Adding new Detectors to Slither

SLITHER

Advanced Topics in Software Engineering(CSE 6324)Fall 2022
Team 4
Iteration 2

- Atul Upadhye
- Kundana Vaka
- Yamini Dhulipalla
- Srikar Sai Yarlagadda
- Vigneshwar Selvaraj

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

Slither[1]

- **Slither** is a Solidity static analysis framework written in Python 3. It runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to easily write custom analyses. Slither enables developers to find vulnerabilities, enhance their code comprehension, and quickly prototype custom analyses [1].
  - Features [1]:
    - Detects vulnerable Solidity code with low false positives
    - Detector API to write custom analyses in Python.
    - Built-in 'printers' quickly report crucial contract information.
    - Many more..

Image Copied from : source

Some of the Current Detectors(80) [1]

- name-reused : Contract's name reused.
- uninitialized-state : Uninitialized state variables.
- dead-code : Functions that are not used.
- similar-names: Variable names are too similar.
- uninitialized-local : Uninitialized local variables.

Plan:

Iteration 2	Iteration 3
• Compare Slither with other static analysis tools : Solhint, SpotBugs(Java), Solium. • Identified detectors missing or should be part of Slither [2]: 1. Incorrect Constructor Name [5]. 2. Incorrect Constructor Order. 3. Visibility not set [6]. • Installing and setup Slither. We used Ubuntu VM for building the tool. • Write code and test detector for each of the identified detectors. [4].	• Find more detectors and implement. • Integrate implemented detectors in the Slither and build tool. • Test the tool.

Incorrect Constructor Name [5]

- Description:
  - Constructors are special functions that are called only once during the contract creation. They often perform critical, privileged actions such as setting the owner of the contract.
  - Before Solidity version 0.4.22, the only way of defining a constructor was to create a function with the same name as the contract class containing it. A function meant to become a constructor becomes a normal, callable function if its name doesn't exactly match the contract name.
  - This behavior sometimes leads to security issues, in particular when smart contract code is re-used with a different name but the name of the constructor function is not changed accordingly.

- Remediation:
  - Solidity version 0.4.22 introduces a new constructor keyword that make a constructor definitions clearer. It is therefore recommended to upgrade the contract to a recent version of the Solidity compiler and change to the new constructor declaration.

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

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 [4]

- Slither provides Python API [3].
- Given .sol contract file as input, we can access the following using the API:
  - contracts_derived() : list(Contracts)
  - contract.name : (str) Name of the Contract.
  - contract.functions :list(Functions)

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 [7]

- Description:
  - Initializing variables is an integral part of programming and constructors can be used for the same with efficient memory management.
  - In general, we want variable initialization and function call dependencies to point in the downward direction. That is, the initialization should be followed by the function that is called and the function that does the calling. This creates a flow down the source code module from high level to low level.
  - The Incorrect Constructor order detector analyzes the sequence in which constructor is ordered with other functions in each contract. The detector throws the message when functions are placed before the constructor.

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

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

Sample Contract

C: > Slither > constructor.sol

```solidity
// 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 [4]

- Slither provides Python API [3].
- Given .sol contract file as input, we can access the following using the API:
  - contracts_derived() : list(Contracts)
  - contract.functions :list(Functions)
  - AbstractDetector object has the slither attribute, which returns the current Slither object.

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

```python
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-
    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
```

Visibility Not Set [6]

- Description:
  - Functions that do not have a function visibility type specified are public by default. This can lead to a vulnerability if a developer forgot to set the visibility and a malicious user is able to make unauthorized or unintended state changes.
- 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.

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

Already a part of the compiler

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

```
[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).
```

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 }
```

Risks

- After implementing **Incorrect Constructor Name and Incorrect Constructor Order** detector, we tried to integrate it in the Slither tool as a plugin.
- However, we were not able to rebuild the complete tool, we were facing issues related to the downloading the source code of the Slither from Git and building it by following Developer Instructions provided.
- How do we plan to resolve this ?
  - 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.
- In the meantime, until we get the necessary help, we are implementing other detectors.

Customers 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.

- **Feedback from Shovon:**

- 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. GitHub - crytic/slither: Static Analyzer for Solidity. (n.d.). GitHub. Retrieved October 15, 2022, from https://github.com/crytic/slither
2. Overview · Smart Contract Weakness Classification and Test Cases. (n.d.). Retrieved October 15, 2022, from https://swcregistry.io/
3. Python API · crytic/slither Wiki. (n.d.). GitHub. Retrieved October 15, 2022, from https://github.com/crytic/slither/wiki/Python-API
4. Adding a new detector · crytic/slither Wiki. (n.d.). GitHub. Retrieved October 15, 2022, from https://github.com/crytic/slither/wiki/Adding-a-new-detector
5. SWC-118 · Overview. (n.d.). Retrieved October 15, 2022, from https://swcregistry.io/docs/SWC-118
6. SWC-100 · Overview. (n.d.). Retrieved October 15, 2022, from https://swcregistry.io/docs/SWC-100
7. Rule Index of Solhint. (n.d.). Solhint. Retrieved October 17, 2022, from https://protofire.github.io/solhint/docs/rules.html
8. Style Guide — Solidity 0.8.18 documentation. (n.d.). Retrieved October 17, 2022, from https://docs.soliditylang.org/en/develop/style-guide.html
9. Martin, R. (2017, September 10). Clean Architecture: A Craftsman's Guide to Software Structure and Design (Robert C. Martin Series) (1st ed.). Pearson.

Thank You