Smart Contract Audit Report

Sharp AI Staking Contract underwent a comprehensive audit on December 7, 2024

Smart Contract	rewards.sol
Type Of Utility	Rewards
Platform	ETH, Ethereum Virtual Machine
Language	Solidity
Method	Manual and Statics Analysis
Address	0xF2c9e1f8c02ACFfB4Cae04e8B9aaB8B900991607

AVERAGE Security Score

The score is determined by analyzing the lines of code and assigning weights to issues based on their severity and confidence levels. To enhance your score, review the detailed results and apply the recommended remediation strategies.

Vulnerability Summary

- 1 Critical
- 1 High
- 2 Medium
- 2 Low
- 1 Information

Content

Classification and Severity

Critical

This vulnerability could lead to significant consequences, such as the loss or mismanagement of funds, or other severe financial impacts.

High

High-severity vulnerabilities represent a major risk to the Smart Contract and the organization. They could result in user fund losses under certain conditions and are difficult to exploit.

Medium

This issue affects the functionality of the contract but does not cause substantial disruption to its overall operations.

Low

This issue has a minor impact on the contract's functionality and does not significantly affect its operation.

Information

This issue does not interfere with the contract's functionality but addressing it would follow best practices.

Audit Scope

This Audit Report mainly focuses on the overall security of the **Sharp AI** token Rewards Smart Contract. This audit was conducted with rigorous attention to the general implementation of the contract and by examining the overall architectural layout of the software implementation. The reliability and correctness of this smart contract's codebase are being assessed.

The auditing process pays special attention to the following considerations:

- Identifies security related issues within each contract and the system of contract.
- A full assessment of the code quality and general software architecture patterns and best practices used.

Audit Method

Rigorous testing of the project has been performed. Detailed code base analysis was conducted, reviewing the smart contract architecture to ensure it is structured and safe.

A detailed, line by line inspection of the codebase was conducted to find any potential security vulnerabilities such as denial of service attacks, race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

Automated and manual testing was employed that included:

- Analysis of on-chain data security
- Analysis of the code in-depth and detailed, manual review of the code, line-by-line.
- Deployment of the code on an in-house testnet blockchain and running live tests
- Determining failure preparations and if worst-case scenario protocols are in place
- Analysis of any third-party code use and verifying the overall security of this

Findings

This report has been developed to identify issues and vulnerabilities in Sharp AI Rewards Smart Contract. During the audit, we uncovered 7 issues of varying severity levels. We employed Manual Review and Static Analysis alongside thorough manual code reviews to identify the following findings.

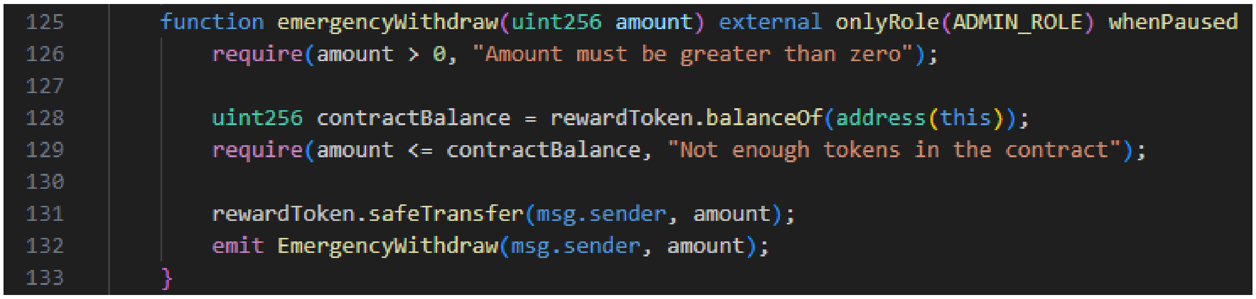
ID	Title	Severity	Status
C001	Incorrect Access Control	Critical	Acknowledged
H001	Reentrancy	High	Acknowledged
M001	Precision Loss During Division	Medium	Acknowledged
L001	Use Of Floating Pragma	Low	Acknowledged
L002	Event Based Reentrancy	Low	Acknowledged
I001	Missing Indexed Keywords	Information	Acknowledged

