionality to handle nested structs. It provides fine-grained information about smart contract codes and has the necessary flexibility to support many applications.

Enabling support for nested structs in Slither would expand its capabilities, allowing users to analyze a wider array of Solidity smart contracts for potential security vulnerabilities. This enhancement would also bolster Slither's effectiveness by increasing its thoroughness and precision in evaluating Solidity code.

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

II. Project Plan:

#	Task Description	Status
1.	Installation a) Python v3.6+ b) Hardhat c) Ubuntu OS d) Slither	Completed
2.	Decide the detector to start working on	Completed
3.	a) Find the sample solidity contract to analyze it using Slither b) Work on reviews from Inception	Completed
4.	a) Integrate Slither and Smart Contract with Hardhat. b) Codebase and Folder Structure Analysis.	Completed

5.	a) Load the solidity smart contract. b) Set up the solidity compiler version based on the sample solidity smart contract. c) Work on reviews from Iteration 1	Completed
6.	a) Create a .sol solidity file. b) Installing dotenv c) Create dotenv file d) Contract deployment	Completed
7.	a) Write a detector to detect vulnerability. b) Analyze the solidity contract with nested structs.	In Progress
8.	a) Generate reports b) Work on reviews from iteration 2	Incomplete

III. About the Detector:

- Previously, we reviewed Solidity language specifications, improved Slither's architecture, designed a detection algorithm for nested structs, integrated it into Slither, and fine-tuned it for accuracy.

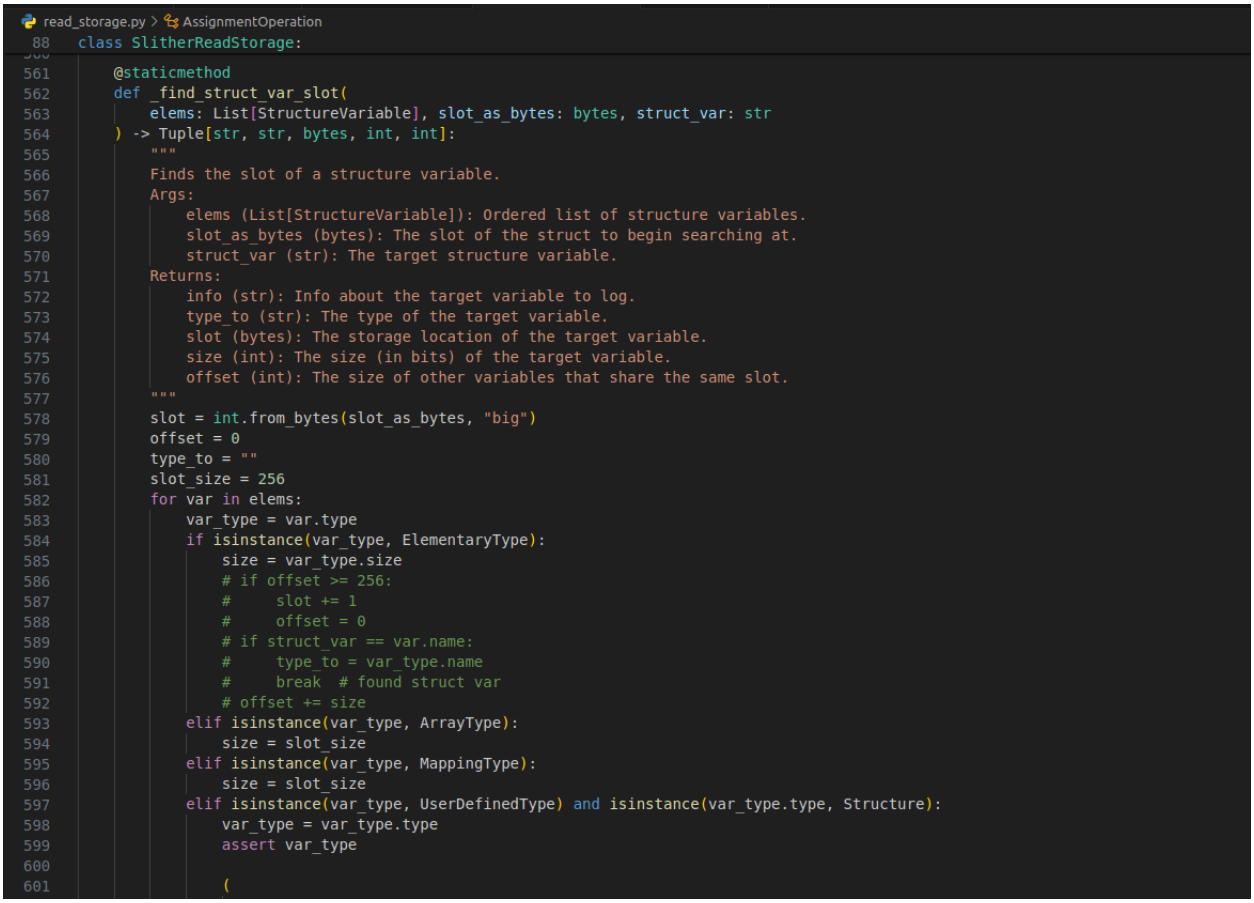
- Our current focus pertains to the "read_storage.py" file within the Slither tool. Specifically, we are addressing the function "_find_struct_var_slot()," which is responsible for retrieving data from nested storage slots.

