CANTINA

Infrared contracts PR 575

Security Review

Cantina Managed review by:

0xRajeev, Lead Security Researcher
Cryptara, Security Researcher

March 25, 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 Gas Optimization . . . 4
  - 3.1.1 Use of post-Increment (`i++`) in loop leads to extra gas used . . . 4
  - 3.1.2 Missing zero balance check in `sweepPayoutToken()` . . . 4
- 3.2 Informational . . . 4
  - 3.2.1 Storage gap duplication in `BribecollectorV1_2.sol` . . . 4
  - 3.2.2 Incorrect Natspec and code comments reduce readability . . . 5
  - 3.2.3 Reusing errors across different kinds of violating conditions reduces readability and UX . . . 5
  - 3.2.4 Missing event emit in `sweepPayoutToken()` . . . 5
  - 3.2.5 `claimFees()` can optimally sweep entire `BribeCollector` balance . . . 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

Infrared simplifies interacting with Proof of Liquidity with liquid staking products such as iBGT and iBERA,

From Mar 12th to Mar 13th the Cantina team conducted a review of infrared-contracts[PR-575] on commit hash f8f8133e. The team identified a total of **7** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	0	0	0
Gas Optimizations	2	1	1
Informational	5	3	2
Total	**7**	**4**	**3**

3 Findings

3.1 Gas Optimization

3.1.1 Use of post-Increment (`i++`) in loop leads to extra gas used

Severity: Gas Optimization

Context: BribeCollectorV1_2.sol#L75

Description: In Solidity, using the post-increment operator (`i++`) instead of the pre-increment operator (`++i`) may slightly increase gas usage. The post-increment operator temporarily stores the initial value before incrementing, resulting in unnecessary additional operations. Although the difference per iteration may seem minimal, this inefficiency accumulates, especially in loops with many iterations, leading to higher overall gas consumption.

Recommendation: Replace all occurrences of post-increment (`i++`) with pre-increment (`++i`) when possible, particularly in loops, to optimize gas efficiency. Adopting this practice consistently improves overall contract performance and reduces deployment and transaction costs.

Infrared: Acknowledged.

Cantina Managed: Acknowledged.

3.1.2 Missing zero balance check in `sweepPayoutToken()`

Severity: Gas Optimization

Context: BribeCollectorV1_2.sol#L96-L101

Description: `sweepPayoutToken()` can be called by anyone to transfer any dust `payoutToken` from `BribeCollector` to `infrared.collectBribes()`. However, this is missing a zero balance check for `payoutToken`.

Recommendation: Consider adding a zero balance check in `sweepPayoutToken()` to avoid having to call `infrared.collectBribes()`, which will save gas and avoid event spam.

Infrared: Fixed in PR 591.

Cantina Managed: Fixed as recommended.

3.2 Informational

3.2.1 Storage gap duplication in `BribecollectorV1_2.sol`

Severity: Informational

Context: BribeCollectorV1_2.sol#L29

Description: In `BribecollectorV1_2.sol`, defining an additional storage gap despite inheriting from `InfraredUpgradeable`, which already includes a storage gap, introduces unnecessary and potentially dangerous spacing in the storage layout. Specifically, the variables `payoutToken` and `payoutAmount` become positioned between the parent (`InfraredUpgradeable`) and child contract gaps. This arrangement increases the risk of storage misalignment when updating or extending either contract. Any modification to the variables or the storage gap in the `InfraredUpgradeable` parent contract could unintentionally shift the storage slot locations of critical state variables in `BribecollectorV1_2.sol`, causing severe data integrity issues.

Moreover, the additional storage gap defined in the child contract does not inherently provide safety or flexibility. Instead, it creates a false sense of security, potentially masking the vulnerability that arises from variable misplacement and storage slot misalignment across the inheritance hierarchy.

