Smart Contract Security Audit

for

stEDU ERC4626 Protocol

Audit Date: July 17, 2025

Audit Version: v1.0

Contents

1 Executive Summary **2**
1.1 Audit Scope . . . 2
1.2 Project Summary . . . 2
1.3 Risk Statistics . . . 2
1.4 Audit Approach Summary . . . 3

2 Detailed Audit Findings **4**
2.1 High Risk Vulnerabilities . . . 4
2.2 Medium Risk Vulnerabilities . . . 5

3 Summary and Recommendations **9**
3.1 Key Findings Summary . . . 9
3.2 Overall Security Recommendations . . . 9
3.3 Positive Security Features . . . 9
3.4 Fix Verification . . . 9

4 Disclaimer **10**

1 Executive Summary

This report presents a comprehensive security audit of the stEDU ERC4626 Protocol smart contract system. The audit identified multiple security vulnerabilities, including 1 high-risk vulnerability, 4 medium-risk vulnerabilities, and 2 low-risk vulnerabilities. The main issues are concentrated in centralization risks, incorrect asset accounting, complex loop operations, and access control concerns.

The codebase exhibits solid overall quality with proper reentrancy protection and time-lock mechanisms, demonstrating good security practices in most areas.

1.1 Audit Scope

This audit covers the following smart contracts:

- stEDU.sol - Main staking contract implementing ERC4626
- wstEDU.sol - Wrapper contract for stEDU tokens

1.2 Project Summary

Type	DeFi Staking Protocol
Arch	Ethereum
CodeBase	stEDU-erc4626
CommitID	8e5b07b
FixCommit	9767fb6
Chain	Ethereum Chain
Audit Method	Cross-manual review; Static Analysis

1.3 Risk Statistics

Risk Level	Count	Color	Status
Critical	0	**Critical**	-
High	1	**High**	Fixed
Medium	3	**Medium**	Fixed
Low	0	**Low**	-
Total	**4 Vulnerabilities Identified**		

Table 1: Vulnerability risk level distribution

1.4 Audit Approach Summary

This report has been prepared for the stEDU ERC4626 Protocol to discover issues and vulnerabilities in the source code as well as any contract dependencies. A comprehensive examination has been performed, utilizing Cross Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors
- Assessing the codebase to ensure compliance with current best practices and industry standards
- Ensuring contract logic meets the specifications and intentions of the client
- Cross referencing contract structure and implementation against similar smart contracts
- Thorough line-by-line manual review of the entire codebase
- Focus on ERC4626 standard compliance and staking mechanism security

The security assessment resulted in findings that ranged from high to low risk. We recommend addressing these findings to ensure a high level of security standards.

2 Detailed Audit Findings

2.1 High Risk Vulnerabilities

High Risk

Vulnerability ID: STEDU-001
Title: Centralization Risk - Owner has excessive control over rewards
Risk Level: High
Status: Not Fixed
Affected Contract: stEDU.sol

Description: The contract owner has excessive control over the staking rewards through the `depositRewards` function. The owner can arbitrarily manipulate the index (share price) by depositing rewards, which directly affects all stakers' returns. This creates a significant centralization risk.

```
/// @notice Push new EDU rewards; boosts `index`. Only callable by owner/treasury.
function depositRewards() external payable onlyOwner whenNotPaused nonReentrant {
    require(msg.value > 0, "No reward sent");
    require(totalSupply() > 0, "Nothing staked");

    wedu.deposit{value: msg.value}();
    uint256 deltaIndex = (msg.value * 1e18) / totalSupply();
    index += deltaIndex;
    emit RewardsDeposited(msg.sender, msg.value, index);
}
```

Figure 1: Owner control over reward distribution

Technical Details:

- The `depositRewards` function is restricted to owner only
- Owner can manipulate the index calculation: `index += (msg.value * 1e18) / totalSupply()`
- No time locks or multi-signature requirements for reward distribution
- Users have no protection against malicious reward manipulation

Impact Assessment:

- Owner can artificially inflate or deflate staking rewards
- Potential for rug pull or unfair reward distribution
- Users may lose funds due to index manipulation
- Breaks trust in the decentralized nature of the protocol

Remediation Recommendations:

- Implement multi-signature wallet for owner functions
- Add time locks for critical operations
- Consider decentralized governance for reward distribution
- Implement maximum reward limits per period

2.2 Medium Risk Vulnerabilities

Medium Risk

Vulnerability ID: STEDU-002
Title: Complex array operations may cause gas limit issues
Risk Level: Medium
Status: Not Fixed
Affected Contract: stEDU.sol

Description: The `unstake` function performs complex array operations on the `_deposits` mapping without gas limit protection. Users with many small deposits may face transaction failures due to gas limit exhaustion.

```
function unstake(uint256 shares) external whenNotPaused nonReentrant returns (uint256) {
    require(shares > 0, "Zero shares");
    require(balanceOf(msg.sender) >= shares, "Insufficient stEDU");

    uint256 unlocked = 0;
    DepositInfo[] storage buckets = _deposits[msg.sender];
    uint256 len = buckets.length;
    for (uint256 i = 0; i < len && unlocked < shares; ) {
        DepositInfo storage b = buckets[i];
        if (block.timestamp - b.timestamp >= UNSTAKE_DELAY) {
            uint256 take = b.shares;
            if (unlocked + take > shares) take = shares - unlocked;
            b.shares -= take;
            unlocked += take;
            if (b.shares == 0) {
                buckets[i] = buckets[buckets.length - 1];
                buckets.pop();
                len--; continue; // inspect element that got swapped in
            }
        }
        unchecked { ++i; }
    }
    require(unlocked == shares, "Requested amount still locked");

    _burn(msg.sender, shares);
    uint256 eduToReturn = (shares * index) / 1e18;
    wedu.withdraw(eduToReturn);
    (bool sent, ) = msg.sender.call{value: eduToReturn}("");
    require(sent, "Transfer failed");

    emit Unstaked(msg.sender, eduToReturn, shares);
    return eduToReturn;
}
```