C001 - Incorrect Access Control (Emergency Withdraw)

Title	**Severity**	**Status**
Incorrect Access Control (Emergency Withdraw)	Critical	Acknowledged

Description

The `emergencyWithdraw` function is restricted to the `ADMIN_ROLE`. However, there are no safeguards to ensure that the `ADMIN_ROLE` is only assigned to trusted accounts. If an unauthorized account gains access to this role, they could withdraw all reward tokens, potentially causing significant financial loss to the project.



```
function emergencyWithdraw(uint256 amount) external onlyRole(ADMIN_ROLE) whenPaused
    require(amount > 0, "Amount must be greater than zero");

    uint256 contractBalance = rewardToken.balanceOf(address(this));
    require(amount <= contractBalance, "Not enough tokens in the contract");

    rewardToken.safeTransfer(msg.sender, amount);
    emit EmergencyWithdraw(msg.sender, amount);
}
```

Recommendation

The assignment of the `ADMIN_ROLE` should be restricted to a multi-signature wallet or highly trusted accounts to minimize the risk of unauthorized access. A time delay mechanism should also be implemented for executing the `emergencyWithdraw` function, allowing time for review and scrutiny before funds are moved. Additionally, all actions related to role assignment, such as `grantRole` and `revokeRole`, should be logged and actively monitored to quickly detect and respond to unauthorized changes.

Alleviation

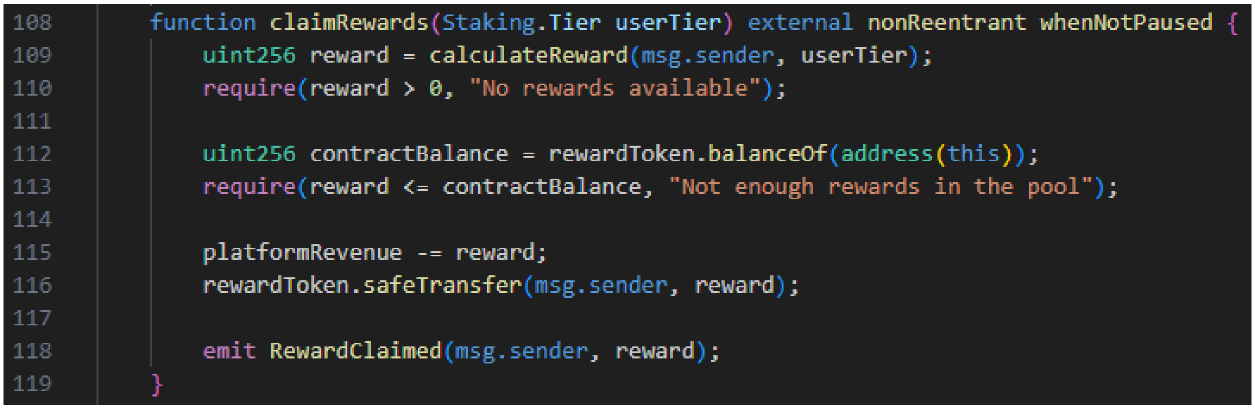
[`Rewards Contract` **]**: Issue acknowledged.

H001 - Reentrancy (Claim Rewards)

Title	Severity	Status
Reentrancy (Claim Rewards)	High	Acknowledged

Description

The `claimRewards` function performs an external call to `safeTransfer` before completing all internal state updates. While the `nonReetrant` modifier is applied, a malicious or poorly designed token contract used as the `rewardToken` could exploit this external call to trigger a reentrant call, potentially causing unexpected behavior or manipulation of the contract's logic.



```
    function claimRewards(Staking.Tier userTier) external nonReentrant whenNotPaused {
        uint256 reward = calculateReward(msg.sender, userTier);
        require(reward > 0, "No rewards available");

        uint256 contractBalance = rewardToken.balanceOf(address(this));
        require(reward <= contractBalance, "Not enough rewards in the pool");

        platformRevenue -= reward;
        rewardToken.safeTransfer(msg.sender, reward);

        emit RewardClaimed(msg.sender, reward);
    }
```

Recommendation

We recommend updating the state after the `safeTransfer` call to ensure critical changes are finalized before external interactions. Use only trusted and well-audited tokens as `rewardToken` to prevent malicious behavior. Additionally, implement a mapping to track claimed rewards for extra protection against reentrancy attacks.

