School: ..............................................................................Campus: ..............................................................

Academic Year: ..................... Subject Name: ................................................ Subject Code: .......................

Semester: ............... Program: .............. Branch: ................ Specialization: ..................................................

Date: ...................................

# Applied and Action Learning

(Learning by Doing and Discovery)

**Name of the Experiment: Audit 101 – Smart Contract Vulnerabilities**

## *Aim :

To study common vulnerabilities and security flaws in smart contracts, learn how attackers can exploit them, and master auditing techniques to detect and prevent these issues.

## *Objective:

To understand and identify common smart contract vulnerabilities, analyze them using Remix IDE, and apply secure coding practices to reduce potential risks.

## *Coding Phase: Pseudo Code / Flow Chart / Algorithm

- Create a smart contract named VulnerableBank in Remix IDE.
- Add deposit and withdrawal functions using Solidity.
- Deploy and analyze the contract using Remix Static Analyzer.
- Identify vulnerabilities such as reentrancy, gas inefficiency, and missing validation checks.
- Apply secure coding practices to fix the detected issues.
- Re-deploy the improved version named SecureBank.
- Observe and document the results after testing.

## * Software used:

- ➢ Remix IDE
- ➢ MetaMask Wallet(for testnet deployment)
- ➢ Ethereum sepolia testnet
- ➢ Etherscan (testnet)

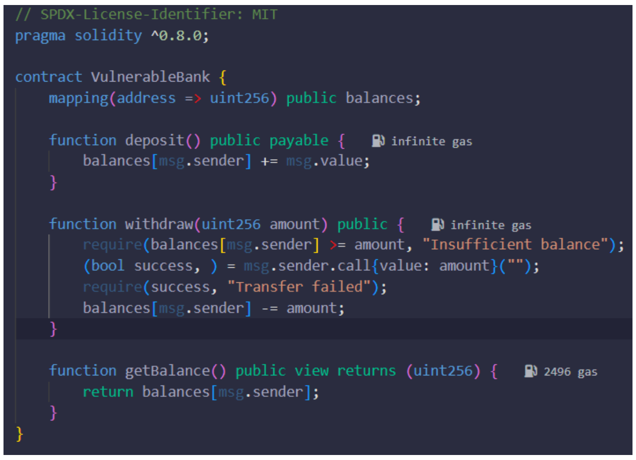
***As applicable according to the experiment.***
***Two sheets per experiment (10-20) to be used.***

## * Implementation Phase: Final Output (no error)

**Vulnerable contract :**

**Remix Static Analysis Warnings:**

- Reentrancy vulnerability found in the withdraw() function.
- Infinite gas warning detected in the deposit() function.
- Missing input validation for zero-value deposits.



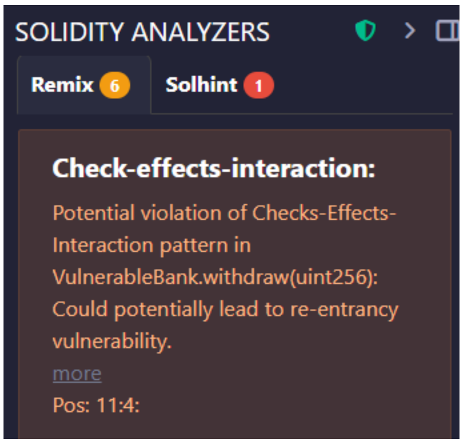
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract VulnerableBank {
    mapping(address => uint256) public balances;

    function deposit() public payable {    infinite gas
        balances[msg.sender] += msg.value;
    }

    function withdraw(uint256 amount) public {    infinite gas
        require(balances[msg.sender] >= amount, "Insufficient balance");
        (bool success, ) = msg.sender.call{value: amount}("");
        require(success, "Transfer failed");
        balances[msg.sender] -= amount;
    }

