Smart Contract Security Audit Report

VulnerableAirdrop Contract Analysis

SECURITY AUDIT REPORT

VulnerableAirdrop Smart Contract

This report identifies critical security vulnerabilities in the VulnerableAirdrop contract and provides actionable recommendations to mitigate risks.

CRITICAL FINDINGS IDENTIFIED

CRITICAL: Denial of Service via Block Gas Limit

The distribute() function will fail when processing more than ~1,150 recipients due to Ethereum block gas limits. This would permanently lock all funds in the contract.

Recommendation: Implement batch processing with a configurable batchSize parameter as shown in the fixed code example.

HIGH RISK: Potential Reentrancy Vulnerability

The contract changes state after making external calls (transfer()), violating checks-effects-interactions pattern. While transfer() has gas limits, this remains dangerous.

Recommendation: Always update state before making external calls. Move hasReceived update before the transfer call.

Smart Contract Security Audit Report

VulnerableAirdrop Contract Analysis

Contract Code Analysis

The original vulnerable contract contains two critical security issues that could lead to fund loss or contract paralysis:

Original Vulnerable Code:

```
contract VulnerableAirdrop {
    address[] public recipients;
    mapping(address => bool) public hasReceived;

    function addRecipients(address[] memory _recipients) public {
        for(uint i = 0; i < _recipients.length; i++) {
            recipients.push(_recipients[i]);
        }
    }

    function distribute() public {
        for(uint i = 0; i < recipients.length; i++) {
            if(!hasReceived[recipients[i]]) {
                payable(recipients[i]).transfer(1 ether);
                hasReceived[recipients[i]] = true;
            }
        }
    }
}
```

Smart Contract Security Audit Report

VulnerableAirdrop Contract Analysis

Recommended Secure Implementation

The fixed implementation addresses all critical vulnerabilities while maintaining the same functionality:

```
contract SafeAirdrop {
    address[] public recipients;
    mapping(address => bool) public hasReceived;
    uint256 public currentIndex;

    function addRecipients(address[] memory _recipients) public {
        for(uint i = 0; i < _recipients.length; i++) {
            recipients.push(_recipients[i]);
        }
    }

    function distribute(uint256 batchSize) public {
        uint256 end = currentIndex + batchSize;
        if (end > recipients.length) end = recipients.length;

        for(uint i = currentIndex; i < end; i++) {
            if(!hasReceived[recipients[i]]) {
                hasReceived[recipients[i]] = true;  // State change first
                payable(recipients[i]).transfer(1 ether);
            }
        }
        currentIndex = end;
    }
}
```

Key Improvements:

1. Batch processing prevents gas limit issues
2. State changes before external calls prevent reentrancy
3. Progress tracking allows partial distributions
4. Configurable batch size for flexibility