

# **CSE 6324 Advance Topics in Software Engineering**

## **Iteration 2 (Written Deliverable)**

### **Building Inherited Reentrancy Detector for Slither**

#### **Team 6**

Vaishnavi Khosla (1001906765)

Suresh Kavadi (1002040703)

Karthik Babu Vadloori (1002064678)

Danie Samanvitha Mellam (1002133148)

**Table of Contents:**

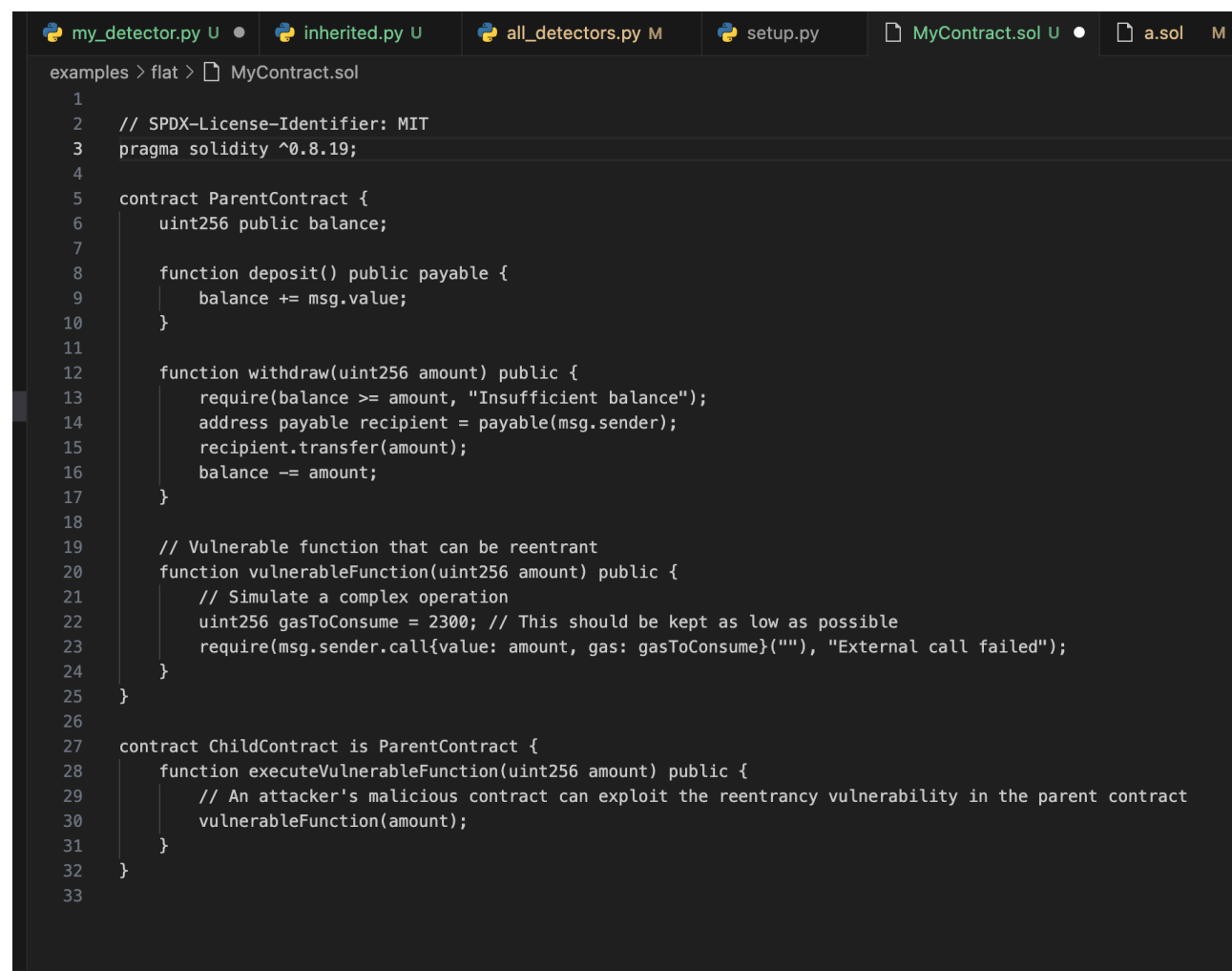
<b>Sr. No.</b>	<b>Title</b>
<b>1.</b>	Project Plan
<b>1.1</b>	Introduction
<b>1.2</b>	Goal
<b>1.3</b>	Scope
<b>1.4</b>	Tasks
<b>1.5</b>	Timeline
<b>1.6</b>	Resources
<b>2.</b>	Specification and Design
<b>3.</b>	Code and Tests
<b>4.</b>	Customers and Users
<b>5.</b>	References

