Final Report

Enhancement of slither for analyzing smart contracts

ADV TOPS SOFTWARE ENGINEERING

2232-CSE-6324-004

Github link: https://github.com/Sampath2901/6324-Project#6324-project

TEAM - 8

SAI NIKHIL KANCHUKATLA – 1002034488
SAMPATH KUMAR MEDIPUDI - 1002032901
SAI KRISHNAM RAJU BHUPATHIRAJU - 1002019782
FARAZ SHAIK AHMAR - 1002035224

Project Plan:

Features:

Inheritance hierarchy:

This is the feature that we are going to add for missing zero check detector.

Iteration Plan

Iterations	Goals	Status
1 (Current)	• To gain a deeper understanding of the slither tool, we perform analysis on example smart contracts, providing a practical experience and familiarization with the tool. • Set up and configure the slither tool in order to reproduce an identical error message. • Evaluate the issue more comprehensively and devise a plan to address and resolve it.	completed
2	• Examine the "missing-zero-check" detector, comprehend all its attributes, and grasp its operational flow. • Develop pseudo code that will facilitate the implementation of the improvements we aim to introduce to the slither tool.	completed

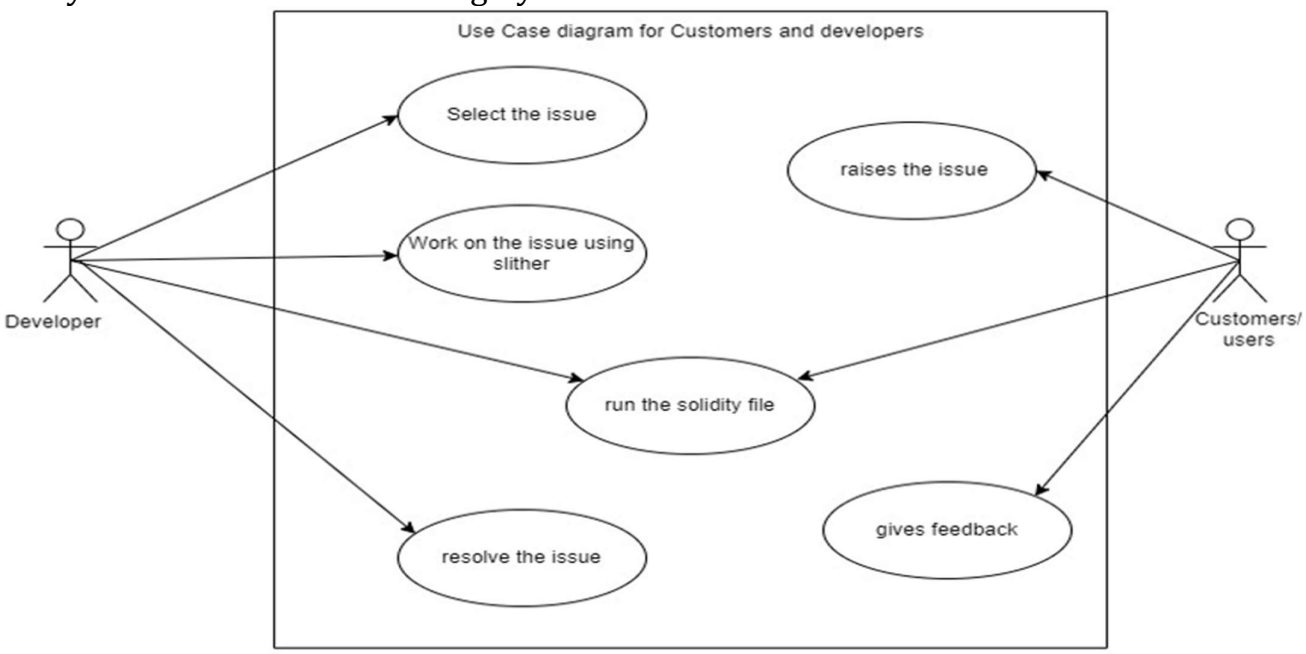
3	• Translate the pseudo code into real code and evaluate the tool to determine if any new vulnerabilities arise after modifying the existing tool. • Verify the effectiveness of the improvements made to the tool by testing it on smart contracts.	completed

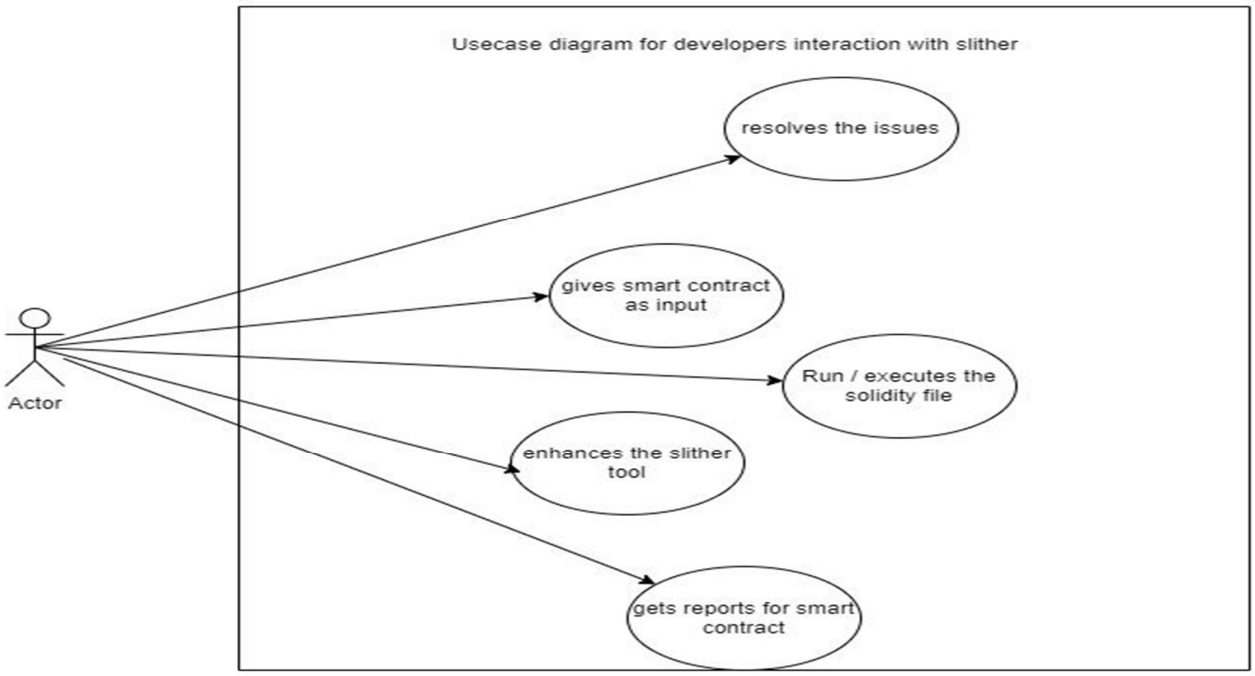
Specification and design:

