Security Audit Report

Date: November 23, 2024
Name: Harsh Akar
Roll no:12
Reg. no:12106155
Contract: Token Transfer and Withdrawal System

Question:4

```
function transfer(address to, uint amount) external {
  if (balances[msg.sender] >= amount) {
    balances[to] += amount;
    balances[msg.sender] -= amount;
  }
}

function withdraw() external {
  uint256 amount = balances[msg.sender];
  (bool success,) = msg.sender.call{value: balances[msg.sender]}("");
  require(success);
  balances[msg.sender] = 0;
}
```

Q:
Imagine you are doing a manual audit and you come across above code. Write a comprehensive report explaining the issue and the fix for the issue.

ANSWER:

Detailed Findings

1. Critical Re-entrancy Vulnerability

Location: withdraw() function Current
Implementation: solidity function

```
withdraw() external {
   uint256 amount = balances[msg.sender];
   (bool success,) = msg.sender.call{value: balances[msg.sender]}("");
require(success);     balances[msg.sender] = 0;
}
```

Issue: The function updates the user's balance *after* sending funds, creating a window for re-entrancy attacks. An attacker could recursively call the withdraw function before their balance is set to zero.

Real-world Impact:

- If exploited, an attacker with just 100 tokens could potentially drain the entire contract
- All user funds would be at risk
- The contract would likely need to be deprecated

Attack Scenario: Let's say Alice is our attacker. She:

1. Deposits 100 tokens
2. Creates a malicious contract with a fallback function that calls withdraw()
3. Initiates the attack
4. Before her balance is set to 0, she can withdraw multiple times
5. Result: She could withdraw far more than her initial 100 tokens

2. Transfer Function Vulnerabilities

The current transfer function also has several security gaps:

```
solidity  function transfer(address to, uint amount)
external {
```

```
if (balances[msg.sender] >= amount) {
balances[to] += amount;
balances[msg.sender] -= amount;
   }
}
```

Issues Found:

- No validation for zero-address transfers
- Missing event logs
- Silent failures
- Potential overflow risks

Recommended Solutions

Fix Re-entrancy:

```
function withdraw() external {        uint256 amount =
balances[msg.sender];     balances[msg.sender] = 0; //
Update first!     (bool success,) =
msg.sender.call{value: amount}("");     require(success,
"Withdrawal failed");
}
```

2. Add Re-entrancy Guard: solidity

```
contract ReentrancyGuard {
   bool private locked;

   modifier noReentrant() {
```

```
require(!locked, "No re-entrancy");       locked
= true;
      _;
      locked = false;
   } }
```

Improve Transfer Function:

```
function transfer(address to, uint amount) external returns (bool) {     require(to
!= address(0), "Invalid recipient");
   require(balances[msg.sender] >= amount, "Insuiicient balance");

   balances[msg.sender] -= amount;
   balances[to] += amount;

   emit Transfer(msg.sender, to, amount);
   return true;
}
```

Action Items Checklist

- Implement Checks-Eiects-Interactions pattern
- Add ReentrancyGuard
- Add event logging
- Implement input validation
- Add emergency pause functionality
- Conduct thorough testing after fixes

Conclusion

The identified vulnerabilities pose an immediate risk to user funds. I strongly recommend implementing these fixes before any further deployment or usage of the contract.