■ SOLID SOURCE SYSTEMS

Smart Contract Security Audit Report

Client: [Redacted]

Contract: SampleToken.sol

Version: 1.0.0

Date: January 2025

SAMPLE AUDIT REPORT (Redacted for Demonstration)

contact@solidsourcesystems.com

solidsourcesystems.com

Executive Summary

This audit identified multiple security vulnerabilities, including a critical reentrancy issue that could lead to complete loss of funds.

All critical and high-severity issues have been addressed and verified. The contract is now considered safe for deployment following remediation.

Audit Target	SampleToken.sol
Language	Solidity 0.8.20
Lines of Code	~450
Audit Duration	5 days
Methods	Manual review, Slither, Aderyn
Final Status	**PASSED (after remediation)**

Findings Summary

Severity	Count	Status
Critical	1	Fixed ✓
High	2	Fixed ✓
Medium	2	Fixed ✓
Low	3	Acknowledged
Informational	4	Acknowledged

Audit Scope

The following defines the scope and boundaries of this security audit.

Contracts Reviewed:

- SampleToken.sol (main token contract)
- Staking.sol (staking rewards module)

Repository:

- Commit hash: abc123def456 (audited version)
- Commit hash: xyz789ghi012 (remediation verified)

Out of Scope:

- Frontend/UI code
- Deployment scripts
- Third-party dependencies (OpenZeppelin contracts)
- Off-chain infrastructure

Methodology:

- Manual line-by-line code review
- Static analysis (Slither, Aderyn)
- Foundry-based unit test review
- Business logic and access control analysis

Detailed Findings

[C-01] Reentrancy Vulnerability in withdraw()

CRITICAL

Location: SampleToken.sol, Line 142

Description:

The withdraw() function updates the user's balance after sending ETH, allowing an attacker to re-enter the function and drain funds before the balance is updated.

> **Business Impact:** An attacker could repeatedly call withdraw() to drain ALL ETH from the contract, resulting in **complete loss of user funds**. This is an exploitable vulnerability that requires immediate remediation before deployment.

Vulnerable Code:

```
function withdraw(uint256 amount) external {
    require(balances[msg.sender] >= amount);
    (bool success, ) = msg.sender.call{value: amount}("");
    require(success);
    balances[msg.sender] -= amount; // Updated AFTER transfer
}
```

Recommendation:

Follow the checks-effects-interactions pattern. Update state before making external calls:

```
function withdraw(uint256 amount) external {
    require(balances[msg.sender] >= amount);
    balances[msg.sender] -= amount; // Update BEFORE transfer
    (bool success, ) = msg.sender.call{value: amount}("");
    require(success);
}
```

Status: ✓ Fixed in commit abc123

[H-01] Missing Access Control on setFee()

HIGH

Location: SampleToken.sol, Line 89

Description:

The setFee() function lacks access control, allowing any user to modify the protocol fee.

> **Business Impact:** An attacker could set the transfer fee to 100%, effectively **stealing all tokens** transferred through the contract. This would result in immediate user fund loss and total loss of protocol trust.

Recommendation:

Add the onlyOwner modifier to restrict access to authorized addresses only.

Status: ✓ Fixed in commit def456

[H-02] Unchecked External Call Return Value

HIGH

Location: Staking.sol, Line 67

Description:

The claimRewards() function does not check the return value of the token transfer, which could silently fail leaving users unable to claim their earned rewards.

> **Business Impact:** Users may believe they have claimed rewards when the transfer actually failed. This leads to **lost user funds** and support tickets, damaging protocol reputation and user trust.

Recommendation:

Use SafeERC20's safeTransfer() or explicitly check the return value and revert on failure.

Status: ✓ Fixed in commit ghi789

Conclusion

This audit identified **one critical** and **two high-severity** vulnerabilities that could have resulted in complete loss of user funds if exploited. The most severe issue—a reentrancy vulnerability in the withdraw function—would have allowed an attacker to drain the entire contract balance.

All critical and high-severity issues have been addressed by the development team and verified in a follow-up review. The contract is now considered **safe for deployment**.

Recommendation	Status
Deploy to testnet for final integration testing	Pending
Consider formal verification for core functions	Optional
Implement monitoring for unusual activity	Recommended
Establish bug bounty program post-launch	Recommended

Disclaimer