• Inputs and Outputs:

- Input: Smart contracts
- Output: The analysis findings from conducting test cases.

• Use-Case (User) [1]: With the enhancements made in the Slither tool, the user can analyze smart contracts thoroughly.

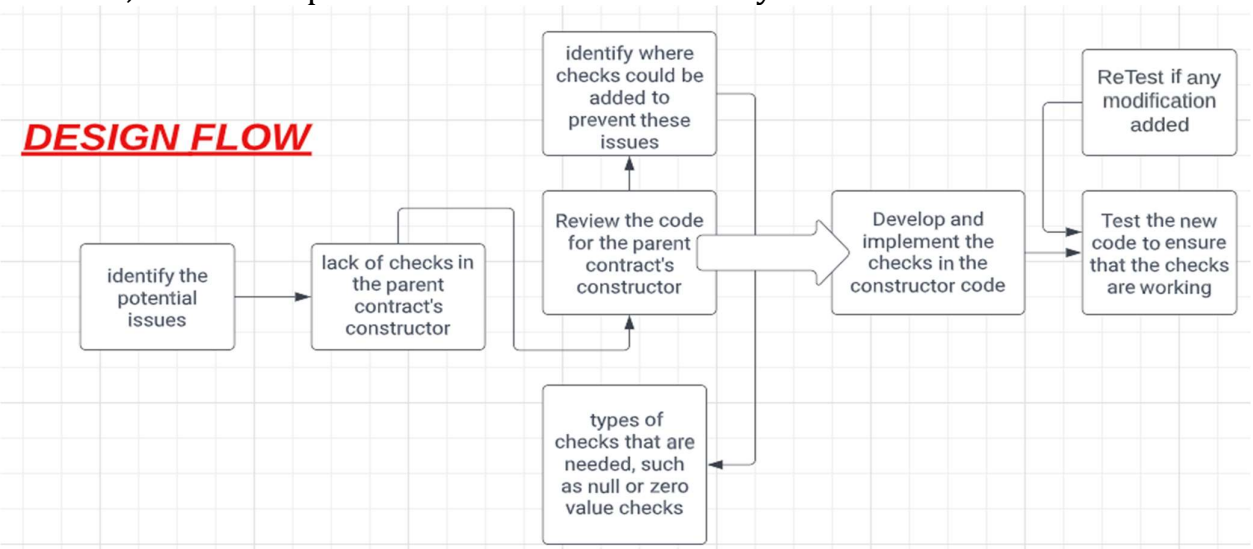




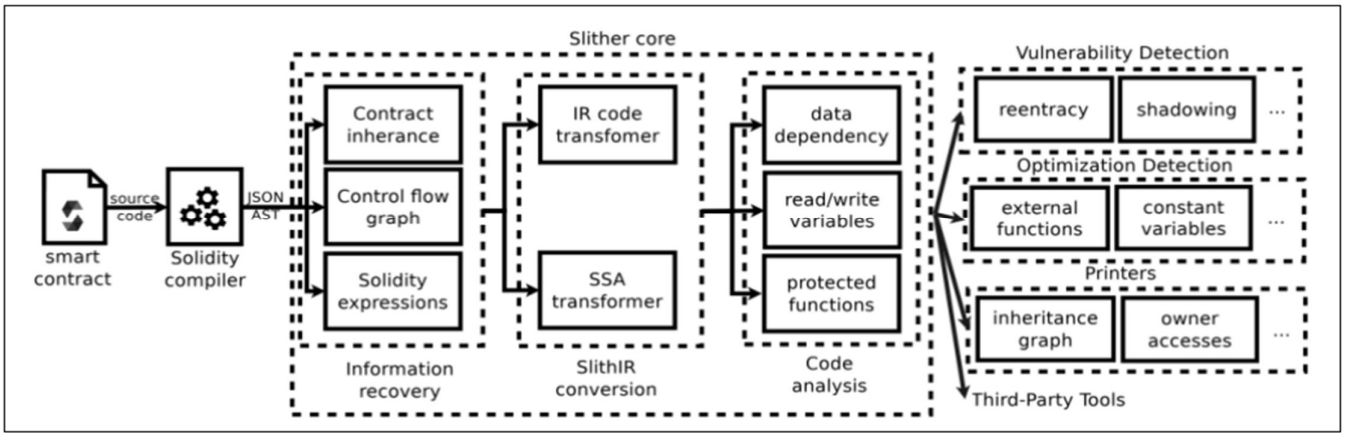
• Mock Designs:

The main idea behind the feature: The zero address is commonly used as a default or provisional value for an address variable when the address parameter or property is not yet initialized. Checking for zero addresses is vital in smart contract security to prevent potential vulnerabilities like reentrancy attacks, where a malicious contract can repeatedly call a vulnerable contract's functions and exhaust its funds.

