Iteration 2 Report

Adding new detectors to Slither

The University of Texas at Arlington

Advanced Topics in Software Engineering

Fall 2022, CSE 6324 - 001

GitHub link: https://github.com/AtulUpadhye17/CSE-6324-Team4

Team 8:

Atul Upadhye (1002030159)

Kundana Vaka(1001827398)

Yamini Dhulipalla(1001913007)

Srikar Sai Yarlagadda(1001860709)

Vigneshwar Selvaraj(1001863627)

Introduction:

Slither is a solidity static analysis framework written in Python 3. It provides fine-grained information about smart contract code and has the necessary flexibility to support many applications. The framework is currently used for the following.[1]

- Automated Vulnerability Detection
- Automated Optimization Detection
- Code Understanding
- Assisted Code Review

Slither has predefined list of “detectors” which are vulnerabilities and optimizations that slither will look for into your smart contracts.[1]

Project Plan:

Objective:

For our project, we plan to implement new detectors and integrate them into the slither developer version to analyze the solidity smart contracts efficiently.

Overview of Slither differs from other static analysis tools:

“Trail of Bits” has published a paper on Slither and compares its bug detection with other static analysis tools by doing experiments for finding issues in Ethereum smart contracts in terms of speed, robustness, the balance of detection, and false positives.[1]

Comparison with Existing tool for Iteration 2:

We compared Slither with other static analysis frameworks such as Solhint, Solium as well as SpotBugs related to Java for identifying detectors. For Solidity specific , we compared Solhint and Slither:

Detectors	Solhint	Current Slither	Extended Slither (Our Tool)
Incorrect Constructor Name	Yes	No	Yes
Incorrect Constructor Order	Yes	No	Yes
Max-Line Length	Yes	No	Planned for Iteration 3.

Iteration 2 Plan:

Identified the vulnerabilities that slither failed to examine in the smart contracts and came up with detectors missing in the tool.

Installed and set up the environment on our local machines.

Experimented with the tool to identify whether a particular detector is already part of it.

Implemented two new detectors and run slither on the solidity files.

Iteration 3 Plan:

Identify and implement the remaining detectors.

Integrate Slither into our development process.

Build and test the tool.

Merge the code and perform code review.

Risks Faced during Iteration 2:

Risk	Type	Plan for Mitigating
Facing issues related to the downloading the source code of the Slither from Git and building it by following Developer Instructions provided.	Major	We have posted for help regarding this on the slack channel. We have also asked Shovon to help figuring out the part of integration and additional detectors we can implement
Integrating the implemented detectors in the slither tool as a plugin.	Major	

Specification and Design:

Implemented Detectors:

Incorrect Constructor Name:

Prior to solidity version 0.4.22, the only way to define a constructor was to create a function with the same name as the contract class. A function meant to become a constructor becomes a standard, callable function if its name doesn't exactly match the contract name.[3]

When smart contract code is reused under a different name without changing the name of the constructor function, security issues can arise.

Remediation: Solidity version 0.4.22 introduces a new constructor keyword that makes constructor definitions clearer. It is recommended to upgrade the contract to a recent version of the Solidity compiler and change to the new constructor declaration.[3]

Incorrect Constructor Order:

Variable initialization and function call dependencies should generally point downward. That is, the initialization should be followed by the call function and the calling function. This causes a flow from high to low level in the source code module.

In each contract, the Incorrect Constructor Order Detector examines the sequence in which the constructor is ordered with other functions. When functions are placed before the constructor, the detector generates a message.