Alleviation

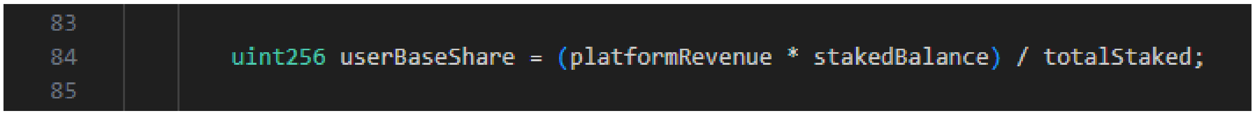
[`Rewards Contract` **]**: Issue acknowledged.

M001.1 - Precision Loss During Division (Share Calculation)

Title	Severity	Status
Precision Loss During Division (Share Calculation)	High	Acknowledged

Description

In the `calculateReward` function, the calculation of `userBaseShare` uses division to determine a user's share of the platform revenue. Due to Solidity's lack of floating-point arithmetic, this division can lead to truncation of fractional values, causing precision loss. Users with small staked balances relative to the total may experience an underestimation of their rewards.



```
    uint256 userBaseShare = (platformRevenue * stakedBalance) / totalStaked;
```

Recommendation

Apply scaling to the numerator in a similar way as with `userBaseShare` . Multiply by a large factor (e.g., `1e18`) before dividing.

Alleviation

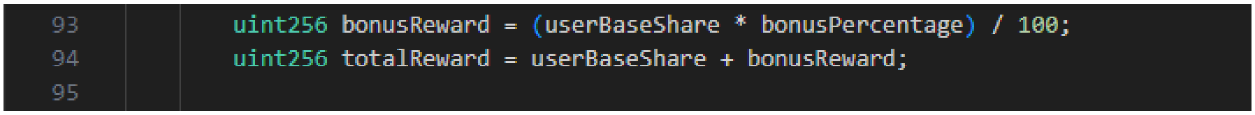
[`Rewards Contract`]: Issue acknowledged.

M001.2 - Precision Loss During Division (Bonus Scaling)

Title	Severity	Status
Precision Loss During Division (Bonus Scaling)	High	Acknowledged

Description

The calculation of `bonusReward` in the `calculateReward` function uses division, which may lead to precision loss, especially for users with smaller base rewards. The lack of floating-point operations in Solidity causes fractional values to be truncated, which can result in an underestimated bonus.



```
    uint256 bonusReward = (userBaseShare * bonusPercentage) / 100;
    uint256 totalReward = userBaseShare + bonusReward;

```

Recommendation

Apply scaling to the numerator in a similar way as with `userBaseShare`. Multiply by a large factor (e.g., `1e18`) before dividing.

Alleviation

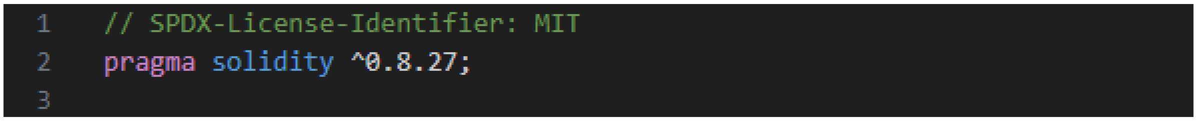
[`Rewards Contract` **]**: Issue acknowledged.

L001 - Use Of Floating Pragma

Title	Severity	Status
Use Of Floating Pragma	Low	Acknowledged

Description

The contract uses a floating pragma version (^0.8.27) in the code. Floating pragmas allow the contract to compile with any newer Solidity version within the specified range, which can lead to unexpected behavior or compatibility issues if future Solidity versions introduce breaking changes.