However, the 'missing-zero-check detector in Slither only examines zero-access checks within the same contract and does not take into account the inheritance hierarchy. To enhance the detector's effectiveness and boost the overall security of analyzed contracts, we plan to include the inheritance hierarchy in the 'missing-zero-check' detector, which is expected to enhance its efficiency.



Architecture Slither: [13]



Competitors for Slither:

- Mythril and Securify properly detected issues in 6 out of 10 DeFi smart contracts that were vulnerable, and Slither in 5 out of 10. [1]
- Securify classified the majority of the errors as violations, which indicates that the tool has a high level of confidence in its ability to recognize security flaws. Additionally, Securify discovered problems with the usage of low-level calls and the absence of validation for user-controlled data, both of which are against the guidelines for safe development in the Solidity programming language. [10]
-

The name of the vulnerable smart contract	Known vulnerabilities discovered?			Audit execution time (seconds)		
	Mythril	Securify	Slither	Mythril	Securify	Slither
unprotected0	no	yes	no	3	2	2
overflow_single_tx	yes	no	no	9	9	2
lottery	no	yes	yes	3	30	3
dos_simple	yes	no	no	3	2	2
FindThisHash	yes	yes	yes	19	2	1
reentrancy_simple	yes	yes	yes	2	3	2
short_address_exampe	no	no	no	2	2	1
roulette	yes	yes	yes	25	2	2
lotto	yes	yes	yes	2	31	2
NaiveReceiver	no	no	no	10	9	2
Total	**6**	**6**	**5**	**78**	**92**	**19**

[https://www.h-x.technology/blog/top-3-smart-contract-audit-tools]

- The below table is the comparison of Accuracy, Performance, Robustness and Reentrancy examples of different tools. [11]

		Slither	Securify	SmartCheck	Solhint
Accuracy	False positives	10.9%	25%	73.6%	91.3%
	Flagged contracts	112	8	793	81
	Detections per contract	3.17	2.12	10.22	2.16
Performance	Average execution time	0.79 ± 1	41.4 ± 46.3	10.9 ± 7.14	0.95 ± 0.35
	Timed out analyses	0%	20.4%	4%	0%
Robustness	Failed analyses	0.1%	11.2%	10.22%	1.2%
Reentrancy examples	DAO	✓	✗	✓	✗
	Spankchain	✓	✗	✗	✗

[https://doi.org/10.48550/arXiv.1908.09878]

- The below table is the summary of current tools and their characteristics. [12]

Tool	Level of rigor	Analysis basis	Interface Means	Description
Oyente	Heuristic	source code (.sol)	Command line-based tool	Oyente [25] is an automated security analysis tool for revealing security vulnerabilities in smart contracts.
Mythril	Analytic and Heuristic	Byte code and source code (.sol)	Command line-based tool	Mythril [29] is an automated security analysis tool for Ethereum smart contracts. It uses Concolic analysis, taint analysis and control flow checking to detect a variety of security vulnerabilities. The analysis is based on laser-Ethereum, a symbolic execution library for EVM bytecode.
Securify	Formal	Byte code and source code (.sol)	UI-based tool	Securify [22] is an automated formal security analysis tool for Ethereum smart contracts. It can detect various security issues such as input validation, reentrancy, and others.
SmartCheck	Analytic and Heuristic	source code (.sol)	UI-based tool	SmartCheck [38] is an automated static code analyzer developed by SmartDec Security Team. It runs analysis in Solidity source code and automatically checks smart contracts for security vulnerabilities and bad practices.

[https://doi.org/10.48550/arXiv.1809.02702]

Features [14]:

- Detects vulnerable Solidity code with low false positives (see the list of trophies)
- Identifies where the error condition occurs in the source code
- Easily integrates into continuous integration and Truffle builds
- Built-in 'printers' quickly report crucial contract information
- Detector API to write custom analyses in Python
- Ability to analyze contracts written with Solidity >= 0.4
- Intermediate representation (SlithIR) enables simple, high-precision analyses
- Correctly parses 99.9% of all public Solidity code

- Average execution time of less than 1 second per contract

Code and Tests:
Configuration and setup:[4]

Slither requires Python 3.8+ and solc, the Solidity compiler. We recommend using solc-select to conveniently switch between solc versions, but it is not required. For additional configuration, see the usage documentation.[4]

First, we utilized the Digital Ocean platform to generate a Linux server and installed Visual Studio on it. Next, we included remote SSH access and created a root account to connect to the server. Finally, we followed the below set of instructions to install the necessary dependencies.