Existing Detector: Visibility Not Set

Functions that do not have a function visibility type specified are public by default. This can lead to a vulnerability if a developer forgets to set the visibility, and a malicious user can make unauthorized or unintended state changes.[5]

Remediation: Functions can be specified as being external, public, internal, or private. It is recommended to make a conscious decision on which visibility type is appropriate for a function. This can dramatically reduce the attack surface of a contract system.[5]

Tools and Technologies:

- Visual Studio IDE
- Ubuntu VM
- Python 3

Environmental Setup:

For Mac OS:

Install VirtualBox on Mac

Create Linux Ubuntu Virtual Machine on VirtualBox

Install SEED ubuntu on Virtual Machine using VirtualBox

Install python 3

For Windows:

Install python 3

To install Slither analyzer and solidity compiler:

```
sudo pip3 install slither-analyzer
sudo add-apt-repository ppa:ethereum/ethereum
sudo apt-get update
sudo apt-get install solc
```

To install developer version of slither [6]:

Use virtualenv to install developer version of Slither.

```
pip3 install virtualenvwrapper
source /usr/local/bin/virtualenvwrapper.sh
mkvirtualenv --python=`which python3` slither-dev
git clone https://github.com/trailofbits/slither
cd slither
pip install -e ".[dev]"
```

Code and Tests:

Input:

Solidity smart contract file

Output:

The smart contract's detected vulnerabilities are provided.

Incorrect Constructor Name Detector:

Python Code:

pseudo_invalid_constructor.py

```python
from slither.slither import Slither

slither = Slither('./invalid_constructor_name.sol')

for contract in slither.contracts:
    #Get the function names
    print ('Contract: '+ contract.name)
    for function in contract.functions:
            #Check if any function name matches with contract name
            if function.name.lower() == contract.name.lower():
                print('Invalid Constructor Name : {}'.format(function.name))

```

Sample Contract:

invalid_constructor_name.sol

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract Missing{
    address private owner;

    modifier onlyowner {
        require(msg.sender==owner);
        _;
    }

    function missing() public{
        owner = msg.sender;
    }
}
```

Output:

```
[10/15/22]seed@VM:~$ python3 invalid_constructor_name.py
Contract: Missing
Invalid Constructor Name : missing
[10/15/22]seed@VM:~$
```

Detector Code as per Slither Detector Documentation Guidelines:

incorrect_constructor_name.py

```python
from slither.detectors.abstract_detector import AbstractDetector, DetectorClassification

class IncorrectConstructorName(AbstractDetector):
    """
    Detect Incorrect Constructor Name
    """

    ARGUMENT = "incorrect-constructor-name"
    HELP = "Incorrect Constructor Name should be proper."
    IMPACT = DetectorClassification.HIGH
    CONFIDENCE = DetectorClassification.HIGH

    WIKI = ""
    WIKI_TITLE = "Incorrect Constructor Name"
    WIKI_DESCRIPTION = "Detect Incorrect Constructor Name"
    WIKI_EXPLOIT_SCENARIO = ".."
    WIKI_RECOMMENDATION = ".."

    def _detect(self):
        results = []

        for contract in self.slither.contracts_derived:
            for f in contract.functions:
                #Check if any function name matches with contract name
                if f.name.lower() == contract.name.lower():

                    # Info to be printed
                    info = ["Incorrect Constructor Name found in ",f,"\n"]

                    res = self.generate_result(info)

                    results.append(res)

        return results


```

Incorrect Constructor Order Detector:

Python Code:

```python
from slither.slither import Slither

slither = Slither('constructor.sol')


for contract in slither.contracts:
    li=contract.functions
    print ('Contract: '+ contract.name)
    li1=contract.functions[0]
    for x in range(len(li)):
        #Check if constructor is present and placed before the functions
        if str(li[x]) == "constructor" and li[x]!=li[0]:
            print("Incorrect constructor Order")
```

Output:

```
PS C:\Slither> python findCall.py
Contract: HelloBlockchain
Incorrect constructor Order
PS C:\Slither>
```

Sample Contract:

C: > Slither > constructor.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.9.0;
contract HelloBlockchain
{
    enum StateType { Request, Respond}
    StateType public  State;
    address public  Requestor;
    address public  Responder;
    string public RequestMessage;
    string public ResponseMessage;
    function SendRequest(string memory requestMessage) public
    {
        if (Requestor != msg.sender)
        {
            revert();
        }
        RequestMessage = requestMessage;
        State = StateType.Request;
    }
    function SendResponse(string memory responseMessage) public
    {
        Responder = msg.sender;
        ResponseMessage = responseMessage;
        State = StateType.Respond;
    }
    constructor(string memory message) public
    {
        Requestor = msg.sender;
        RequestMessage = message;
        State = StateType.Request;
    }
}
```