## Project Plan

### Introduction:

When it comes to blockchain and smart contract development, inherited reentrancy flaws pose a serious security risk. Although these vulnerabilities are frequently disregarded, they have the potential to have disastrous effects, such as the unapproved withdrawal of cash and the interruption of contract operation. In order to maintain the integrity and security of blockchain-based systems, this project plan describes our methodical approach to finding, evaluating, and mitigating inherited reentrancy vulnerabilities in smart contracts.

In Slither, the standard Reentrancy Detector is enhanced by the special function of the Inherited Reentrancy Detector. The purpose of the Inherited Reentrancy Detector is to find reentrancy vulnerabilities that could spread through inheritance hierarchies, whereas the Reentrancy Detector is intended to find reentrancy flaws in specific functions within a smart contract.



```

examples > flat > MyContract.sol
1
2 // SPDX-License-Identifier: MIT
3 pragma solidity ^0.8.19;
4
5 contract ParentContract {
6     uint256 public balance;
7
8     function deposit() public payable {
9         balance += msg.value;
10    }
11
12    function withdraw(uint256 amount) public {
13        require(balance >= amount, "Insufficient balance");
14        address payable recipient = payable(msg.sender);
15        recipient.transfer(amount);
16        balance -= amount;
17    }
18
19    // Vulnerable function that can be reentrant
20    function vulnerableFunction(uint256 amount) public {
21        // Simulate a complex operation
22        uint256 gasToConsume = 2300; // This should be kept as low as possible
23        require(msg.sender.call{value: amount, gas: gasToConsume}(""), "External call failed");
24    }
25 }
26
27 contract ChildContract is ParentContract {
28     function executeVulnerableFunction(uint256 amount) public {
29         // An attacker's malicious contract can exploit the reentrancy vulnerability in the parent contract
30         vulnerableFunction(amount);
31     }
32 }
33

```

The code sample mentioned above:

The deposit and withdraw features of ParentContract allow for the manipulation of its balance. A reentrancy vulnerability is simulated by the vulnerableFunction in ParentContract. It permits the use of arbitrary amounts of gas in an external contract's external call, which could be exploited for malevolent purposes.

The function executeVulnerableFunction, which ChildContract inherits from ParentContract, invokes the parent contract's vulnerable function.

An attacker might use this to launch a malicious contract and use it to call executeVulnerableFunction repeatedly to deplete the balance of the parent contract. This introduces an inherited reentrancy vulnerability.

### **Project Goals:**

The primary goal of the Inherited Reentrancy Detector project is to develop a specialized tool that identifies inherited reentrancy vulnerabilities in smart contracts deployed on blockchain networks. The detector will focus on detecting vulnerabilities specifically inherited from parent contracts, providing proactive security measures, and raising awareness of the risks associated with reentrancy vulnerabilities.

The following objectives drive our project:

- Finding inherited reentrancy vulnerabilities in a range of smart contracts—including those that are descended from other contracts that include reentrancy vulnerabilities—is the goal of detection.
- Mitigation: Create and put into action plans to lessen the effects of inherited reentrancy vulnerabilities, stopping illegal money transfers and interruptions to contracts.
- Preventive measures include establishing best practices and recommendations for the construction of secure smart contracts to stop new inherited reentrancy vulnerabilities from being introduced.

**Project Scope:**

The primary scope of our Iteration 2 is to investigate, identify, and build inherited reentrancy custom detector capable of identifying reentrancy vulnerabilities inherited from parent contracts in smart contracts.

