CANTINA

Tea Staking Contracts

Security Review

Cantina Managed review by:

Optimum, Lead Security Researcher
Chinmay Farkya, Associate Security Researcher

February 19, 2025

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Findings **4**
- 3.1 Low Risk . . . 4
  - 3.1.1 Rewards allocated during periods of no stakers will be locked in the contract . . . 4
  - 3.1.2 `REWARDS_DISTRIBUTOR_ROLE` can prevent the update of `rewardsDuration` . . . 4
  - 3.1.3 `REWARDS_DISTRIBUTOR_ROLE` can cause delay and losses in rewards distribution process 4
- 3.2 Gas Optimization . . . 5
  - 3.2.1 Gas optimization in accounting variables . . . 5
- 3.3 Informational . . . 5
  - 3.3.1 Strategies of stakers to maximize rewards . . . 5
  - 3.3.2 Pausing and unpausing is controlled by the same role . . . 5
  - 3.3.3 Recovery of donated ERC20 tokens might not work correctly . . . 5
  - 3.3.4 Documentation errors . . . 6
  - 3.3.5 Deployment scripts miss assigning privileged roles . . . 6
  - 3.3.6 The deadline of a signed message has no upper limit . . . 6
  - 3.3.7 Consider removing `rewardBalance` . . . 6

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Tea is a decentralized protocol for open-source developers to capture the value they create.

From Feb 5th to Feb 7th the Cantina team conducted a review of tea-staking-contracts on commit hash 19184d10. The team identified a total of **11** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	3	0	3
Gas Optimizations	1	1	0
Informational	7	4	3
Total	**11**	**5**	**6**

3 Findings

3.1 Low Risk

3.1.1 Rewards allocated during periods of no stakers will be locked in the contract

Severity: Low Risk

Context: StakingRewards.sol#L286

Description: A known limitation of this type of staking contract is that rewards are allocated per period. If there are no stakers during a given period (i.e., `totalSupply` of stTEA is 0), the pre-allocated rewards for that period become inaccessible rather than being retroactively assigned to the first staker. This issue is particularly relevant between the first call to `notifyRewardAmount` and the first call to `stake`. In other cases, it is more of a natural consequence of staking demand fluctuations.

Recommendation: Consider adding a check in `notifyRewardAmount` to revert the transaction if `totalSupply()` is 0. However, this could impact composability if `notifyRewardAmount` is called by another contract. As an alternative, instead of modifying the Solidity code, ensure that at least a minimal amount (e.g., a few wei of TEA) is staked whenever the rewards program is active to keep all rewards accessible.

Tea: Acknowledged.

Cantina Managed: Acknowledged.

3.1.2 `REWARDS_DISTRIBUTOR_ROLE` can prevent the update of `rewardsDuration`

Severity: Low Risk

Context: StakingRewards.sol#L327-L332

Description: Only the `DEFAULT_ADMIN_ROLE` has permission to modify the staking contract's rewards duration. It has a check that prevents resetting the `rewardsDuration` if a reward period is currently running.

This check can create a problem if the `REWARDS_DISTRIBUTOR_ROLE` notifies some new reward amount to purposefully push the periodFinish value in order to revert calls to `setRewardsDuration()`. This quirk can be misused by the `REWARDS_DISTRIBUTOR_ROLE` to DoS any updates of `rewardsDuration`, even by notifying dust rewards.

Recommendation: Document this behaviour and layout instructions for correctly operating the trusted role.

Tea: Acknowledged.

Cantina Managed: Acknowledged.

3.1.3 `REWARDS_DISTRIBUTOR_ROLE` can cause delay and losses in rewards distribution process

Severity: Low Risk

Context: StakingRewards.sol#L286-L299

Description: The `REWARDS_DISTRIBUTOR_ROLE` is permissioned to notify reward amounts. There can be problems if this trusted entity notifies low amounts or zero amount of rewards:

- If they notify zero amount, this can delay the reward disbursal because this can spread the whole leftover amount (associated with for ex. 1 or 2 days) over the next 7 days.
- The new `rewardRate` could even be set to zero leading to complete loss of leftover rewards if leftover / 7 days rounds down to zero $\Rightarrow$ this will turn legitimate existing rewards into un-emitted dust.

Recommendation: Consider adopting an off-chain process that prevents the `REWARDS_DISTRIBUTOR_ROLE` from emitting low/zero rewards.

Tea: Acknowledged.

Cantina Managed: Acknowledged.

3.2 Gas Optimization

3.2.1 Gas optimization in accounting variables

Severity: Gas Optimization

Context: StakingRewards.sol#L89-L90

Description: Some state variables are being initialised to zero even though the default value of their data types is zero. They are `periodFinish` and `rewardRate`. This costs additional gas but this step is not required.

Recommendation: Consider removing the value assignment to zero, and just declare the variables.

Tea: Addressed in commit 274ef66d.

