Centurion University
Shaping Lives... Empowering Communities...

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 Experiement : Peer Audit – Contract Security Review

Objective/Aim:

To analyze and review a Solidity smart contract for potential **security vulnerabilities** and understand best practices in **secure smart contract development**.

Apparatus/Software Used:

- **VS Code** (IDE for editing Solidity code)
- **Hardhat / Remix IDE** (for compiling and testing contracts)
- **MetaMask** (Ethereum wallet)
- **Test Network** (Hardhat local node or Sepolia)
- **Public API** (e.g., Weather API, Crypto Price API) – to simulate real-world data.

Theory:

Smart contracts are **immutable** once deployed on the blockchain, so **security audits** are essential before deployment.
A **security audit** is a detailed review of contract code to detect vulnerabilities like:

- Reentrancy attacks
- Integer overflows/underflows
- Unchecked external calls
- Access control flaws
- Denial of Service (DoS)
- Front-running or timestamp dependence

Common Security Practices

1. Use the **latest Solidity version**.
2. Implement **ReentrancyGuard** for external calls.
3. Validate **input parameters**.
4. Use **require()** and **assert()** properly.
5. Use **OpenZeppelin** libraries for safe math and access control.

Procedure:

1. **Start** the process of contract auditing.
2. **Open** the smart contract code in Remix IDE or Hardhat environment.
3. **Compile** the contract to check for syntax or version errors
4. **Run Static Analysis** using Remix's Analysis tool or external tools (Slither, Mythril).
5. **Identify warnings and vulnerabilities** such as:

- Reentrancy
- Integer overflow/underflow
- Unchecked external calls
- Access control flaws

6. **Perform manual code review** to analyze logic, variable visibility, and modifier usage.
7. **Document all detected issues** with their severity (Low/Medium/High).
8. **Apply security fixes** following best practices like:

- Checks-Effects-Interactions pattern
- Using require() for validations
- Adding ReentrancyGuard

9. **Re-run analysis** to verify that all major vulnerabilities are resolved.
10. **Compile and redeploy** the corrected contract.
11. **End** the audit process after verification.

Observation:

1. The contract compiled successfully without syntax errors.
2. Static analysis detected **warnings related to low-level call() usage**.
3. The withdraw() function updated the state **after** sending Ether — making it **vulnerable to reentrancy attacks**.
4. No overflow or underflow issues were found (since Solidity 0.8+ includes automatic checks).
5. Visibility of functions (public, private) was correctly defined.
6. The contract lacked **access control** or an owner modifier for restricted actions.
7. After applying the **Checks-Effects-Interactions** pattern, the reentrancy issue was resolved.
8. Implementing **ReentrancyGuard** further secured external calls.
9. Re-analysis showed **no remaining critical vulnerabilities**.
10. The contract functioned correctly after fixes — deposits and withdrawals worked safely.

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.