Beramonium Audit Report

February 7, 2025

Conducted by:

Bogo Cvetkov (b0g0), Independent Security Researcher

Table of Contents

1. About b0g0 3

2. About Beramonium 3

3. Risk Classification 3

3.1. Impact 3

3.2. Likelihood 3

3.3. Handling severity levels 4

4. Executive Summary 4

5. Disclaimer 4

6. Findings 7

6.1. High Severity 7

6.1.1. A malicious staker can unstake a token without removing it from the staked tokens list 7

6.2. Governance 10

6.2.1. Governance Privileges 10

6.3. Informational 10

6.3.1. Insufficient validation 10

6.3.2. Gas optimizations 11

6.3.3. Emit events on important state changes 12

6.3.4. Typography 12

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 3 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	323acc0b515c37b5573f5c2b887a4881ca98a6a0
Mitigation	15bc2279c636f85a92257d26569057bfa0ec9c88
Dates	5 February - 7 February 2025

Scope

Contract	Address
BeramoniumGemhuntersListeners.sol	-

Issue Statistic

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

6. Findings

6.1. High Severity

6.1.1. A malicious staker can unstake a token without removing it from the staked tokens list

Context: BeramoniumGemhuntersListeners.sol

Description :

The **BeramoniumGemhuntersListeners** contract allows NFT holders from the beramonium collection to stake them and earn rewards. Upon calling of **stake()** for each staker an internal list(array) called **_flexStakedList** is updated to keep track of the tokens that have been deposited for each caller:

```
function stake(uint16[] calldata tokenIds) public {
...
 unchecked {
  uint16 stakeCount = stakedBeraCount(msg.sender);
....
_ flexStakedList.setAt(msg.sender, 0, stakeCount + uint16(tokenIds.length));
...
  for (uint16 i = 0; i < tokenIds.length; i++) {
     tokenId = tokenIds[i];
    _flexStakedList.setAt(msg.sender, i + stakeCount + 1, tokenId);
     _beramonium.safeTransferFrom(msg.sender, address(this), tokenId);
...
   }
 }

}
```

Upon unstaking through **unstakeByIndices(),** the **_flexStakedList** gets updated again to exclude the tokens that are withdrawn by the staker:

```
function unstakeByIndices(uint16[] calldata indices) public {\
...
  for (i = 0; i < indices.length; i++) {
   uint16 removedId = _flexStakedList.getAt(msg.sender, indices[i] + 1);
   uint16 last = _flexStakedList.getAt(msg.sender, stakeCount);
   _flexStakedList.setAt(msg.sender, indices[i] + 1, last);

    stakeCount--;

@>    _beramonium.safeTransferFrom(address(this), msg.sender, removedId);
    .....
   }
   // Commit the final stake count
   _flexStakedList.setAt(msg.sender, 0, stakeCount);

}
```

However there is a re-entrancy vulnerability in the flow when tokens get sent out to the staker before the storage for stake count inside **_flexStakedList** is finally updated, which allows a staker to exploit the unstaking flow by withdrawing a tokenId, without removing it from the array.

Here is a breakdown of the issue

1. Bob stakes 5 NFTs and his **_flexStakedList** looks like this - **[5,id1,id2,id3,id4,id5]** (index 0 stores the number of NFTs stored after it)
2. Bob calls **unstakeByIndices()** with **indices=[0]** to unstake NFT with **id1**

3. Inside the loop, the state of the storage variable **_flexStakedList** right before the **safeTransferFrom()** call looks like this **[5,id5,id2,id3,id4,id5]**
4. **safeTransferFrom()** calls the **onERC721Received** callback of the caller, where he can execute any logic he like
5. Inside Bob contracts **onERC721Received** callback after receiving **id1** it calls **unstakeByIndices()** again, with **indices=[4]** to get **id5.**
6. Since this is a re-entrant call and not all storage is updated this is what happens:
    a. **_flexStakedList[0]**(id count) is not yet updated, which allows bob to provide the indice for the last **id5** which was however copied **in storage** into the index of **id1** during the initial call - **[5,id5,id2,id3,id4,id5]**
    b. **id5** is sent out to **Bob contract**
    c. The re-entrancy call finishes and after that the initial unstake call also concludes leaving the storage for the **_flexStakedList** array in the following state - **[4,id5,id2,id3,id4]**
    d. As you can see Bob unstaked **id1 & id5,** but he still has **id5** in his array
    e. Imagine Bob sells or trades the NFT to Alice and she decides to stake it
    f. Since Bob still has it in his staking list he can steal Alice NFT

Recommendation:

The main issue here is that unsafe re-entrancy is allowed through **safeTransferFrom()**.

The first thing is to add OpenZepplein **nonReentrant** modifiers to both stacking and unstacking functions, which would prevent in-flight calls like the one above.

Also it is always best to follow the so-called CEI pattern, which means that any external calls must be made only after all storage has been updated. Here the **_flexStakedList** length is updated in storage **after** all the external transfer calls, which violates the principle.

Resolution:

Fixed

6.2. Governance

6.2.1. Governance Privileges

Context: BeramoniumGemhuntersListeners.sol

Description:

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

- add/remove listeners
- upgrading the contract

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: BeramoniumGemhuntersListeners.sol

Description:

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

- Inside **initialize()** check that **_beramonium** is not **address(0)**
- Inside **pushListener()** check that **listener** is not **address(0)**

Recommendation:

Consider implementing the above mentioned recommendations

Resolution:

Fixed

6.3.2. Gas optimizations

Context: BeramoniumGemhuntersListeners.sol

Description:

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

- Consider adding a parameters for defining a start and end index when iterating through the staked tokens in **stakedBeras(),** this provides greater flexibility when the list is big
- Consider if it makes sense to add a public function which allows to query the current list of **listeners** which is private variable and cannot be checked, even by the admins

Recommendation:

Consider implementing the above mentioned recommendations

Resolution:

Partially resolved

6.3.3. Emit events on important state changes

Context: BeramoniumGemhuntersListeners.sol

Description:

The following state changing functions do not emit events:

- pushListener()
- popListener()

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: BeramoniumGemhuntersListeners.sol

Description:

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

- The **_beramonium** does not need an underscore since it is a public variable
- Consider adding an underscore to the **listeners** state variable, since it is private

Recommendation:

Consider implementing the above mentioned recommendations

Resolution:

Fixed