- Installed python and checked for the version using the following command:
  python3 –version

- Install npm using the following command:/
  apt install npm

- Install nodejs using the following command:
  sudo apt install nodejs

- Install solc using the following command:
  apt install solc

- Install slither using the following command:
  git clone https://github.com/crytic/slither.git && cd slither
  python3 setup.py install

Issue Description

Issue #981: Copy-paste from github[8]
“missing-zero-check” detector doesn't seem to check the arguments for constructor of parent contracts. Even if the parent contract's constructor has zero-access checks & the child contract is using the same variable, it's gets flagged. [8]

```solidity
pragma solidity 0.8.19;


abstract contract Ownable {
    address public owner1;
    address public owner2;

    constructor (address __owner1, address __owner2) {
        require(__owner1 != address(0), "Zero");
        owner1 = __owner1;
        owner2 = __owner2;
    }
}

contract ABC is Ownable {

    address public owner3;

    constructor(address _owner1, address _owner2) Ownable(_owner1, _owner2) {
        owner3 = _owner1;
    }
}
```

[Figure,[8] https://github.com/crytic/slither/issues/981]

Slither output:

```
ABC.constructor(address,address)._owner1 (contracts/ABC.sol#21) lacks a zero-check on :
                - owner3 = _owner1 (contracts/ABC.sol#22)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
. analyzed (2 contracts with 103 detectors), 1 result(s) found
```

[Figure,[8] https://github.com/crytic/slither/issues/981]

We managed to set up the Slither tool correctly and were able to replicate an identical error message as depicted in the below image

```
root@ubuntu-s-1vcpu-2gb-amd-fra1-01:~# slither issue.sol
Compilation warnings/errors on issue.sol:
Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a com
ment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Ident
ifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information.
--> issue.sol



ABC.constructor(address,address)._owner1 (issue.sol#19) lacks a zero-check on :
                - owner3 = _owner1 (issue.sol#20)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validati
on

Pragma version0.8.19 (issue.sol#1) necessitates a version too recent to be trusted. Consider deploying
 with 0.6.12/0.7.6/0.8.16
solc-0.8.19 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidit
y

ABC.owner3 (issue.sol#17) should be immutable
Ownable.owner1 (issue.sol#5) should be immutable
Ownable.owner2 (issue.sol#6) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be
-declared-immutable
issue.sol analyzed (2 contracts with 84 detectors), 6 result(s) found
```

- We must examine the "missing-zero-check" detector, comprehend all its attributes, and grasp its operational flow.
- We need to modify “MissingzeroAddressValidation” and add an inheritance hierarchy to it.
- The below figure contains all the information regarding “MissingzeroAddressValidation”

missing_zero_address_validation.py 1, M

slither > slither > detectors > operations > missing_zero_address_validation.py > ...

```python
"""
Module detecting missing zero address validation

"""
from collections import defaultdict
from typing import DefaultDict, List, Tuple, Union

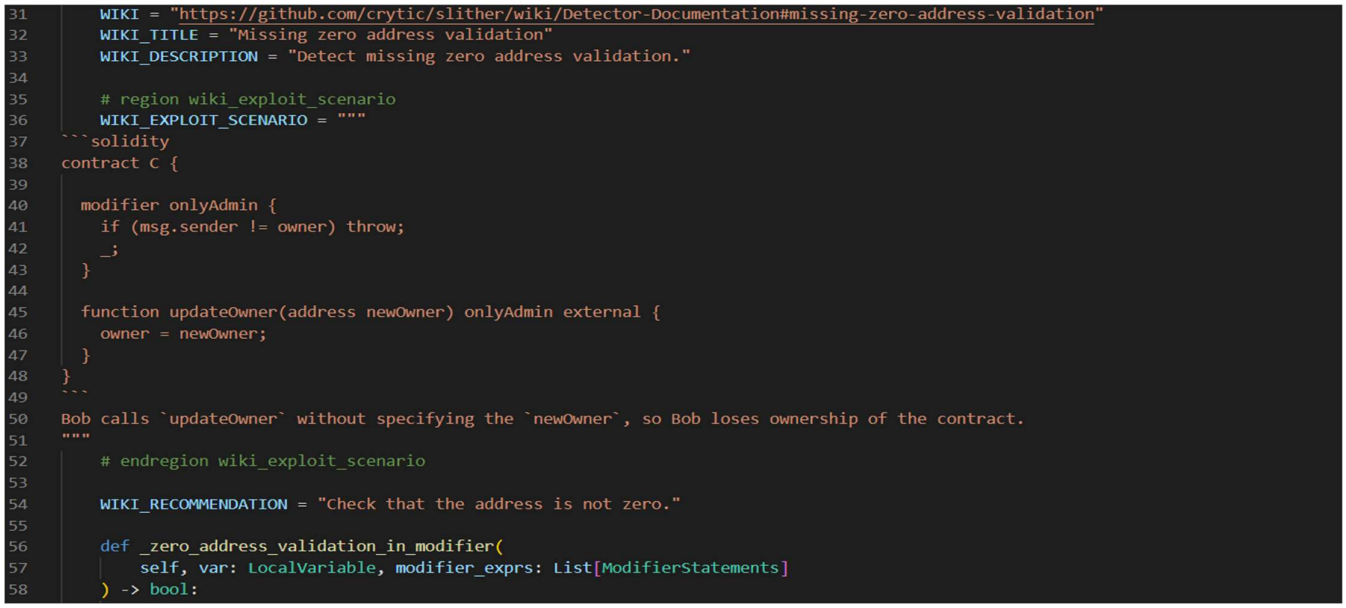
from slither.analyses.data_dependency.data_dependency import is_tainted
from slither.core.cfg.node import Node
from slither.core.declarations.contract import Contract
from slither.core.declarations.function import ModifierStatements
from slither.core.declarations.function_contract import FunctionContract
from slither.core.solidity_types.elementary_type import ElementaryType
from slither.core.variables.local_variable import LocalVariable
from slither.detectors.abstract_detector import AbstractDetector, DetectorClassification
from slither.slithir.operations import Call
from slither.slithir.operations import Send, Transfer, LowLevelCall
from slither.utils.output import Output


class MissingZeroAddressValidation(AbstractDetector):
    """
    Missing zero address validation
    """

    ARGUMENT = "missing-zero-check"
    HELP = "Missing Zero Address Validation"
    IMPACT = DetectorClassification.LOW
    CONFIDENCE = DetectorClassification.MEDIUM