However, this Iteration focuses on the following comprehensive set of key aspects, that enhance Slither by **Specialized Detection, increasing Efficiency.**

In this phase of our Iteration 2, we have focused on working over the following Tasks:

- Research and understanding Inherited reentrancy vulnerabilities
- Root Cause Identification
- Building detector
- Documentation

**Tasks and Activities:****Task 1: Compiling Requirements and Research**

Research the effects of inherited reentrancy vulnerabilities in the actual world.  
Identify common patterns and code structures that lead to inherited reentrancy.

**Task 2: Designing Detectors**

Identify the input sources for the detector (such as the Solidity code and compiled contracts).  
Specify the output format (vulnerability reports, for example) for the detector.  
Create the user interface (if any) so that you may communicate with the detector.

**Task 3: Development of Detectors**

Put the detector's basic logic into practice to find vulnerabilities related to inherited reentrancy.  
Provide Solidity code parsing and analysis tools. Make the detector's command-line interface simple to use and intuitive. Use blockchain frameworks and libraries to communicate with Ethereum.

**Timeline:**

We have been working over the project, from 10/16/23 to 11/06/23 where all the team members actively involved in giving their ideas & expertise in building Inherited-Reentrancy-Detector

**Iteration 2:****Start: 10/16/23End: 11/06/23**

Task		Resource (Team Member)
<b>Requirements Analysis (2 days):</b>		
Day 1	Subtask 1	Suresh Kavadi
Day 2	Subtask 2	Karthik Babu Vadloori
<b>Design and Architecture (3 days):</b>		
Day 3	Subtask 1	Suresh Kavadi
Day 4	Subtask 2	Vaishnavi Khosla
Day 5	Evaluating Subtasks	Danie Samanvitha
<b>Development (4 days):</b>		
Day 6 and 7	Subtask 1	Suresh kavadi
Day 8 and 9	Subtask 2	Suresh kavadi
<b>Testing and Verification (2 days):</b>		
Day 10 and 11	Subtask 1 and 2	Suresh Kavadi
<b>Documentation (2 days):</b>		
Day 12	Building the structure, Gathering References andResources	Vaishnavi Khosla
Day 13	Written Deliverable Implementation	Suresh kavadi

**Resources:**

## Human Resources:

Vaishnavi Khosla (1001906765)

Suresh Kavadi (1002040703)

Karthik Babu Vadloori (1002064678)

Danie Samanvitha Mellam (1002133148)

## Technical Resources:

Slither

Solc compilers  
ChatGPT  
Visual Studio code  
Google docs

## Specification and Design

### Specification:

**Detector Name:** Inherited Reentrancy Detector

**Objective:** To identify and report inherited reentrancy vulnerabilities in Ethereum smart contracts that inherit from the Parent contract, providing actionable insights for developers and auditors to enhance contract security

### Design Outline

#### Detector Components

##### 1. Parser:

- **Purpose:** Extract Solidity source code or compiled contracts for analysis.
- **Functionality:** Parse and tokenize source code, extract contract inheritance relationships.

##### 2. Analyzer:

- **Purpose:** Analyze the inheritance hierarchy to identify potential reentrancy vulnerabilities.
- **Functionality:**
  - Traverse inheritance tree to detect contracts inheriting from **ReentrancyGuard**.
  - Analyze contract functions to identify those vulnerable to reentrancy.
  - Check for direct and indirect external calls in functions.
  - Flag vulnerable functions and provide context information.

##### 3. Reporting Module:

- **Purpose:** Generate clear and informative vulnerability reports.
- **Functionality:**
  - Create human-readable reports with detailed information on vulnerabilities.
  - Assign severity levels (e.g., critical, high, medium) to detected vulnerabilities.
  - Include suggestions and best practices for mitigating vulnerabilities.

#### Analysis Approach

##### 1. Contract Inheritance Analysis:

- Traverse the inheritance hierarchy of each contract to identify parent and child contracts.
- Log contracts inheriting from **Parent Contract** and their relationships.

**2. Function Analysis:**

- Examine each function in the contract to identify those that call external contracts.
- Analyze function parameters, return types, and the presence of external calls.

