Beramonium
Audit Report

January 8, 2025

Conducted by:

Bogo Cvetkov (b0g0), Independent Security Researcher

Table of Contents

1. About b0g0

Bogo Cvetkov (b0g0) is a smart contract security researcher with a proven track record of consistently uncovering vulnerabilities in a wide spectrum of DeFi protocols. Constantly pushing the limits of his expertise, he strives to be a superior security partner to any protocol & client he dedicates himself to!

2. About Beramonium

Beramonium is the first & biggest idle RPG on Berachain

3. Risk Classification

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1. Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users
- **Medium** - leads to a moderate loss of assets in the protocol or some disruption of the protocol's functionality
- **Low** - funds are not at risk

3.2. Likelihood

- **High** - almost certain to happen, easy to perform, or highly incentivized
- **Medium** - only conditionally possible, but still relatively likely

- **Low** - requires specific state or little-to-no incentive

3.3. Handling severity levels

- **Critical** - Must fix as soon as possible (if already deployed)
- **High** - Must fix (before deployment if not already deployed)
- **Medium** - Should fix
- **Low** - Could fix
- **Governance** - Could fix

4. Executive Summary

For the duration of 4 days **b0g0** has invested his expertise as a security researcher to analyze the smart contracts of **Beramonium** protocol and assess the state of its security. For that time a total of 6 issues have been detected, out of which **1** have been assigned a severity level of **High** and **0** a severity level of **Medium**.

5. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This effort is limited by time, resources, and expertise. My evaluation of the codebase aims to uncover as many vulnerabilities as possible, given the above limitations! Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended!

B0g0 assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Overview

Project	Beramonium
URL	https://gemhunters.beramonium.io
Platform	Berachain
Language	Solidity
Repo	https://github.com/Xanewok/bcg-vesting-audit
Commit Hash	025b736277c18adbfbccf490a272d2a0be367e93
Mitigation	51afa0d382d6e4206919bd66f9eed266cd9d81e5
Dates	3 January - 7 January 2025

Scope

Contract	Address
BcgVesting.sol	-

Issue Statistic

Severity	Count
High	1
Medium	0
Low/Informational	5
Total	**6**

6. Findings

6.1. High Severity

6.1.1. Pending reward calculation is broken due to updating of the wrong timestamp

Context: BcgVesting.sol

Description :

The BCG vesting contract is used to stake BCG tokens and earn rewards in the form of **beramoTokens.** For each day a BSG token is staked it gets unlocked part of the rewards it is entitled to. The total number of days a BSG token can earn rewards is 364 and the amount of rewards that are unlocked each day are determined by the multiplier of the token, where some tokens that are unique earn x10 more rewards than the standard ones.

Staking begins once the **onTokenStaked()** function is called. This sets the **startTimestamp** and **lastCollectionTimestamp** fields in the vesting record for the token:

```
vestingState[tokenId].vesting = VestingPeriod({
            owner: staker,
            startTimestamp: _now,
            // Invariant: lastCollectionTimestamp >= startTimestamp
            lastCollectionTimestamp: _now
        });
```

Both fields are important since they are used to track the days passed since staking started. And based on the days passed, the rewards to collect are calculated. Additionally there is the **daysCollected** variable that tracks how many reward days have been accumulated for each token, so that once the max is reached, the token will not claim any more rewards than it should.

Unlocked rewards are collected through the **collectPendingRewards()** function:

```
function collectPendingRewards(uint16 tokenId) public validateBcgTokenId(tokenId) {
          ...
       (uint256 fullDays, uint256 linearRewards) = _pendingLinearRewards(tokenId);
        if (linearRewards > 0) {
            vestingState[tokenId].daysCollected += uint16(fullDays);

            vestingState[tokenId].vesting.lastCollectionTimestamp = uint48(
                data.vesting.startTimestamp + (fullDays * 1 days) // ←---- @audit
            );

            // Safety: The token address is immutable and picked by the creator,
            // so no risk of reentrancy.
            _beramoToken.safeTransfer(data.vesting.owner, linearRewards);
        }
}
```

The function calls internally **_pendingLinearRewards(),** which is responsible for calculating how many days have passed since the last reward claim:

```
function _pendingLinearRewards(
        uint16 tokenId
    ) internal view returns (uint256 fullDays, uint256 rewards) {
...
       fullDays = _fullDaysElapsed(data.vesting.lastCollectionTimestamp, _now);
        if (fullDays > 0) {
            // Ensure we never go beyond the vesting period (364 days)
            fullDays = (data.daysCollected + fullDays) > VESTING_PERIOD_IN_DAYS
                ? VESTING_PERIOD_IN_DAYS - data.daysCollected
                : fullDays;

         rewards = fullDays * BASE_BERA_DAILY_UNLOCK * _allocationMultiplier(tokenId);

            return (fullDays, rewards);
        }
...
}
```

The whole issue lies in how the **lastCollectionTimestamp** gets updated inside **collectPendingRewards():**

```
 vestingState[tokenId].vesting.lastCollectionTimestamp = uint48(
        data.vesting.startTimestamp + (fullDays * 1 days) // @audit !!! - this should
be lastCollectionTimestamp, not startTimestamp
  );
```