```

[Figure, [4] https://github.com/crytic/slither]



```
    WIKI = "https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation"
    WIKI_TITLE = "Missing zero address validation"
    WIKI_DESCRIPTION = "Detect missing zero address validation."

    # region wiki_exploit_scenario
    WIKI_EXPLOIT_SCENARIO = """
```solidity
contract C {

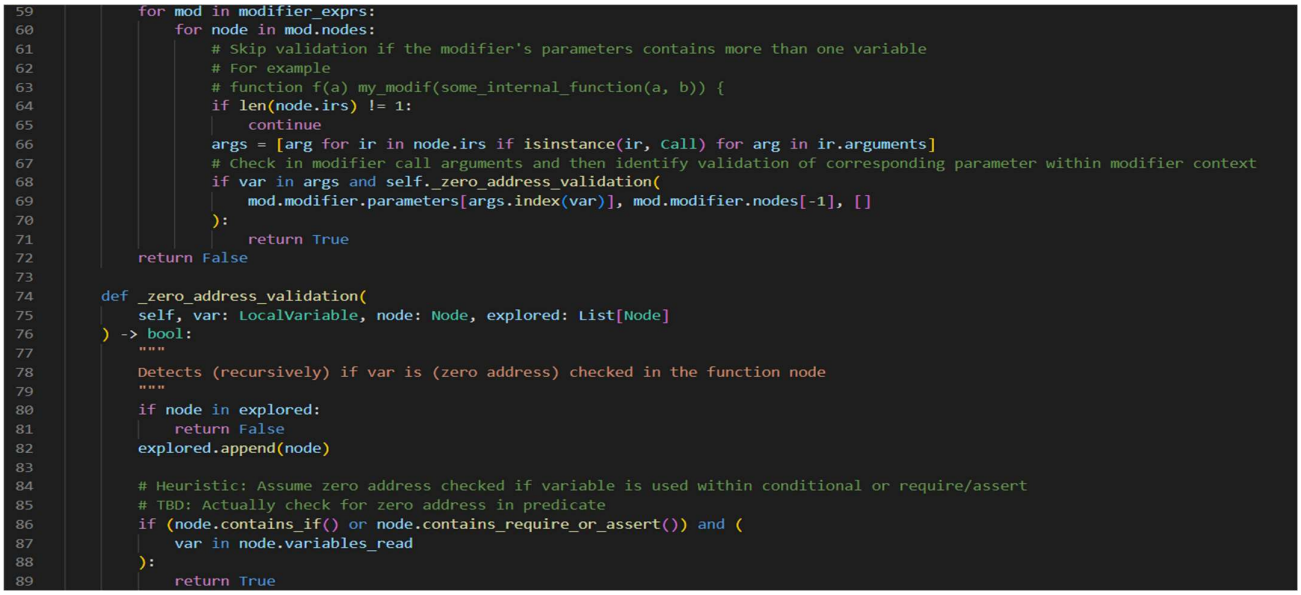
  modifier onlyAdmin {
    if (msg.sender != owner) throw;
    _;
  }

  function updateOwner(address newOwner) onlyAdmin external {
    owner = newOwner;
  }
}
```
Bob calls `updateOwner` without specifying the `newOwner`, so Bob loses ownership of the contract.
"""
    # endregion wiki_exploit_scenario

    WIKI_RECOMMENDATION = "Check that the address is not zero."

    def _zero_address_validation_in_modifier(
        self, var: LocalVariable, modifier_exprs: List[ModifierStatements]
) -> bool:
```

[Figure, [4] https://github.com/crytic/slither]



```
        for mod in modifier_exprs:
            for node in mod.nodes:
                # Skip validation if the modifier's parameters contains more than one variable
                # For example
                # function f(a) my_modif(some_internal_function(a, b)) {
                if len(node.irs) != 1:
                    continue
                args = [arg for ir in node.irs if isinstance(ir, Call) for arg in ir.arguments]
                # Check in modifier call arguments and then identify validation of corresponding parameter within modifier context
                if var in args and self._zero_address_validation(
                    mod.modifier.parameters[args.index(var)], mod.modifier.nodes[-1], []
                ):
                    return True
        return False

    def _zero_address_validation(
        self, var: LocalVariable, node: Node, explored: List[Node]
    ) -> bool:
        """
        Detects (recursively) if var is (zero address) checked in the function node
        """
        if node in explored:
            return False
        explored.append(node)

        # Heuristic: Assume zero address checked if variable is used within conditional or require/assert
        # TBD: Actually check for zero address in predicate
        if (node.contains_if() or node.contains_require_or_assert()) and (
            var in node.variables_read
        ):
            return True
```

[Figure, [4] https://github.com/crytic/slither]

```
        # Check recursively in all the parent nodes
        for father in node.fathers:
            if self._zero_address_validation(var, father, explored):
                return True
        return False

    def _detect_missing_zero_address_validation(
        self, contract: Contract
    ) -> List[Union[Tuple[FunctionContract, DefaultDict[LocalVariable, List[Node]]]]]:
        """
        Detects if addresses are zero address validated before use.
        :param contract: The contract to check
        :return: Functions with nodes where addresses used are not zero address validated earlier
        """
        results = []

        for function in contract.functions_entry_points:
            var_nodes = defaultdict(list)

            for node in function.nodes:
                sv_addrs_written = [
                    sv
                    for sv in node.state_variables_written
                    if sv.type == ElementaryType("address")
                ]

                addr_calls = False
                for ir in node.irs:
                    if isinstance(ir, (Send, Transfer, LowLevelCall)):
                        addr_calls = True
```