**3. Reentrancy Detection:**

- Check each function for potential reentrancy by examining if an external call precedes any state changes.
- Consider the sequence of operations to identify vulnerable patterns.

**4. Reporting and Recommendations:**

- Create detailed reports listing vulnerable functions, their contract relationships, and relevant code snippets.

**Output**

- The detector will produce detailed vulnerability reports with the following information:
  - Contract name and address (if applicable).
  - Vulnerable functions and their details.

Addition to this, our scope for Final Deliverables is to test & integrate this customized detector in slither community , once we test this, we will raise the pull request to add this to slither Github Repository

---

**Code and Tests**

Here is the code logic to for Inherited reentrancy detector-



```

slither > detectors > examples > inherited.py > ...
1  from slither.detectors.abstract_detector import AbstractDetector, DetectorClassification
2
3  class InheritedReentrancyDetector(AbstractDetector):
4      ARGUMENT = 'inherited-reentrancy-detector ReentrancyGuard' # Update to accept 'ReentrancyGuard' as an argument
5      HELP = 'Detect reentrancy vulnerabilities in contracts that inherit from ReentrancyGuard'
6      IMPACT = DetectorClassification.HIGH
7      CONFIDENCE = DetectorClassification.HIGH
8
9      WIKI = 'https://your-wiki-link.com'
10     WIKI_TITLE = 'Inherited Reentrancy Detector'
11     WIKI_DESCRIPTION = 'This detector identifies reentrancy vulnerabilities in contracts that inherit from ReentrancyGuard.'
12     WIKI_EXPLOIT_SCENARIO = 'An attacker can exploit reentrancy vulnerabilities to drain funds or disrupt contract behavior.'
13     WIKI_RECOMMENDATION = 'Follow best practices to prevent reentrancy vulnerabilities, such as using the Checks-Effects-Interacts pattern.'
14
15     def _detect(self):
16         findings = []
17
18         for contract in self.slither.contracts:
19             # Check if the contract inherits from 'ReentrancyGuard'
20             if any(inherited.name == 'ReentrancyGuard' for inherited in contract.inherits):
21                 for function in contract.functions:
22                     if self.calls_external_contracts(function):
23                         info = [
24                             f'Reentrancy vulnerability found in contract: {contract.name}',
25                             f'In function: {function.name}',
26                             'Recommendation: Ensure proper state changes are made before interacting with external contracts.'
27                         ]
28                         finding = self.generate_result(info)
29                         findings.append(finding)
30
31         return findings
32
33     def calls_external_contracts(self, function):
34         for callee in function.callees:
35             if callee.is_contract:
36                 return True
37         return False
38

```

### Reentrancy Detection Logic:

The core logic for identifying reentrancy vulnerabilities occurs within the "\_detect" method, particularly in the following key components:

- **Inheritance Check (Contract Inheritance):**
  - The detector identifies contracts that inherit from 'ReentrancyGuard.' This is a critical step to ensure that the code under analysis is subject to the reentrancy detection.
- **Function Analysis (Function Iteration):**
  - For contracts that inherit from 'ReentrancyGuard,' the detector proceeds to analyze each function within those contracts. Functions are iterated for inspection.
- **Reentrancy Detection (External Contract Calls):**
  - In the function analysis step, the detector employs the "calls\_external\_contracts" method to examine each function for calls to external contracts.
  - If a function is found to make calls to external contracts (as identified by "is\_contract" check), it is considered a potential point of reentrancy.
- **Finding Generation (Reporting):**
  - When a function that makes calls to external contracts is detected, a "finding" is generated. This finding contains information about the contract and function

where the vulnerability exists.

```

examples > flat > MyContract.sol
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract ReentrancyGuard {
5     bool private locked;
6
7     modifier nonReentrant() {
8         require(!locked, "Reentrant call detected");
9         locked = true;
10        _;
11        locked = false;
12    }
13 }
14
15 contract ParentContract is ReentrancyGuard {
16     uint256 public balance;
17
18     function deposit() public payable {
19         balance += msg.value;
20     }
21
22     function withdraw(uint256 amount) public nonReentrant {
23         require(balance >= amount, "Insufficient balance");
24         (bool success, ) = msg.sender.call{value: amount}("");
25         require(success, "Transfer failed");
26         balance -= amount;
27     }
28 }
29
30 contract ChildContract is ParentContract {
31     // This function inherits the reentrancy vulnerability from ParentContract
32     function executeVulnerableFunction(uint256 amount) public {
33         // An attacker's malicious contract can exploit the reentrancy vulnerability here
34         withdraw(amount);
35     }
36 }