It always uses **startTimestamp** as a base to which to add the **fullDays** that have passed since the last claim, which makes the calculated days invalid and causes the staker to receive less rewards that they should. Here is an example:

- For the sake of simplicity let's assume **startTimestamp** is 0 and each day passed increments it by **1**
- **Alice** stakes **Token1** which sets the state like this - **startTimestamp = 0 / lastCollectionTimestamp = 0 / daysCollected = 0**
- 10 **(now(10) - lastCollectionTimestamp(0))** days pass and **Alice** calls **collectPendingRewards(),** which updates the state to **startTimestamp = 0 / lastCollectionTimestamp = 0+10 =10 / daysCollected = 0+10 = 10**
- 10 **(now(20) - lastCollectionTimestamp(10))** more days pass and **Alice** calls **collectPendingRewards(),** which updates the state to **startTimestamp = 0 / lastCollectionTimestamp = 0+10 = 10 / daysCollected = 10+10 = 20.** As you can see **lastCollectionTimestamp** was updated again to **10,** instead of **20**
- 10 more days pass, but the protocol calculates **20 (now(30) - lastCollectionTimestamp(10)), Alice** calls **collectPendingRewards(),** which updates the state to **startTimestamp = 0 / lastCollectionTimestamp = 0+20 = 20 / daysCollected = 20+20 = 40.** As you can see **daysCollected** has been increased by 20, which strips the staker from 10 days of staking rewards

Recommendation

Refactor the accumulation logic like this:

```
vestingState[tokenId].vesting.lastCollectionTimestamp = uint48(
      data.vesting.lastCollectionTimestamp + (fullDays * 1 days)
);
```

Days should be accumulated on top of **lastCollectionTimestamp** not **startTimestamp**

Resolution:

Fixed

6.2. Governance

6.2.1. Governance Privileges

Context: BcgVesting.sol

Description:

The contract **DEFAULT_ADMIN_ROLE** account has control over several variables that can impact the outcome of a transaction:

- set the staker role and revoke roles
- Initialize the vesting pool

Recommendation:

Consider incorporating a Gnosis multi-signature contract as the **DEFAULT_ADMIN_ROLE** and ensuring that the Gnosis participants are trusted entities

Resolution:

Acknowledged

6.3. Informational

6.3.1. Insufficient validation

Context: BcgVesting.sol

Description:

All issues related to validation are collected here to keep the report focused and easy to read:

- Inside the constructor() check that **beramoToken** & **stakeController** are not **address(0),** especially **beramoToken** which is immutable.
- Inside **pendingRewards()** consider attaching the **validateBcgTokenId()** modifier or in case you don't want the function to revert you can check at the top if **tokeId < BCG_TOKEN_COUNT** and just return 0 to skip unnecessary storage reads
- Inside **onTokenStaked()** check that **vestingPoolInitialized** is enabled before allowing any stakes. Also validate that the provided **staker** parameter is not **address(0)** since this might DOS withdraws or cause them to be sent into the void
- Inside **onTokenUnstaked()** check that the tokens have not been unstaked already. There is no use to allowing the function to be called multiple times. Execute a check on **owner, startTimestamp & lastCollectionTimestamp** and if they are 0 already, revert.

Recommendation:

Consider implementing the above mentioned recommendations

Resolution:

Fixed

6.3.2. Gas optimizations

Context: BcgVesting.sol

Description:

All issues related to gas are collected here to keep the report focused and easy to read:

- Inside **collectPendingRewards()** you can check at the top if **startTimestamp == 0** and return early. The check is also done later in the **_pendingLinearRewards()** call, but doing it early saves on some storage reads and operations.
- Inside **collectPendingRewardsBatch()** you can skip the token validity check, since **collectPendingRewards()** already has the **validateBcgTokenId** modifier. This will prevent double checks on each element

Recommendation:

Consider implementing the above mentioned recommendations

Resolution:

Fixed

6.3.3. Emit events on important state changes

Context: BsgVesting.sol

Description:

The following state changing functions do not emit events:

- initializeVestingPool()
- onTokenStaked()
- onTokenUnstaked()
- collectPendingRewards()

It is considered a best practice to emit events that mark changes in the state of the smart contract. It provides transparency to anyone observing how the contract state has changed and also allows querying by offchain listeners when that is necessary.

Recommendation:

Consider emitting relevant events in the above functions

Resolution:

Fixed

6.3.4. Typography

Context: BcgVesting.sol

Description:

All issues related to typography are collected here to keep the report focused and easy to read:

- The **INITIAL_UNLOCK_TOTAL** could also be simplified to **INITIAL_UNLOCK_TOTAL =LINEAR_UNLOCK_TOTAL,** since the formula does exactly that
- Convention is that interface names should start with I - consider renaming **BcgTokenStakeListener** to **IBcgTokenStakeListener**
- Inside **collectPendingRewards()** add some informative message to the **require()** statement that checks if the caller is the owner or the staker role, currently there is none.

Recommendation:

Consider implementing the above mentioned recommendations

Resolution:

Partially fixed