[Figure, [4] https://github.com/crytic/slither]

```
                # Continue if no address-typed state variables are written and if no send/transfer/call
                if not sv_addrs_written and not addr_calls:
                    continue

                # Check local variables used in such nodes
                for var in node.local_variables_read:
                    # Check for address types that are tainted but not by msg.sender
                    if var.type == ElementaryType("address") and is_tainted(
                        var, function, ignore_generic_taint=True
                    ):
                        # Check for zero address validation of variable
                        # in the context of modifiers used or prior function context
                        if not (
                            self._zero_address_validation_in_modifier(
                                var, function.modifiers_statements
                            )
                            or self._zero_address_validation(var, node, [])
                        ):
                            # Report a variable only once per function
                            var_nodes[var].append(node)
            if var_nodes:
                results.append((function, var_nodes))
        return results

    def _detect(self) -> List[Output]:
        """Detect if addresses are zero address validated before use.
        Returns:
            list: {'(function, node)'}
        """
```

[Figure, [4] https://github.com/crytic/slither]

```
151
152         # Check derived contracts for missing zero address validation
153         results = []
154         for contract in self.compilation_unit.contracts_derived:
155             missing_zero_address_validation = self._detect_missing_zero_address_validation(contract)
156             for (_, var_nodes) in missing_zero_address_validation:
157                 for var, nodes in var_nodes.items():
158                     info = [var, " lacks a zero-check on ", ":\n"]
159                     for node in nodes:
160                         info += ["\t\t- ", node, "\n"]
161                     res = self.generate_result(info)
162                     results.append(res)
163         return results
```

[Figure, [4] https://github.com/crytic/slither]

By Analyzing above code, these are distinct functions used:

1. **_zero_address_validation_in_modifier:** This method determines whether a local variable, which symbolizes an address, is checked against the zero address when used with a modifier. The local variable is examined to see if it is used as an input in a function call that denotes zero address validation as it iterates over the modifier expressions and their accompanying nodes. The method returns True, indicating that the zero address is being validated, if it is discovered that the local variable is being utilized in a function call that does so.

2. **_zero_address_validation:** The zero address in a function node is checked recursively by this technique to see if a local variable—which represents an address—is legitimate. It initially determines if the local variable is utilized within the function nodes require/assert or conditional statements. The method returns True, indicating that there is zero address validation if the local variable is discovered to be utilized in a conditional or require/assert statement.

3. **_detect_missing_zero_address_validation**: This method does the major analysis by iterating over all the functions in a contract and determining if the addresses used in the functions are validated against the zero address before being utilized. It determines if state variables of type address are written in a function node and whether send/transfer/call actions are present. If one of these requirements is fulfilled, the node's local variables are checked against the zero-address using the _zero_address_validation function. If a local variable is found to be utilized without being checked against the zero address, the function and the relevant nodes are added to the results list.

4. **_detect**: This function serves as the detector's entry point, and it calls the _detect_missing_zero_address_validation method for each contract created from the compilation unit. The results are collected and delivered as a list of Output objects, which reflect the detector's findings.

   - The current implementation of the MissingZeroAddressValidation detector does not specifically check for arguments passed to the constructor of parent contracts. The detector only analyzes the functions entry points of the derived contracts and does not differentiate between constructors and regular functions.

Functionality we implemented:

```
def _zero_address_validation_in_constructor(self, var: LocalVariable, constructor: Function) -> bool:
    if constructor:
        for node in constructor.nodes:
            if (node.contains_if() or node.contains_require_or_assert()) and (
                var in node.variables_read
            ):
                return True
        # Check recursively for parent constructors
        parent_contracts = constructor.contract.inheritance

        for parent_contract in parent_contracts:
            if self._zero_address_validation_in_constructor(var, parent_contract.constructor):
                return True

    return False
```

"_zero_address_validation_in_constructor" function checks for zero address validation of a variable in the context of the parent contract constructor. If the variable has already been validated in the parent constructor, the function should return True, indicating that zero address validation has already been performed.

Line to Line explanation of logic :

- **def _zero_address_validation_in_constructor(self, var: LocalVariable, constructor: Function) -> bool:**

Define a method called _zero_address_validation_in_constructor that takes in three parameters: self (which refers to the instance of the class), var (which is the local variable to check for zero-address validation), and constructor (which is the constructor function to check).

- **if constructor:**

Check if the constructor function exists. If it doesn't exist, then there is no

constructor to check, so return False.

- **for node in constructor.nodes:**

  Loop through all the nodes (basic blocks) in the constructor.

- **if (node.contains_if() or node.contains_require_or_assert()) and (var in node.variables_read):**

  Check if the node contains an if-statement, require-statement, or assert-statement and if var is read in this node. If both conditions are true, then var is being checked for zero address, so return True.

- **parent_contracts = constructor.contract.inheritance**

  Get a list of parent contracts that constructor inherits from.

- **for parent_contract in parent_contracts:**
  if self._zero_address_validation_in_constructor(var, parent_contract.constructor):
  return True

  For each parent contract, recursively call the _zero_address_validation_in_constructor method to check if var is zero-address validated in the parent contract's constructor. If var is zero-address validated in any parent contract's constructor, return True.

- **return False**

  If var is not zero-address validated in the current constructor or any parent constructor, return False.