```

Below Smart contract has multilevel inheritance it has

- **ReentrancyGuard** is a contract that provides reentrancy protection with the **nonReentrant** modifier.
- **ParentContract** inherits from **ReentrancyGuard** and has a **withdraw** function that is protected against reentrancy attacks.
- **ChildContract** further inherits from **ParentContract** and includes a function **executeVulnerableFunction**. This function inherits the reentrancy vulnerability from **ParentContract**. An attacker's malicious contract can exploit the reentrancy vulnerability by repeatedly calling **executeVulnerableFunction**, potentially draining the contract's balance.

**Output: for the above smart contract**

```

(sliether-dev) sureshkavadi@sureshs-MacBook-Air slither % slither --detect reentrancy --examples/flat/MyContract.sol
'solc --version' running
'solc examples/flat/MyContract.sol --combined-json abi,ast,bin,bin-runtime,srcmap,srcmap-runtime,userdoc,devdoc,hashes --allow-paths ../Users/sureshkavadi/sliether/examples/flat' running
INFO:Detectors:
Reentrancy in ParentContract.withdraw(uint256) (examples/flat/MyContract.sol#22-27):
  External calls:
  - (success) = msg.sender.call{value: amount}() (examples/flat/MyContract.sol#24)
  State variables written after the call(s):
  - balance -= amount (examples/flat/MyContract.sol#26)
  ParentContract.balance (examples/flat/MyContract.sol#16) can be used in cross function reentrancies:
  - ParentContract.balance (examples/flat/MyContract.sol#16)
  - ParentContract.deposit() (examples/flat/MyContract.sol#18-20)
Reference: https://github.com/crytic/sliether/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Slither:examples/flat/MyContract.sol analyzed (3 contracts with 1 detectors), 1 result(s) found
(sliether-dev) sureshkavadi@sureshs-MacBook-Air slither %

```

Here's the explanation of the Slither output:

1. **Reentrancy in ParentContract.withdraw(uint256):**
  - Slither has detected a reentrancy vulnerability in the **ParentContract** within the **withdraw** function, which takes an unsigned integer parameter.
  - The vulnerability is associated with an external call made within the **withdraw** function.
2. **External calls:**
  - Slither identifies an external call made with **msg.sender.call{value: amount}()**. This is a call to an external contract.
3. **State variables written after the call(s):**
  - After the external call, the **balance** state variable is modified with **balance -= amount**.
4. **ParentContract.balance can be used in cross-function reentrancies:**
  - Slither identifies that **ParentContract.balance** is being manipulated in a way that it can be used in cross-function reentrancies.
  - The reference mentions that **ParentContract.balance** can be accessed and manipulated, potentially by malicious external contracts.
  - It also identifies that the **ParentContract.deposit()** function is part of the vulnerability.