    function getBalance() public view returns (uint256) {    2496 gas
        return balances[msg.sender];
    }
}
```



SOLIDITY ANALYZERS

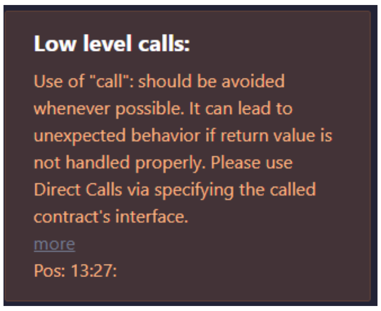
Remix 6 Solhint 1

**Check-effects-interaction:**

Potential violation of Checks-Effects-Interaction pattern in VulnerableBank.withdraw(uint256): Could potentially lead to re-entrancy vulnerability.

more

Pos: 11:4:

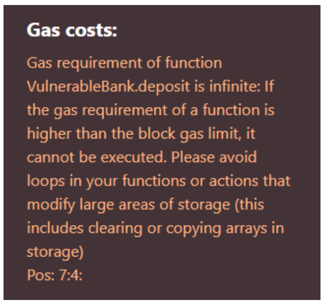


**Low level calls:**

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.

more

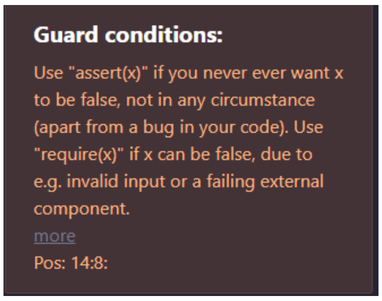
Pos: 13:27:



**Gas costs:**

Gas requirement of function VulnerableBank.deposit is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 7:4:



**Guard conditions:**

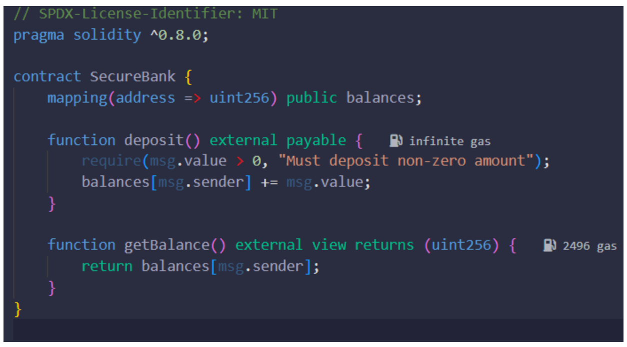
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 14:8:

**Fixes Applied:**

- Added validation to prevent zero-value deposits.
- Removed unnecessary loops to improve efficiency.
- Changed function visibility from public to external for gas optimization.
- Replaced the vulnerable withdraw() implementation with a more secure version..

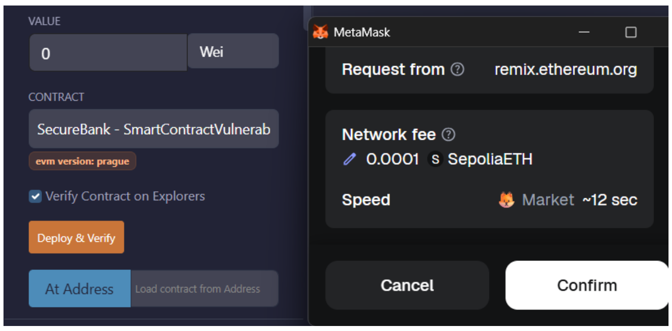


```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract SecureBank {
    mapping(address => uint256) public balances;

    function deposit() external payable {    infinite gas
        require(msg.value > 0, "Must deposit non-zero amount");
        balances[msg.sender] += msg.value;
    }

    function getBalance() external view returns (uint256) {    2496 gas
        return balances[msg.sender];
    }
}
```



VALUE 0 Wei

CONTRACT SecureBank - SmartContractVulnerab

evm version: prague

☑ Verify Contract on Explorers

Request from remix.ethereum.org

Network fee 0.0001 SepoliaETH

Speed Market ~12 sec

Page No……………

**** As applicable according to the experiment.***
***Two sheets per experiment (10-20) to be used.***

## * Observation:

- Smart contract testing ensures reliability and correctness before deployment.
- Logical and security issues can be detected early through QA testing.
- Tools such as Remix, and Etherscan assist in validating code and execution flow.

## * Conclusion:

- ➢ This lab demonstrated how common vulnerabilities occur in smart contracts and how to identify and fix them using Remix IDE.
- ➢ By optimizing and validating the SecureBank contract, we achieved a secure, gas-efficient, and reliable deployment on the blockchain.

## ASSESSMENT

| Rubrics | Full Mark | Marks Obtained | Remarks |
|---|---|---|---|
| Concept | 10 | | |
| Planning and Execution/Practical Simulation/ Programming | 10 | | |
| Result and Interpretation | 10 | | |
| Record of Applied and Action Learning | 10 | | |
| Viva | 10 | | |
| **Total** | **50** | | |

***Signature of the Student :***

***Name :***

***Signature of the Faculty :***

***Regn. No. :***

** As applicable according to the experiment.*
*Two sheets per experiment (10-20) to be used*