Recommendation: Eliminate the redundant storage gap defined in `BribecollectorV1_2.sol`. Rely on the existing gap provided by the parent contract, `InfraredUpgradeable`, for future variable additions or changes. Additionally, keep in mind that modifying variables or the storage gap size in `InfraredUpgradeable` may break storage alignment in child contracts, potentially corrupting state variables. Therefore, ensure that storage layout modifications in parent contracts are thoroughly analyzed for their impact on inherited contracts before implementation.

Infrared: Acknowledged.

Cantina Managed: Acknowledged.

3.2.2 Incorrect Natspec and code comments reduce readability

Severity: Informational

Context: BribeCollectorV1_2.sol#L25, BribeCollectorV1_2.sol#L35, BribeCollectorV1_2.sol#L67, BribeCollectorV1_2.sol#L74, BribeCollectorV1_2.sol#L97

Description: There are some incorrect Natspec and code comments, which reduce readability.

Recommendation: Consider correcting comments as listed below:

1. "/// @notice Payout amount is a constant value that is paid out the caller of the `claimFees` function." → "1. "/// @notice Payout amount is a constant value that is paid by the caller of the `claimFees` function.".
2. " /// @notice Set the payout token for the bribe collector." → " /// @notice Set the payout amount for the bribe collector.".
3. " // increase the allowance of the payout token to the infrared contract to be send to" → " // Set the allowance of the payout token to the infrared contract to be sent to".
4. "// From all the specified fee tokens, transfer them to the recipient." → "// For all the specified fee tokens, transfer them to the recipient.".

Infrared: Fixed in PR 591.

Cantina Managed: Fixed as recommended.

3.2.3 Reusing errors across different kinds of violating conditions reduces readability and UX

Severity: Informational

Context: BribeCollectorV1_2.sol#L59-L61, BribeCollectorV1_2.sol#L78-L80, BribeCollectorV1_2.sol#L82-L84, BribeCollectorV1_2.sol#L87-L89

Description: There are checks for different kinds of errors in `BribeCollector` and some of them reuse errors for different kinds of violating conditions. This reduces code readability and UX for users when they encounter these error messages.

Recommendation: Consider changing the below errors as suggested for the different contexts:

1. `InvalidAmount()` in `senderBalance < payoutAmount` check → `InsufficientBalance()`.
2. `RewardTokenNotSupported()` in `feeToken == payoutToken` check → `InvalidFeeToken()`.
3. `RewardTokenNotSupported()` in `infrared.whitelistedRewardTokens(feeToken)` check → `FeeTokenNotWhitelisted()`.
4. `InvalidAmount()` in `feeAmount > contractBalance` check → `InsufficientFeeTokenBalance()`.

Infrared: Fixed in PR 591.

Cantina Managed: Fixed as recommended.

3.2.4 Missing event emit in `sweepPayoutToken()`

Severity: Informational

Context: BribeCollectorV1_2.sol#L95-L102

Description: `sweepPayoutToken()` does not emit an event itself but its call to `infrared.collectBribes()` emits `BribesCollected` event. However, this will be the same event emitted when `infrared.collectBribes()` is called from either `sweepPayoutToken()` or `collectBribes()`.

Recommendation: Consider emitting a sweep-specific event in `sweepPayoutToken()` for better monitoring.

Infrared: Acknowledged.

Cantina Managed: Acknowledged.

3.2.5 `claimFees()` can optimally sweep entire `BribeCollector` balance

Severity: Informational

Context: BribeCollectorV1_2.sol#L69-L71

Description: `claimFees()` currently transfers only `payoutAmount` of `payoutToken` in its call to `infrared.collectBribes()`. Any excess dust payout tokens in the contract will have to be swept later by calling `sweepPayoutToken()`. This sweeping can optimally be done when `claimFees()` is called by not transferring only the `payoutAmount` transferred in by the caller, but instead transferring the entire `BribeCollector` balance to include any dust tokens from earlier.

Recommendation: Consider approving and transferring `ERC20(payoutToken).balanceOf(address(this)` instead of `payoutAmount` in `claimFees()`.

Infrared: Fixed in PR 591.

Cantina Managed: Fixed as recommended.