Changes done in existing function _detect_missing_zero_address_validation like below picture

```python
# Check local variables used in such nodes
for var in node.local_variables_read:
    # Check for address types that are tainted but not by msg.sender
    if var.type == ElementaryType("address") and is_tainted(
        var, function, ignore_generic_taint=True
    ):
        # Check for zero address validation of variable
        # in the context of modifiers used or prior function context
        if not (
            self._zero_address_validation_in_modifier(
                var, function.modifiers_statements
            )
            or self._zero_address_validation(var, node, [])
            or self._zero_address_validation_in_constructor(var, function.contract.constructor)
        ):
            # Report a variable only once per function
            var_nodes[var].append(node)
```

Updated code:

slither > detectors > operations > missing_zero_address_validation.py > ...

```python
"""
Module detecting missing zero address validation

"""
from collections import defaultdict
from typing import DefaultDict, List, Tuple, Union

from slither.analyses.data_dependency.data_dependency import is_tainted
from slither.core.cfg.node import Node
from slither.core.declarations.contract import Contract
from slither.core.declarations.function import ModifierStatements
from slither.core.declarations.function_contract import FunctionContract
from slither.core.solidity_types.elementary_type import ElementaryType
from slither.core.variables.local_variable import LocalVariable
from slither.detectors.abstract_detector import AbstractDetector, DetectorClassification
from slither.slithir.operations import Call
from slither.slithir.operations import Send, Transfer, LowLevelCall
from slither.utils.output import Output
from slither.core.declarations import Function



class MissingZeroAddressValidation(AbstractDetector):
    """
    Missing zero address validation
    """

    ARGUMENT = "missing-zero-check"
    HELP = "Missing Zero Address Validation"
    IMPACT = DetectorClassification.LOW
    CONFIDENCE = DetectorClassification.MEDIUM
```

```
    WIKI = "https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation"
    WIKI_TITLE = "Missing zero address validation"
    WIKI_DESCRIPTION = "Detect missing zero address validation."

    # region wiki_exploit_scenario
    WIKI_EXPLOIT_SCENARIO = """
```solidity
contract C {

  modifier onlyAdmin {
    if (msg.sender != owner) throw;
    _;
  }

  function updateOwner(address newOwner) onlyAdmin external {
    owner = newOwner;
  }
}
```
Bob calls `updateOwner` without specifying the `newOwner`, so Bob loses ownership of the contract.
"""
    # endregion wiki_exploit_scenario

    WIKI_RECOMMENDATION = "Check that the address is not zero."

    def _zero_address_validation_in_modifier(self, var: LocalVariable, modifier_exprs: List[ModifierStatements]) -> bool:

        for mod in modifier_exprs:
            for node in mod.nodes:

                # Skip validation if the modifier's parameters contains more than one variable
                # For example
                # function f(a) my_modif(some_internal_function(a, b)) {
                if len(node.irs) != 1:
                    continue
                args = [arg for ir in node.irs if isinstance(ir, Call) for arg in ir.arguments]

                # Check in modifier call arguments and then identify validation of corresponding parameter within modifier context
                if var in args and self._zero_address_validation(
                    mod.modifier.parameters[args.index(var)], mod.modifier.nodes[-1], []
                ):
                    return True
        return False

    def _zero_address_validation(self, var: LocalVariable, node: Node, explored: List[Node]) -> bool:

        """
        Detects (recursively) if var is (zero address) checked in the function node
        """

        if node in explored:
            return False

        explored.append(node)

        # Heuristic: Assume zero address checked if variable is used within conditional or require/assert
        # TBD: Actually check for zero address in predicate
        if (node.contains_if() or node.contains_require_or_assert()) and (var in node.variables_read):

            return True

```

```
        # Check recursively in all the parent nodes
        for father in node.fathers:
            if self._zero_address_validation(var, father, explored):
                return True

        return False

    def _zero_address_validation_in_constructor(self, var: LocalVariable, constructor: Function) -> bool:
        if constructor:
            for node in constructor.nodes:
                if (node.contains_if() or node.contains_require_or_assert()) and (
                    var in node.variables_read
                ):
                    return True
            # Check recursively for parent constructors
            parent_contracts = constructor.contract.inheritance

            for parent_contract in parent_contracts:
                if self._zero_address_validation_in_constructor(var, parent_contract.constructor):
                    return True

        return False

    def _detect_missing_zero_address_validation(
        self, contract: Contract
) -> List[Union[Tuple[FunctionContract, DefaultDict[LocalVariable, List[Node]]]]]:
        """
        Detects if addresses are zero address validated before use.
        :param contract: The contract to check
        :return: Functions with nodes where addresses used are not zero address validated earlier
        """
```

```
        results = []

        for function in contract.functions_entry_points:
            var_nodes = defaultdict(list)

            for node in function.nodes:
                sv_addrs_written = [
                    sv
                    for sv in node.state_variables_written
                    if sv.type == ElementaryType("address")
                ]

                addr_calls = False
                for ir in node.irs:
                    if isinstance(ir, (Send, Transfer, LowLevelCall)):
                        addr_calls = True

                # Continue if no address-typed state variables are written and if no send/transfer/call
                if not sv_addrs_written and not addr_calls:
                    continue

                # Check local variables used in such nodes
                for var in node.local_variables_read:
                    # Check for address types that are tainted but not by msg.sender
                    if var.type == ElementaryType("address") and is_tainted(
                        var, function, ignore_generic_taint=True
                    ):
                        # Check for zero address validation of variable
                        # in the context of modifiers used or prior function context
                        if not (
                            self._zero_address_validation_in_modifier(
                                var, function.modifiers_statements
                            )
```

```
                                or self._zero_address_validation(var, node, [])
                                or self._zero_address_validation_in_constructor(var, function.contract.constructor)
                            ):
                                # Report a variable only once per function
                                var_nodes[var].append(node)
                    if var_nodes:
                        results.append((function, var_nodes))
                return results


    def _detect(self) -> List[Output]:
            """Detect if addresses are zero address validated before use.
            Returns:
                list: {'(function, node)'}
            """

            # Check derived contracts for missing zero address validation
            results = []
            for contract in self.compilation_unit.contracts_derived:
                missing_zero_address_validation = self._detect_missing_zero_address_validation(contract)
                for (_, var_nodes) in missing_zero_address_validation:
                    for var, nodes in var_nodes.items():
                        info = [var, " lacks a zero-check on ", ":\n"]
                        for node in nodes:
                            info += ["\t\t- ", node, "\n"]
                        res = self.generate_result(info)
                        results.append(res)
            return results