- At present, our contract incorporates a nested structure, but the function fails to identify the slots for the nested struct. It mistakenly attributes slot 0 to the "Person" struct.

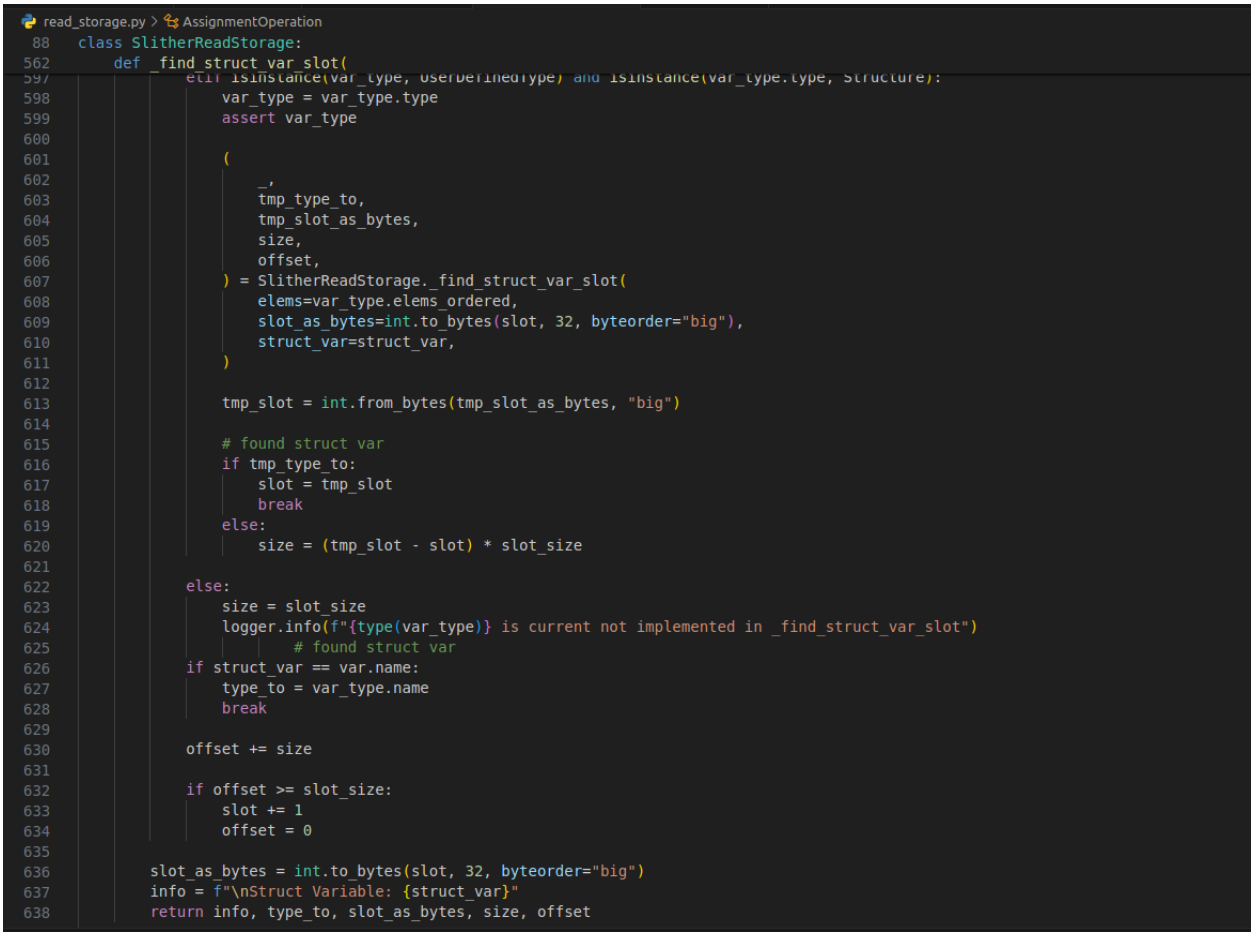
- The following function is pivotal to our tool, and we've adjusted ascertain whether the instance is a mapping or an array type. In such cases, we've assigned a slot size of 256 bits.