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.27;

```

Recommendation

Replace the floating pragma with a fixed pragma version to ensure the contract compiles consistently with a specific version of the Solidity compiler.

Alleviation

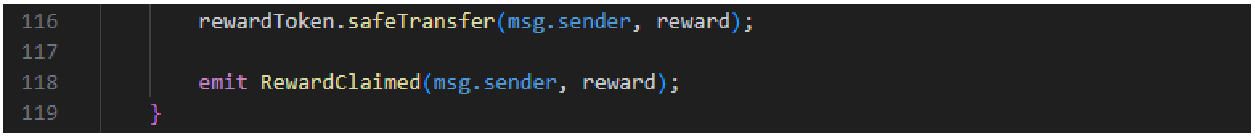
[Rewards Contract **]**: Issue acknowledged.

L002 - Event Based Reentrancy

Title	Severity	Status
Event Based Reentrancy	Low	Acknowledged

Description

The claimRewards function emits the RewardClaimed event after transferring tokens via safeTransfer . While the nonReentrant modifier prevents direct reentrancy, emitting an event after an external call could provide information that might be exploited in certain attack scenarios, such as front-running or reentrant attempts triggered through external systems monitoring these events.



```
        rewardToken.safeTransfer(msg.sender, reward);

        emit RewardClaimed(msg.sender, reward);
    }
```

Recommendation

Reorder the operations to emit the RewardClaimed event before the external call to safeTransfer . This ensures that sensitive state updates and external interactions are properly sequenced to avoid unintended vulnerabilities.

Alleviation

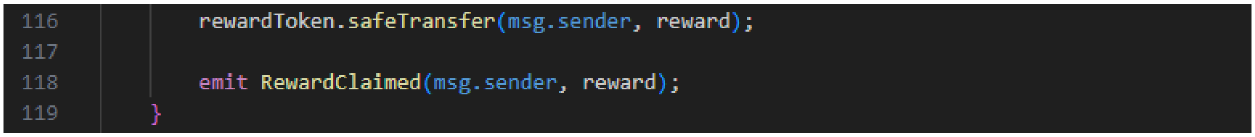
[Rewards Contract]: Issue acknowledged.

I001 - Missing Indexed Keywords

Title	Severity	Status
Missing Indexed Keywords	Information	Acknowledged

Description

The `RewardClaimed` event is defined without using the `indexed` keyword for the `user` parameter. Adding `indexed` allows event logs to be efficiently filtered by the `user` address, making it easier for off-chain systems to monitor and query specific user activities.



```
        rewardToken.safeTransfer(msg.sender, reward);

        emit RewardClaimed(msg.sender, reward);
    }
```

Recommendation

Add the `indexed` keyword to the `user` parameter in the `RewardClaimed` event to enable efficient event filtering.

Alleviation

[`Rewards Contract`]: Issue acknowledged.

Conclusion

The **Rewards** contract is a secure and efficient solution for distributing rewards based on staking tiers and user contributions. Its thoughtful design integrates scalable reward distribution mechanisms and admin-controlled revenue management, making it a safe and reliable choice. While the contract is well-constructed and safe to use, it can achieve even greater security and functionality by focusing on the following improvements:

1. **Admin Role Accountability**: Implement stricter controls for administrative functions, including limiting revenue adjustments and emergency withdrawals to prevent misuse. Multi-signature wallets can add another layer of security.
2. **Reward Calculation**: Enhance calculation logic to eliminate potential division by zero errors and inconsistencies when scaling rewards, ensuring accuracy in all conditions.
3. **Reward Pool Management**: Introduce safeguards to prevent insufficient reward balances during claims, along with automated alerts for proactive fund management.

By addressing these areas, the Rewards contract will not only maintain its safety but also set a higher benchmark for trust and reliability in decentralized reward systems.

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. To get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us based on what it says or doesn’t say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HedgePay and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HedgePay) owe no duty of care towards you or any other person, nor does HedgePay make any warranty or representation to any person on the accuracy or completeness of the report.

The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HedgePay hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HedgePay hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HedgePay, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.