```

Output: [FIX]

We achieved in resolving the issue #981, From the below screenshot we can observe that “Missing-zero-check” detector able to check the arguments for constructor of parent contracts. Even if the parent contract's constructor has zero-access checks & the child contract is using the same variable

```
root@ubuntu-s-1vcpu-2gb-amd-nyc1-01:~/slither# slither Issue.sol
Compilation warnings/errors on Issue.sol:
Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX
-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source cod
e. Please see https://spdx.org for more information.
--> Issue.sol


INFO:Detectors:
Pragma version0.8.19 (Issue.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.
solc-0.8.19 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
ABC.owner3 (Issue.sol#16) should be immutable
Ownable.owner1 (Issue.sol#4) should be immutable
Ownable.owner2 (Issue.sol#5) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
INFO:Slither:Issue.sol analyzed (2 contracts with 85 detectors), 5 result(s) found
```

Risks:

Risks	Major/Minor and it's Probability and Exposure	Solution	Current Status
Installation of Slither (Configuration and setup)	Major risk P = 20% and E = 20 , so extra 4 hrs	For new users to reduce this risk, it's crucial that they thoroughly read the Slither team's documentation.	Completed
Unfamiliarity with the tool	Minor risk P = 30% and E = 10, so extra 3 hrs	It's crucial for new users to read the material given by the Slither team in order to lower this risk and watch the handson videos on youtube.	Completed
Installation of dependencies(Python)	Minor risk P = 15% and E = 10, so extra 1.5 hrs	Users face difficult in setting up compatable version of python to run slither.	Completed
Failure to meet iteration targets	Major risk P = 30% and E = 15 , so extra 4.5 hrs	The idea is to divide the work evenly and work together in a group meeting following each session or on weekends.	Completed
Logic failure	Major risk P = 30% and E = 20, so extra 6 hrs	The danger can be mitigated by thoroughly testing and debugging the code.	Completed

Customers and Users:

- Slither is used by **Block chain Developers** to run 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. [4]
- Slither helps automate security reviews for **Block chain organizations**. Slither provides an API to inspect Solidity code via custom scripts. We use this API to rapidly answer unique questions about the code we're reviewing. We have used Slither to:

    1. Identify code that can modify a variable's value.
    2. Isolate the conditional logic statements that are influenced by a particular variable's value.
    3. Find other functions that are transitively reachable as a result of a call to a particular function. [5]

- **Block chain developers** use slither to view high-level information about the contract using predefined printers.
- **Researchers** use its own intermediate representation, SlithIR, to build innovative vulnerability analyses on Solidity. It provides access to the CFG of the functions, the inheritance of the contracts, and lets you inspect Solidity expressions. [5]

Security researchers use slither to detect and describe security issues with underlying vulnerabilities, severity, and recommended fixes for our smart contract. [6]

Slither is most useful for the following 4 things: [7]

- Automated Vulnerability Detection: Easily detect vulnerabilities or security bugs in your code with low or no human effort.
- Automated optimization detection: Slither can detect code optimizations that the compiler misses while compiling.
- Slither can help you understand code better by summarizing and displaying contract information.
- Slither also helps with code reviews as its API can be easily interacted with by a user

Feedback from customer:

Egan Lobo(Solidity Beginner): It's great to observe that team has achieved their goals by resolving the project. I would like to take the moment to appreciate each person in the team for their continues contribution.

Narendra Mareddy (Junior Developer in Ethereum): The Logic used my team is accurate in resolving this issue in proper way on time, also I can see the design & functionality understandable for future users who read it.

References:

• [1] https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/

• [2] https://doi.org/10.48550/arXiv.1908.09878

• [3] https://doi.org/10.48550/arXiv.1809.02702

• [4] https://github.com/crytic/slither

• [5] https://blog.trailofbits.com/2018/10/19/slither-a-solidity-static-analysis-framework/

• [6] https://medium.com/coinmonks/automated-smart-contract-security-review-with-slither-1834e9613b01

• [7] https://www.linkedin.com/pulse/how-secure-smart-contracts-slither-damilare-d-fagbemi/?trk=pulse-article_more-articles_related-content-card

• [8] https://github.com/crytic/slither/issues/981

• [9] https://app.diagrams.net/

• [10] https://www.h-x.technology/blog/top-3-smart-contract-audit-tools

• [11] https://doi.org/10.48550/arXiv.1908.09878

• [12] https://doi.org/10.48550/arXiv.1809.02702

• [13] https://blog.trailofbits.com/2019/05/27/slither-the-leading-static-analyzer-for-smart-contracts/

• [14] https://github.com/crytic/slither#features

• [15] https://www.sciencedirect.com/topics/computer-science/inheritance-hierarchy