Detector Code as per Slither Detector Documentation Guidelines:

C: > Slither > Incorrect_constructor_order.py > ...

```
from slither.detectors.abstract_detector import AbstractDetector, DetectorClassification


class IncorrectConstructorOrder(AbstractDetector):
    """
    Detect Constructor not in sequence with functions
    """

    ARGUMENT = 'incorrect-constructor-order' # slither will launch the detector with slither.py --detect incorrect-constructor-order
    HELP = 'constructor not in sequence with functions'
    IMPACT = DetectorClassification.HIGH
    CONFIDENCE = DetectorClassification.HIGH

    WIKI = ''
    WIKI_TITLE = 'Incorrect Constructor Order'
    WIKI_DESCRIPTION = 'Detect Constructor not in sequence with functions'
    WIKI_EXPLOIT_SCENARIO = ''
    WIKI_RECOMMENDATION = ''

    def _detect(self):
        results = []

        for contract in self.slither.contracts_derived:

            list_of_methods=contract.functions

            for x in range(len(list_of_methods)):
                #Check if constructor is present and placed before the functions
                if str(list_of_methods[x]).lower == "constructor" and list_of_methods[x]!=list_of_methods[0]:

                    #info to be printed
                    info = ['Incorrect constructor Order found in ',contract,"\n"]

                    res = self.generate_result(info)

                    results.append(res)

        return results
```

Existing Detector "Visibility Not Set" is already a part of the compiler:

We found that, if function visibility is not set, the tool itself reports this issue:

Sample Contract:

contractDemo.sol

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.17;
3
4  contract ContractDemo{
5
6      function no_visibility()  {
7
8      }
9
10 }
```

Output by Slither Tool:

```
[10/15/22]seed@VM:~$ slither contractDemo.sol
Compilation warnings/errors on contractDemo.sol:
Warning: SPDX license identifier not provided in source file. Before publishing, consider ad
ding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use
 "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org
 for more information.
--> contractDemo.sol

Error: No visibility specified. Did you intend to add "public"?
 --> contractDemo.sol:5:6:
  |
5 |             function no_visibility()  {
  |             ^ (Relevant source part starts here and spans across multiple lines).
```

Customer and Users:

- Any security audit firm, smart contract developer, security expert, or academic researcher can use this tool with our new detectors added to Slither to make the smart contract auditing process more efficient.
- Shovon.

Feedback:

- We asked Shovon to review our implemented detectors, and he suggested that we since most of detectors are focused on Constructor part, we can explore more constructors related guidelines to be followed in Solidity.
- Shovon also suggested that we can focus on Solidity Style Guidelines and try to implement detectors for them. We will try to implement them in Iteration 3.

References:

[1] Overview. (n.d.). Retrieved October 15, 2022, from https://blog.trailofbits.com/2019/05/27/slither-the-leading-static-analyzer-for-smart-contracts/

[2] A. Vulnerability Detection Evaluation Retrieved October 15, 2022, from https://arxiv.org/pdf/1908.09878.pdf

[3] SWC-118 · Overview. (n.d.). Retrieved October 15, 2022, from https://swcregistry.io/docs/SWC-118

[4] Martin, R. (2017, September 10). Clean Architecture: A Craftsman's Guide to Software Structure and Design (Robert C. Martin Series) (1st ed.). Pearson.

[5] SWC-100 · Overview. (n.d.). Retrieved October 15, 2022, from https://swcregistry.io/docs/SWC-100

[6] GitHub - crytic/slither: Developer installation. (n.d.). GitHub. Retrieved October 15, 2022, from https://github.com/crytic/slither/wiki/Developer-installation

[7] Rule Index of Solhint. (n.d.). Solhint. Retrieved October 17, 2022, from https://protofire.github.io/solhint/docs/rules.html