Cantina Managed: Fix verified.

3.3 Informational

3.3.1 Strategies of stakers to maximize rewards

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The `stakeWithBoost` function requires a signature from the `TEA_RANK_ATTESTER_ROLE`. This signature includes the staker's address, the boost coefficient, a nonce, and a deadline. However, it does not include the amount to be staked. As a result, stakers can strategically time their deposits based on fluctuations in boost coefficients, withdraw and re-stake if the coefficient increases and stop depositing in case the boost coefficient drops. Since we lack visibility into the logic governing boost coefficient adjustments, we cannot fully assess all potential risks associated with this mechanism.

Recommendation: While this is not necessarily a security issue, it is an important design consideration. Future iterations of the signature scheme may benefit from including the staked amount to prevent strategic re-staking and ensure more predictable behavior.

Tea: Acknowledged.

Cantina Managed: Acknowledged.

3.3.2 Pausing and unpausing is controlled by the same role

Severity: Informational

Context: StakingRewards.sol#L334-L338

Description: The staking contract can be paused : the permission to both pause / unpause the contract is controlled by the same role ie. DEFAULT_ADMIN_ROLE. It is generally a best practice to employ a different role with the ability to pause, that cannot unpause.

Recommendation: Employ a "PAUSER" role that is allowed to only pause the contract.

Tea: Separated pauser into its own role in commit 830837be.

Cantina Managed: Fix verified.

3.3.3 Recovery of donated ERC20 tokens might not work correctly

Severity: Informational

Context: StakingRewards.sol#L322-L325

Description: `recoverERC20()` method is used to sweep donated stray tokens from the staking contract. Only `DEFAULT_ADMIN_ROLE` has the required permissions.

But the problem is that those tokens are directly sent to msg.sender, and it is possible that the address having `DEFAULT_ADMIN_ROLE` is not able to handle those tokens (for e.g. a contract lacking methods to transfer/approve those tokens elsewhere). This can make the recover functionality useless.

Recommendation: Document and verify that the address holding `DEFAULT_ADMIN_ROLE` should be able to handle/ use these recovered tokens elsewhere.

Tea: Acknowledged.

Cantina Managed: Acknowledged.

3.3.4 Documentation errors

Severity: Informational

Context: StakingRewards.sol#L318

Description: There is one instance of irrelevant natspec :

- `recoverERC20()`: "*Added to support recovering LP Rewards from other systems such as BAL to be distributed to holders*" ⇒ irrelevant comment.

Recommendation: Consider removing this line.

Tea: Addressed in commit f1c25f1e.

Cantina Managed: Fix verified.

3.3.5 Deployment scripts miss assigning privileged roles

Severity: Informational

Context: StakingRewards.sol#L78-L79

Description: For proper functioning of the system, two privileged roles need to be assigned to some address : namely the `REWARDS_DISTRIBUTOR_ROLE` and `TEA_RANK_ATTESTER_ROLE`. But no address is being assigned these roles currently. Both the constructor and the deployment scripts are missing it.

Recommendation: Even though these roles can be later assigned to any address by the address holding `DEFAULT_ADMIN_ROLE`, consider adding it to deployment script to prevent an error.

Tea: Acknowledged.

Cantina Managed: Acknowledged.

3.3.6 The deadline of a signed message has no upper limit

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description/Recommendation: To mitigate the risk of users timing their calls to `stakeWithBoost` to maximize rewards (as highlighted in finding 2), consider incorporating a signing timestamp into the signed message. This would allow you to enforce a restriction where the difference between the `deadline` and the signing time does not exceed a predefined limit (e.g., one day). If this delta is exceeded, the transaction should revert. Ensure that the timestamp used is measured in seconds and aligns with Solidity's timestamp conventions.

Tea: Addressed in commit 254b035a.

Cantina Managed: Fix verified.

3.3.7 Consider removing `rewardBalance`

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: `rewardBalance` is used inside `notifyRewardAmount` for a check against `rewardRate` as we can see in the following code:

```
// ...
// Account for rewards directly.
// Forced transfers affecting `address(this).balance` considered burned.
uint256 _rewardBalance = rewardBalance + reward;

// Ensure the provided reward amount is not more than the reward balance in the contract.
// This keeps the reward rate in the right range, preventing overflows due to
// very high values of rewardRate in the earned and rewardsPerToken functions;
// Reward + leftover must be less than 2^256 / 10^18 to avoid overflow.
/// @audit can this trigger at all if we're not using balance checks?
if (rewardRate > _rewardBalance / rewardsDuration) revert RewardRateTooHigh();

rewardBalance = _rewardBalance;
// ...
```

However, as discussed with the developer, we could not find a reason to justify this variable and this particular condition that is enforced in the code.

Recommendation: Consider removing all instances of `rewardBalance` and the code around it.

Tea: Addressed in commit d8639608.

Cantina Managed: Fix verified.