However, we are currently working on enhancing this capability for dynamic arrays. We are actively exploring methods to determine the slot size for such dynamic arrays. [12]

- The code for the function in question has been modified to successfully identify the slot, yet we are still working on resolving the issue related to extracting further details from the nested struct. Please note that this is an ongoing work in progress.



```
read_storage.py > AssignmentOperation
 88  class SlitherReadStorage:
561      @staticmethod
562      def _find_struct_var_slot(
563          elems: List[StructureVariable], slot_as_bytes: bytes, struct_var: str
564      ) -> Tuple[str, str, bytes, int, int]:
565          """
566          Finds the slot of a structure variable.
567          Args:
568              elems (List[StructureVariable]): Ordered list of structure variables.
569              slot_as_bytes (bytes): The slot of the struct to begin searching at.
570              struct_var (str): The target structure variable.
571          Returns:
572              info (str): Info about the target variable to log.
573              type_to (str): The type of the target variable.
574              slot (bytes): The storage location of the target variable.
575              size (int): The size (in bits) of the target variable.
576              offset (int): The size of other variables that share the same slot.
577          """
578          slot = int.from_bytes(slot_as_bytes, "big")
579          offset = 0
580          type_to = ""
581          slot_size = 256
582          for var in elems:
583              var_type = var.type
584              if isinstance(var_type, ElementaryType):
585                  size = var_type.size
586                  # if offset >= 256:
587                  #     slot += 1
588                  #     offset = 0
589                  # if struct_var == var.name:
590                  #     type_to = var_type.name
591                  #     break  # found struct var
592                  # offset += size
593              elif isinstance(var_type, ArrayType):
594                  size = slot_size
595              elif isinstance(var_type, MappingType):
596                  size = slot_size
597              elif isinstance(var_type, UserDefinedType) and isinstance(var_type.type, Structure):
598                  var_type = var_type.type
599                  assert var_type
600
601                  (
```



```
read_storage.py > AssignmentOperation
class SlitherReadStorage:
    def _find_struct_var_slot(
            elif isinstance(var_type, UserDefinedType) and isinstance(var_type.type, Structure):
                var_type = var_type.type
                assert var_type

                (
                    _,
                    tmp_type_to,
                    tmp_slot_as_bytes,
                    size,
                    offset,
                ) = SlitherReadStorage._find_struct_var_slot(
                    elems=var_type.elems_ordered,
                    slot_as_bytes=int.to_bytes(slot, 32, byteorder="big"),
                    struct_var=struct_var,
                )

                tmp_slot = int.from_bytes(tmp_slot_as_bytes, "big")

                # found struct var
                if tmp_type_to:
                    slot = tmp_slot
                    break
                else:
                    size = (tmp_slot - slot) * slot_size

            else:
                size = slot_size
                logger.info(f"{type(var_type)} is current not implemented in _find_struct_var_slot")
                    # found struct var
            if struct_var == var.name:
                type_to = var_type.name
                break

            offset += size

            if offset >= slot_size:
                slot += 1
                offset = 0

        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
```

IV. Risk factors and Mitigation plan:

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 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:

During the first iteration of the project, we systematically carried out the installation process on an Ubuntu OS. This involved setting up critical tools, namely Hardhat, Python, and Slither. We ensured the successful installation of Hardhat, Python, Pip3, and Slither, establishing a solid foundation for the following project phases. These subsequent stages include the deployment of a Smart Contract on the Polygon Mumbai blockchain and our ongoing work to enhance Slither's functionality for analyzing contracts that incorporate nested structs.[1][3][4]

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. To do this, we needed to learn how to deploy a smart contract and choose the right blockchain for the and explore various types of blockchains, such as public ones like Ethereum and private ones, each having different features. After researching options like Binance Smart Chain and Arbitrum, we opted for Polygon Mumbai. 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). [11]

B. Nested Structure Smart Contract

contracts > NestedStructs.sol

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

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);
    }
}
```

The above code will add and retrieve the information of a person.

In the following way

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 dotenv.

```
up to date, audited 567 packages in 1s

91 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Many applications require configuration settings such as API keys, database connection strings, security tokens, and other sensitive information. Storing these configurations directly in code can be a security risk and make it challenging to update settings across different environments. [9]

.env files are easy to transport and share between different environments. This makes it simpler to maintain consistent configurations across various stages of development, testing, and deployment.