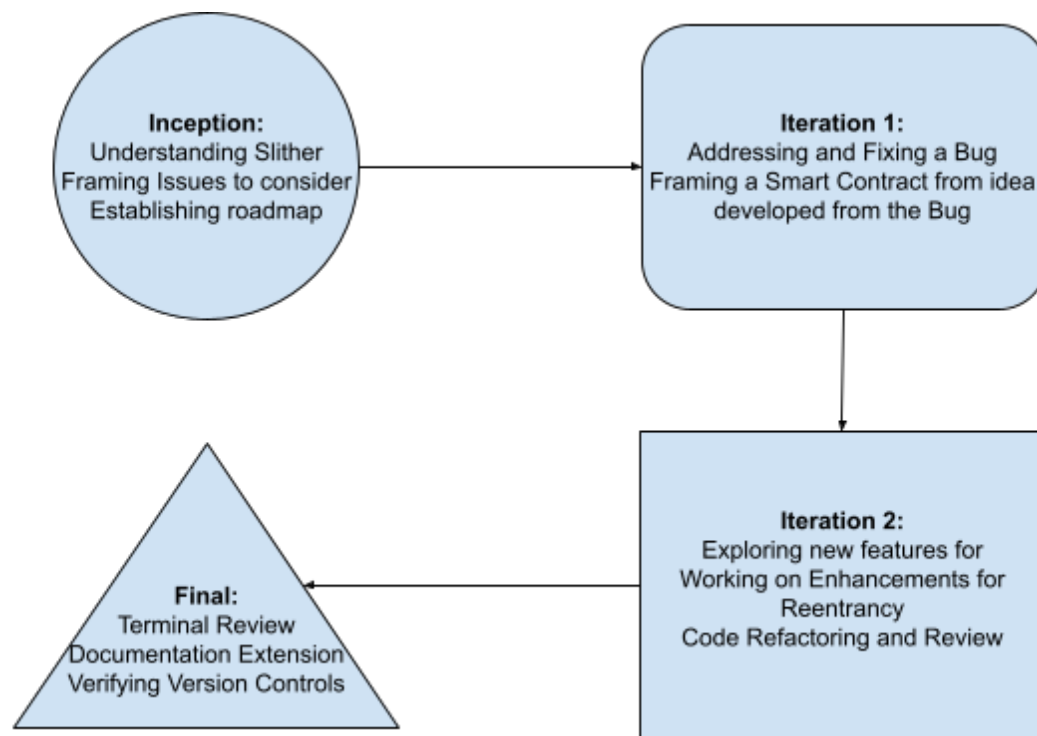
### Conclusion:

The analysis of the provided Solidity smart contract code using Slither reveals a reentrancy vulnerability in the **ParentContract**. Specifically, the vulnerability is detected within the **withdraw** function, which allows external calls and manipulates the **balance** state variable. This reentrancy vulnerability makes the contract susceptible to malicious external contracts that can exploit this weakness.

### Future Enhancements:

In the Final deliverables we plan to test this detector on list of smart contracts, & Integrate it in slither GithubRepository with Pull request and Document the step-by-step procedure how we build the detector

The following is a diagram of our summarized plan for future developments:



## Customers and Users

### Blockchain Programmers:

The main users of these detectors are blockchain developers who create and implement smart contracts on blockchain platforms. To find and fix inherited reentrancy problems in their code, they employ these techniques.

### Auditors for Smart Contracts:

Inheritance reentrancy detectors are used by skilled smart contract auditors and security specialists during their auditing and security evaluation procedures. They assess the security of smart contracts for their clients using these techniques.

### Blockchain-Based Security Firms:

Businesses that specialize in blockchain security may give security evaluation services to blockchain projects by utilizing inherited reentrancy detectors. They can recognize and address weaknesses in the projects they evaluate.

**Platforms for Smart Contract Governance:**

Inheritable reentrancy detectors can be integrated by platforms or services that provide governance over blockchain projects to automate security audits and guarantee that newly submitted code is secure.

**Institutions of Education:**

These detectors could be incorporated into the curricula of academic institutions that offer courses on blockchain and smart contract creation to teach students safe coding techniques.

**Open-Source Group:**

These detectors can be used by contributors and developers working on open source blockchain projects to increase the security of those projects.

---

**References:**

[Github: Slither]

<https://github.com/crytic/slither>

[TrailofBits Blog] "Slither: The Leading Static Analyzer for Smart Contracts"

<https://blog.trailofbits.com/2019/05/27/slither-the-leading-static-analyzer-for-smart-contracts/>

Multiple Inheritance in Solidity

<https://coinsbench.com/multiple-inheritance-in-solidity-835c02eb58cb>

Team 6 Github Repository:

[https://github.com/Suresh-uta/ASE\\_CSE\\_6324](https://github.com/Suresh-uta/ASE_CSE_6324)











