Puffer Finance: Carrot Vesting Upgrade

Security Review

Cantina Managed review by:

Gerard Persoon, Lead Security Researcher
Ladboy233, Security Researcher

October 31, 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 level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

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 are 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

Puffer is a decentralized native liquid restaking protocol (nLRP) built on Eigenlayer. It makes native restaking on Eigenlayer more accessible, allowing anyone to run an Ethereum Proof of Stake (PoS) validator while supercharging their rewards.

From Oct 29th to Oct 30th the Cantina team conducted a review of puffer-contracts on commit hash 728755fe. 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	1	0	1
Low Risk	4	0	4
Gas Optimizations	1	0	1
Informational	5	0	5
Total	**11**	**0**	**11**

3 Findings

3.1 Medium Risk

3.1.1 `recoverPuffer()` prevents pending tokens to be retrieved

Severity: Medium Risk

Context: CarrotVesting.sol#L166-L168, CarrotVesting.sol#L194-L202

Description: If someone does a `startVesting()` just before `recoverPuffer()` then he has send his `Carrot` tokens, but will never receive `Puffer` tokens, because all `Puffer` tokens are removed and because `$.isDismantled` is set.

Recommendation: Consider making this a two step process:

1. Disable `startVesting()`.
2. After a reasonable time, at least `newDuration`, call `recoverPuffer()`.

Note: using `pause()` doesn't solve this, because that also pauses `claim()`.

Puffer Finance: Acknowledged.

Cantina Managed: Acknowledged.

3.2 Low Risk

3.2.1 Missing init function

Severity: Low Risk

Context: CarrotVesting.sol#L92-L97

Description: One init function isn't called. Although its currently empty is safer to call it in case this is changed in the future.

Recommendation: Consider also calling `__Context_init()`.

Puffer Finance: Acknowledged.

Cantina Managed: Acknowledged.

3.2.2 `calculateClaimableAmount()` doesn't check `$.isDismantled`

Severity: Low Risk

Context: CarrotVesting.sol#L313-L315

Description: Function `calculateClaimableAmount()` doesn't check `$.isDismantled`. This means that `calculateClaimableAmount()` indicates a user can claim tokens, while `claim()` will revert.

Recommendation: Consider adding a check `$.isDismantled` for in `calculateClaimableAmount()`.

Puffer Finance: Acknowledged.

Cantina Managed: Acknowledged.

3.2.3 Rounding issues in calculation

Severity: Low Risk

Context: CarrotVesting.sol#L130, CarrotVesting.sol#L322, CarrotVesting.sol#L346-L350

Description: In extreme cases, due to rounding issues, too much `Puffer` can be calculated. See the POC below.

Proof of Concept:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.30;
import "hardhat/console.sol";

contract testcalc {
   constructor() {
      uint endOfVesting = block.timestamp;
      uint duration = 11;
      uint steps = 7;
      uint depositedTimestamp = block.timestamp - duration;
      uint claimedAmount = 0;
      uint depositedAmount = 100_000_000 ether;
      uint256 MAX_CARROT_AMOUNT = 100_000_000 ether;
      uint256 TOTAL_PUFFER_REWARDS = 55_000_000 ether;
      uint256 EXCHANGE_RATE = 1e18 * TOTAL_PUFFER_REWARDS / MAX_CARROT_AMOUNT;
      uint256 claimingTimestamp = endOfVesting > block.timestamp ? block.timestamp : endOfVesting;
      uint256 numStepsClaimable = (claimingTimestamp - depositedTimestamp) / (duration / steps);
      uint256 depositedAmountClaimable = (depositedAmount * numStepsClaimable) / steps;
      uint256 claimableAmount = (depositedAmountClaimable * EXCHANGE_RATE / 1e18);
      uint r = uint128(claimableAmount) - claimedAmount;
      console.log(r /1e18); // should be <= TOTAL_PUFFER_REWARDS
      console.log(TOTAL_PUFFER_REWARDS / 1e18);
      console.log(numStepsClaimable); // should be <= steps
   }
}
```

The output is:

```
86428571 ==> this is more than available
55000000
      11 ==> this is more than the number of steps
```

Recommendation: Consider to enforce `(duration / steps) * steps == duration` in `reinitializeVesting()`.

Puffer Finance: Acknowledged.

Cantina Managed: Acknowledged.

3.2.4 Unsafe typecasts

Severity: Low Risk

Context: CarrotVesting.sol#L350, CarrotVesting.sol#L360, CarrotVesting.sol#L366

Description: Several unsafe typecasts are done. If the input value is to large, it will be truncated without error. Although in the current code it doesn't cause problems, it is still safer to use `safeCast`

Recommendation: Consider using `SafeCast.toUint128()`.

Puffer Finance: Acknowledged.

Cantina Managed: Acknowledged.

3.3 Gas Optimization

3.3.1 Caching of variables

Severity: Gas Optimization

Context: CarrotVesting.sol#L166-L178, CarrotVesting.sol#L313-L316

Description: Some variables could be cached to save some gas.

Recommendation: Consider caching the following variables. Do check this indeed saves gas:

- `$.vestings[msg.sender].length`.
- `$.vestings[msg.sender][i]`.
- `$.vestings[user].length`.

Puffer Finance: Acknowledged.

Cantina Managed: Acknowledged.

3.4 Informational

3.4.1 Two ways for OpenZeppelin paths

Severity: Informational

Context: remappings.txt#L4-L5, CarrotVesting.sol#L7-L9

Description: The path for the OpenZeppelin directory is written in two ways. `remappings.txt` make sure they both point to the same destination so it does work. However it is confusing.

Recommendation: Consider changing the code to:

```
- import { ... } from "@openzeppelin-contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
+ import { ... } from "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
```

Puffer Finance: Acknowledged.

Cantina Managed: Acknowledged.

3.4.2 Unused error messages

Severity: Informational

Context: CarrotVesting.sol#L28, CarrotVesting.sol#L33

Description: Some error message are not used.

Recommendation: Consider removing them:

```
- error AlreadyInitialized();
- error AlreadyDeposited();
```

Puffer Finance: Acknowledged.

Cantina Managed: Acknowledged.

3.4.3 `recoverPuffer()` can be called multiple times

Severity: Informational

Context: CarrotVesting.sol#L194-L202

Description: The function `recoverPuffer()` can be called multiple times. This doesn't hurt, but it does result in multiple emits.

Recommendation: If you want to prevent this, you can first check `$.isDismantled == false`.

Puffer Finance: Acknowledged.

Cantina Managed: Acknowledged.

3.4.4 Too many vestings can cause out of gas errors

Severity: Informational

Context: CarrotVesting.sol#L166-L178, CarrotVesting.sol#L353-L358

Description: If too many vestings are added for a user, then an out of gas error can occur trying too `claim()`. This is limited to the used, so he only shoots himself in the foot.

Recommendation: If you want to prevent this, you can check and limit the size of the `$.vestings[msg.sender]`.

Puffer Finance: Acknowledged.

Cantina Managed: Acknowledged.

3.4.5 Old version of OpenZeppelin libraries

Severity: Informational

Context: package.json#L15-L16

Description: An old version of the OpenZeppelin libraries is used. Newer versions might have fixed bugs.

Recommendation: Consider updating the OpenZeppelin libraries.

Puffer Finance: Acknowledged.

Cantina Managed: Acknowledged.