When working on a project with multiple team members, using .env files allows for easy collaboration. Team members can share the configuration file without sharing sensitive information like API keys or passwords.

When working with Docker containers, you can pass environment variables to containerized applications using .env files. This makes it easier to configure containerized applications with different settings for various environments.

D. 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 with in the code to provide security and flexibility. [9]

.env

```
1
2  PRIVATE_KEY=your private key
3  ACCOUNT_ADDRESS=your account address
4  ETHERSCAN_API_FOR_POLYGON=your etherscan api key
5
6
7
```

E. Writing deployment scripts for Hardhat tool

```
hardhat.config.js > ...
1  require("@nomicfoundation/hardhat-toolbox");
2  const dotenv = require("dotenv");
3  dotenv.config();
4
5  /** @type import('hardhat/config').HardhatUserConfig */
6  module.exports = {
7    solidity: "0.8.19",
8    networks: {
9      mumbai_testnet: {
10       url: "https://polygon-mumbai.blockpi.network/v1/rpc/public",
11       chainId: 80001,
12       accounts: [
13         `${process.env.PRIVATE_KEY}`,
14       ],
15     },
16   },
17 };
18
```

- Install the Hardhat Toolbox. Installing the Hardhat toolbox that extends the functionality of Hardhat, a development environment for Ethereum smart contracts.

- Import the dotenv Package. The dotenv package is imported to enable the usage of environment variables stored in a .env file.[9]

- Load the Environment Variables from the .env File into the Current Process. The dotenv.config() method is called to load the environment variables from the .env file into the current Node.js process. This allows the variables specified in the .env file to be accessed using process.env.VARIABLE_NAME syntax throughout the application.[10]

- Specify the RPC URL of the Mumbai Testnet. The code specifies the Remote Procedure Call (RPC) URL of the Polygon Mumbai Testnet. In our project , we used the Mumbai Testnet's RPC URL for communication.[6]

- Specify the chainId Field (Chain ID) of the Mumbai Testnet. The chainId field specifies the unique identifier (chain ID) of the Polygon Mumbai Testnet.
- Specify the Private Keys of the Account for Deployment and Testing. The accounts field specifies an array containing the private keys of the Ethereum accounts that Hardhat should use for deploying and testing contracts on the Mumbai Testnet.

F. Command for deploying the contract on polygon blockchain

```
-IdeaPad-5-15ITL05-Ua:~/Slither_github_issue$ npx hardhat run scripts/deploy.js --network mumbai_testnet
```

```
NestedStructExample deployed to: 0xB091F9c43b4D58C6040F65841E4DF8Be27726BC2
```

The smart contract is deployed on polygon mumbai testnet with an address of 0xB091F9c43b4D58C6040F65841E4DF8Be27726BC2
https://mumbai.polygonscan.com/address/0xB091F9c43b4D58C6040F65841E4DF8Be27726BC2

G. Verify the contract because Slither works only on Verified Contracts:

Slither-read–storage task needs the source of the contracts available.

Update hardhat.config.js for accommodating polygon api key

```
16    },
17    etherscan: {
18      apiKey: {
19        // sepolia: process.env.ETHERSCAN_API_KEY as string,
20        polygonMumbai: process.env.ETHERSCAN_API_FOR_POLYGON,
21      },
22    },
```

Write the verified script.

- Mention the address of the contract to verify.
- Give the path in the following order along with the smart contract name.

```
scripts > verify.js > main
1   const Hre = require("hardhat");
2
3   async function main() {
4     await Hre.run("verify:verify", {
5       address: "0xB091F9c43b4D58C6040F65841E4DF8Be27726BC2",
6       contract: "contracts/NestedStructs.sol:NestedStructExample",
7     });
8
9     console.log("Verified!!!");
10  }
11
12  main()
13    .then(() => process.exit(0))
14    .catch((error) => {
15      console.error(error);
16      process.exit(1);
17    });
18
```

- Run the command to verify the contract. Mention the mumbai_testnet as defined in the hardhat.config.js file that we have made. [6]

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 as of now 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.

The current slither detector is unable to detect the nested Structs. We will improve the detector to detect and suggest better!

VI. Target Users/Customers:

- CSE 6324-001 class
- Aditya Vichare (CSE 6324-001 student)
- Blockchain Developers
- Smart Contract Auditors
- Open-source Contributors
- Educators and Trainees.

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://onboardbase.com/blog/env-file-guide/

11. https://docs.arbitrum.io/stylus/how-tos/local-stylus-dev-node
12. https://stackoverflow.com/questions/70011797/return-nested-struct-inside-struct-in-solidity

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