Beam Nodes Security Review

Pashov Audit Group

Conducted by: peanuts, shaflow, johnson37

January 28th 2025 - January 30th 2025

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **avierauy/beam-nodes** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Beam Nodes

Beam Nodes is an NFT minting contract that operates directly on the BEAM chain, while users on other chains can reserve NFTs through deposits, which are later batch-minted by an external backend. It supports various payment methods and includes referral rewards and discount incentives.

5. Risk Classification

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

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for 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

6. Security Assessment Summary

review commit hash - df88d614b119cee8a1908c958b6027746c49c453

fixes review commit hash - cdff871eba00a53c66b6d4fc4670aa0a09915cda

Scope

The following smart contracts were in scope of the audit:

- `BeamNodes`

7. Executive Summary

Over the course of the security review, peanuts, shaflow, johnson37 engaged with Beam Nodes to review Beam Nodes. In this period of time a total of **5** issues were uncovered.

Protocol Summary

Protocol Name	Beam Nodes
Repository	https://github.com/avierauy/beam-nodes
Date	January 28th 2025 - January 30th 2025
Protocol Type	Crosschain NFT minting

Findings Count

Severity	**Amount**
Medium	1
Low	4
Total Findings	**5**

Summary of Findings

ID	Title	Severity	Status
[M-01]	Missing a function to set royalties for specific tokenId	Medium	Resolved
[L-01]	Missing chainId check in batchMint() and claim()	Low	Resolved
[L-02]	Consider overriding revokeRole and renounceRole	Low	Resolved
[L-03]	Fixed prices can be arbitraged by large price movement	Low	Acknowledged
[L-04]	Referral system can be attacked via the sybil attack	Low	Acknowledged

8. Findings

8.1. Medium Findings

[M-01] Missing a function to set royalties for specific tokenId

Severity

Impact: Medium

Likelihood: Medium

Description

The `resetDefaultRoyalty` function is used to clear the royalty information set for specific tokenId.

```
function resetDefaultRoyalty(uint256 _tokenId) external onlyRole
      (ADMIN_ROLE) {
        _resetTokenRoyalty(_tokenId);
        emit ResetDefaultRoyalty(_tokenId);
    }
```

However, there is a lack of a function to set royalty information for specific tokenId.

Recommendations

Add a function to set royalty information for specific tokenId.

```
+   function setRoyaltyForId
+ (uint256 _tokenId, address _receiver, uint96 _feeNumerator) external onlyRole(ADMIN_
+       _setTokenRoyalty(_tokenId, _receiver, _feeNumerator);
+       emit ResetDefaultRoyalty(_tokenId);
+   }
```

8.2. Low Findings

[L-01] Missing chainId check in `batchMint()` and `claim()`

The `batchMint` and `claim` functions are intended to be called only on the BEAM chain, but the functions do not perform a chainId check.

```
function batchMint(
      address[]memory_recipients,
      uint256[]memory_quantities
    ) external onlyRole(MINTER_ROLE
        require(_recipients.length == _quantities.length, InvalidValues());
        for (uint256 i = 0; i < _recipients.length; i++) {
            _safeMint(_recipients[i], _quantities[i]);
            emit BatchMint(_recipients[i], _quantities[i]);
        }
    }

    function claim(
      uint256_quantity,
      bytes32[]calldata_proof
    ) external whenNotPaused nonReentrant {
        if (hasClaimed[msg.sender]) revert AlreadyClaimed();

        bool isValidLeaf = _verifyProof(msg.sender, _quantity, _proof);

        if (!isValidLeaf) revert NotInMerkle();

        _safeMint(msg.sender, _quantity);

        hasClaimed[msg.sender] = true;

        emit Claim(msg.sender, _quantity);
    }
```

It is recommended to fetch `block.chainid` and compare it with `BEAM_CHAIN_ID`.

[L-02] Consider overriding revokeRole and renounceRole

AccessControl.sol has two functions, `revokeRole()` and `renounceRole()`, that removes the roles of the caller.

```
function revokeRole(bytes32 role, address account) public virtual onlyRole
    (getRoleAdmin(role)) {
        _revokeRole(role, account);
    }

    function renounceRole
      (bytes32 role, address callerConfirmation) public virtual {
        if (callerConfirmation != _msgSender()) {
            revert AccessControlBadConfirmation();
        }

        _revokeRole(role, callerConfirmation);
    }
```

Since the protocol is quite heavily centralized, consider overriding these functions in BeamNodes.sol and reverting them to prevent any accidental revoking of roles.

[L-03] Fixed prices can be arbitraged by large price movement

In BeamNode, users can choose to mint NFT via depositing native token, USDC, or BEAM token. The admin can set different base price for different deposit tokens. This will make sure that we have a similar deposit price with the different deposit tokens.

Although the admin can update these base prices according to the real-time token price, it's impossible to sync the token price all the time. If the native token or the BEAM token price changes a lot, users may mint NFT tokens quite cheaper than expected.

```
function getPrice(TokenType _tokenType) public view returns (uint256) {
        if (_tokenType == TokenType.NATIVE) {
            return nativeBasePrice;
        } else if (_tokenType == TokenType.USDC) {
            return usdcBasePrice;
        } else {
            return beamBasePrice;
        }
    }
```

It's suggested to use one chainlink oracle to calculate the real-time token price.

[L-04] Referral system can be attacked via the sybil attack

In BeamNode, we have one referral system. When users deposit with one referral for the first time, they can buy one NFT at a discounted price. When users want to deposit with the referrer for the second time, we will not provide one discount for them.

The problem is that this design is quite easy to be bypassed. If users want to deposit with the referrer the second time, they can transfer funds to another address and deposit and get the discount.

```
function _setReferrer(address _user, address _referrer) internal {
        if (users[_user].referrer == address(0) && _isValidReferrer
          (_user, _referrer)) {
            users[_user].referrer = _referrer;
        } else {
            require(_referrer == address(0), InvalidReferral());
        }
    }
```

If this design is necessary, we should add one depositor whitelist for this function.