Figure 2: Complex array operations in unstake function

Technical Details:

- Loop iteration count depends on user's deposit history
- Array restructuring operations within the loop
- No maximum iteration limits implemented
- Gas costs increase with number of deposits

Remediation Recommendations:

- Implement maximum iterations per transaction
- Add pagination for large unstaking operations

- Consider alternative data structures for deposit tracking
- Implement emergency withdrawal mechanisms

Medium Risk

Vulnerability ID: STEDU-003
Title: Excessive owner privileges in wstEDU rescue function
Risk Level: Medium
Status: Not Fixed
Affected Contract: wstEDU.sol

Description: The `rescueERC20` function allows the owner to withdraw any ERC20 tokens except stEDU from the contract. This creates a risk of abuse where the owner could extract value from tokens accidentally sent to the contract.

```
/// @dev Allows the owner to rescue any ERC-20 tokens (except stEDU) sent by accident.
function rescueERC20(IERC20 token, uint256 amount, address to) external onlyOwner {
    require(token != stEDU, "stEDU locked");
    token.transfer(to, amount);
}
```

Figure 3: Rescue function with excessive privileges

Technical Details:

- Owner can withdraw any ERC20 token except stEDU
- No time locks or approval mechanisms
- Function lacks proper access control beyond owner check
- Could be used to extract value from user mistakes

Remediation Recommendations:

- Implement time locks for rescue operations
- Add community governance for rescue decisions
- Consider removing or restricting the rescue function
- Implement proper notification mechanisms

Medium Risk

Vulnerability ID: STEDU-004
Title: Incorrect totalAssets Accounting Leads to MEV and Price Manipulation
Risk Level: Medium
Status: Not Fixed
Affected Contract: stEDU.sol

Description:
'totalAssets()' simply returns the WEDU balance of the vault. When surplus WEDU is transferred into the contract *before* 'sync()' is called, 'totalAssets' grows while 'index' (share-price) remains unchanged. This breaks the ERC4626 accounting invariant $pricePerShare \approx totalAssets/totalSupply$.

```
    ////////////////////////////////////////////////////////////////*/
    /// @notice Fold any extra WEDU accidentally sent to the vault into `index`,
    ///         the tokens stranded. Callable by anyone.
    function sync() external whenNotPaused nonReentrant {
        uint256 expected = (index * totalSupply()) / 1e18;
        uint256 actual   = wedu.balanceOf(address(this));
        require(actual > expected, "No surplus");
        uint256 surplus  = actual - expected;
        require(totalSupply() > 0, "No shares");

        index += (surplus * 1e18) / totalSupply();
        emit SurplusSynced(surplus, index);
    }
```

Figure 4: Surplus WEDU causes accounting mismatch

Technical Details:

- Any address can donate WEDU to the vault, increasing `WEDU.balanceOf(address(this))`.
- Until someone calls `sync()`, `index` is not updated, so share price is understated.
- External protocols that price shares using $totalAssets/totalSupply$ will overestimate share value.
- MEV bots can front-run integrations or oracle updates, then call `sync()` to capture the surplus.

Impact Assessment:

- Price oracles, lending protocols, or routers may misprice stEDU collateral, leading to bad debt.
- Attackers can create artificial TVL spikes and extract value through timing-based arbitrage.
- Misreporting undermines transparency of assets under management.

Remediation Recommendations:

- Derive `totalAssets` from `index * totalSupply / 1e18` to ensure accounting consistency.
- Auto-call `sync()` in `receive()` when surplus above a threshold, or incentivise callers with a fee reward.
- Emit a `SurplusPending` flag/event and have off-chain integrators wait until `sync()` clears the flag.

3 Summary and Recommendations

3.1 Key Findings Summary

The main issues identified in this audit include:

1. **Centralization Risk**: Owner has excessive control over reward distribution
2. **Gas Limit Issues**: Complex array operations may cause transaction failures
3. **Incorrect Asset Accounting**: totalAssets misreport enables price manipulation
4. **Access Control**: Excessive privileges in rescue functions

3.2 Overall Security Recommendations

1. **Implement decentralized governance**: Reduce centralization risks through multi-signature wallets and time locks
2. **Optimize gas usage**: Implement pagination and iteration limits for complex operations
3. **Enhance access control**: Review and restrict owner privileges where appropriate
4. **Improve input validation**: Add comprehensive validation for all user inputs
5. **Strengthen monitoring**: Implement detailed event logging for security monitoring
6. **Comprehensive testing**: Conduct thorough testing with edge cases and stress tests
7. **Emergency procedures**: Implement proper emergency pause and recovery mechanisms

3.3 Positive Security Features

The audit also identified several positive security features:

- Proper reentrancy protection using ReentrancyGuard
- Time-lock mechanism (7 days) prevents flash loan attacks
- Disabled standard ERC4626 entry points to enforce custom logic
- Pausable functionality for emergency situations
- Proper use of OpenZeppelin security libraries

3.4 Fix Verification

After fixing all identified issues, the following verification is recommended:

- Conduct re-audit of modified code
- Perform comprehensive testing on testnet
- Consider establishing a bug bounty program
- Implement continuous monitoring post-deployment

4 Disclaimer

This audit report is based on analysis of the provided smart contract code and has limited scope. This report cannot guarantee discovery of all possible security vulnerabilities, nor can it guarantee code security in all situations. The project team is advised to conduct multiple rounds of security audits before deployment and establish continuous security monitoring mechanisms.