



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, understand how they can be exploited, and learn the methods used in smart contract auditing to detect and prevent them.

#### \*Objective:

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

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

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

#### \* Software used:

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

Page No.....

**\* 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 in withdraw().
- Infinite gas warning for deposit() function.
- No 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 Solhint

**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 non-zero deposit validation.
- Removed unnecessary loops.
- Changed public to external to optimize gas.
- Removed vulnerable withdraw() implementation for security focus.

```
// 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  
Deploy & Verify  
At Address Load contract from Address

MetaMask  
Request from remix.ethereum.org  
Network fee   
0.0001 s SepoliaETH  
Speed Market ~12 sec  
Cancel Confirm

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 or security bugs can be detected early through QA testing.
- Tools like Remix, MythX, and Etherscan help validate code and execution flow.

### \* Conclusion:

- This lab demonstrated how common vulnerabilities arise in smart contracts and how to identify and fix them using Remix IDE.
- Through the optimization and validation of the SecureBank contract, we ensured 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		
<b>Total</b>	<b>50</b>		

***Signature of the Student :***

***Name :***

***Signature of the Faculty :***

***Regn. No. :***

Page No.....

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