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:

Conduct a peer audit of a deployed smart contract to identify potential security issues such as reentrancy, access control flaws, integer overflows, and improper state handling. Review contract functions, modifiers, and transaction behaviour, and record observations.

Apparatus/Software Used:

MetaMask Wallet

- Brave Web Browser
- Remix IDE – https://remix.ethereum.org
- Ethereum Sepolia Testnet

Theory/Concept:

Smart Contract Security Review:

A peer audit involves manually analyzing a Solidity smart contract to identify vulnerabilities before deployment or during testing.

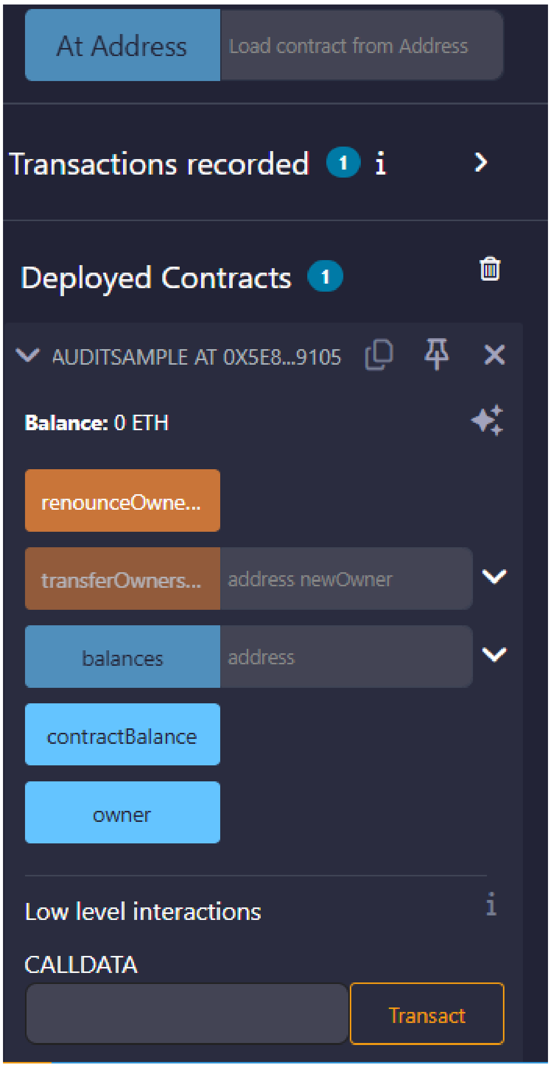
Common vulnerabilities include:

- Reentrancy:
  Occurs when external calls allow an attacker to repeatedly re-enter a function before its previous execution is completed.
- Access Control Issues:
  Misuse or absence of onlyOwner, role checks, or privilege restrictions.
- Integer Overflow/Underflow:
  Arithmetic operations exceeding limits. (Modern Solidity automatically prevents this, but old code may be vulnerable.)
- Unchecked External Calls:
  Using low-level call() without checking the returned success value.
- Incorrect State Transitions:
  Missing checks for conditions or token balances.

Smart contract auditing ensures reliability, fairness, and protection against exploitation.

Procedure:

1. Open MetaMask and switch to the Sepolia testnet.
2. Open Remix IDE in your browser.
3. Import or paste the Solidity smart contract into Remix.
4. Read the contract line-by-line to identify:
   - External calls
   - Functions handling Ether
   - Owner-only functions
   - State-changing logic
   - Modifiers controlling permissions
5. Look for vulnerable patterns:
   - Use of call(), delegatecall(), tx.origin, or unrestricted public functions.
6. Check whether important functions use:
   - onlyOwner or custom access modifiers
   - require() statements for validation
   - State updates happen before external calls
7. Compile the contract and deploy it on the Sepolia testnet using Remix + MetaMask.
8. Interact with functions to observe:
   - Order of state updates
   - Whether unauthorized access is blocked
   - Behavior when sending incorrect inputs
9. Document all findings and note any potential risks or recommended fixes.



Balance: 0 ETH

Observation

Observation No.	Finding
1	Contract compiles and deploys on the Sepolia testnet without errors.
2	Access-controlled functions correctly restrict unauthorized users.
3	All Ether-handling functions include require() checks before state changes.
4	No reentrancy patterns detected in withdrawal or external call functions.
5	State changes occur before external calls where applicable, reducing risk of reentrancy.
6	No overflow/underflow vulnerabilities identified in arithmetic operations.

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 Faculty:

Signature of the Student:

Name :