CANTINA

Airdrop contracts

Security Review

Cantina Managed review by:

Eric Wang, Lead Security Researcher
RustyRabbit, Security Researcher

March 8, 2025

Contents

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

Corn is a layer 2 network focused on revolutionizing the capital efficiency of Bitcoin as a nascent asset class. Designed to provide scalable infrastructure that leverages Ethereum in a manner that allows for the secure management of billions of dollars in liquidity with low transactional costs secured by the Bitcoin L1.

On Feb 4th the Cantina team conducted a review of PR 45 and `VestingWallet.sol` at commit hash e3305b6. The team identified a total of **5** issues:

Issues Found

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

3 Findings

3.1 Medium Risk

3.1.1 The current beneficiary may be allowed to call the `proposeBeneficiary()` function

Severity: Medium Risk

Context: VestingWallet.sol#L90-L94

Description: The NatSpec of the `proposeBeneficiary()` function specifies that the function is restricted to only the contract owner. In IVestingWallet.sol#L152-L156:

```
/**
* @notice Proposes a new beneficiary. This can only be called by the owner of the contract.
* @param _newBeneficiary The new beneficiary.
*/
function proposeBeneficiary(address _newBeneficiary) external;
```

Technically, the `owner` of the contract is the `beneficiary`. However, in the actual implementation, the function is guarded by `requiresAuth`, so the caller does not necessarily have to be the `owner`.

Moreover, the `owner` (i.e., `beneficiary`) should not have the permission to call the `proposeBeneficiary()` function. Otherwise, it could change the beneficiary to another address at any time without delay, which goes against the purpose of the two-step design of the beneficiary change and potentially bypasses the Governor.

Recommendation: For the code, consider adding a check to the `proposeBeneficiary()` function to ensure the caller is not the current beneficiary explicitly. For the NatSpec, consider changing the comments to "This can only be called by an authorized caller except the current beneficiary." A similar change also applies to IVestingWallet.sol#L32.

Bitcorn: There is indeed a mismatch between the natspec and the implementation of the `proposeBeneficiary` function. Good catch, we've addressed it in PR 46.

Additionally, since the `requiresAuth` modifier would necessitate extra steps and coordination among multiple people from the foundation to mistakenly allow the beneficiary to call the `proposeBeneficiary` function, we consider the risk to be low to nonexistent.

Cantina Managed: Acknowledged.

3.2 Low Risk

3.2.1 Design choices and security considerations for `VestingWallet`

Severity: Low Risk

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

Description: Below are some design choices and security considerations for the vesting wallet:

1. Adding tokens during the vesting period has the same effect as if they were added before the start of the period. As a result, a percentage of the newly added token would be instantly unlocked and accessible to the beneficiary.
2. Adding tokens to the wallet is permissionless. Anyone can add native tokens or any ERC-20 tokens to the vesting wallet. The beneficiary should be cautious of not interacting with potentially malicious ERC-20 tokens in the wallet contract.
3. Anyone can call the `release()` function to push the rewards to the beneficiary. If the beneficiary loses its private key, since it no longer controls the address, allowing anyone to push the rewards may lead to a loss of funds. Restricting access control to the `release()` function to only the beneficiary can mitigate this issue. However, it requires the beneficiary to be able to call the `release()` function instead of passively receiving the funds.
4. In the case where the beneficiary is compromised, to prevent the attacker from receiving funds from the vesting wallet (either the currently or future claimable funds), a privileged role would have to upgrade the contract to rescue the funds.

Recommendation: Consider clarifying the vesting wallet's design in public documentation so users can be aware of it. If necessary, consider implementing security measures (either in the contract or off-chain) to reduce the risks outlined above.

Bitcorn: Thank you for the suggestions.

1. Adding tokens during the vesting period has the same effect as if they were added before the start of the period.

In our case the amount and the vesting schedule are fixed and agreed upon by the beneficiary and the team, we do not plan to add any additional tokens to the vesting wallet.

2. Adding tokens to the wallet is permissionless.

That's a good point. We'll consider having a quick FAQ document to address this and other usage questions.

3. Anyone can call the release() function to push the rewards to the beneficiary.

Supporting beneficiaries that may not be able to call the release function was a part of the requirements.

4. ...a privileged role would have to upgrade the contract to rescue the funds.

Due to legal risks, we were unable to implement the clawback or transfer functionality, and may not be allowed to upgrade in case of a hack or loss of private keys. Still we erred on the side of being able to upgrade just in case.

Cantina Managed: Acknowledged.

3.2.2 2 step process for 3 party ownership change can lead to locked funds

Severity: Low Risk

Context: VestingWallet.sol#L97

Description: In a normal 2-step ownership change both previous and future owner are required to send a transaction to facilitate the ownership transfer. This ensures that:

1. The old owner agrees with the transfer.
2. The new owner is able and willing to accept the responsibilities.

In the case of the `VestingWallet` a 3rd party is involved (the CornGovernor) which can initiate the transfer by sending a `proposeBeneficiary` transaction. The old beneficiary is then required to call `acceptBeneficiary` to finalize the ownership transfer.

At no point however is the new owner involved in this process. Therefor if the new owner is unable to act when the ownership (or the beneficiary as they are effectively the same) needs to be changed, the process cannot be completed.

The reason for not involving the new beneficiary is that it should be possible for the beneficiary to be an address which may not be able to interact with the `VestingWallet` as part of custodial platforms.

Recommendation: Consider splitting ownership and beneficiary where the ownership can be transferred with involvement of the new owner, and have a separate process to designate the new beneficiary. Both of which can be subject to CornGovernor approval if needed.

Bitcorn: The current design stems from the fact that we want to support exchanges/custodian addresses as beneficiaries, that may not be able to accept the ownership of the contract.

Cantina Managed: Acknowledged.

3.3 Informational

3.3.1 Minor code and specification improvements

Severity: Informational

Context: VestingWallet.sol#L32, VestingWallet.sol#L89-L94, VestingWallet.sol#L116

Description:

1. VestingWallet.sol#L32: Inheriting from `ContextUpgradeable` is unnecessary and can be removed since `OwnableUpgradeable` already inherits it.
2. VestingWallet.sol#L89-L94: For consistency, implement a zero address check on `_newBeneficiary`.
3. VestingWallet.sol#L116: For better clarity, change the event `EtherReleased()` to `NativeCurrencyReleased()`. It also improves code consistency as the term "native currency" is used in the NatSpec most of the time.

Recommendation: Consider implementing the above suggestions.

Bitcorn: Fixed in PR 46.

Cantina Managed: Verified.

3.3.2 `beneficiary` and `owner` are redundant

Severity: Informational

Context: VestingWallet.sol#L104-L106

Description: `beneficiary` is the storage variable that is used to send the funds to. `owner` is inherited from `OwnableUpgradeable` but has been limited in use by overriding `transferOwnership` and `renounceOwnership`.

`acceptBeneficiary` is the only place where the `onlyOwner` modifier is used in order to ensure only the previous owner (ie. previous beneficiary) can finalize the change to the new beneficiary.

As such `owner` and `beneficiary` are redundant and one of them can be eliminated in favor of the other.

Recommendation: Consider replacing the `beneficiary` variable by `owner`or vice versa.

Note that this is depending on whether you choose to follow the advice of finding 2 step process for 3 party ownership change can lead to locked funds.

Bitcorn: We considered to eliminate the owner variable and, to be completely transparent, ended with the the current design mainly due to time constraints. We'll stand by this decision to avoid last-minute refactoring.

Cantina Managed: Acknowledged.