code4rena

Berachain

Smart Contract Security Assessment

Version 1.0

Audit dates: Jan 28 — Jan 31, 2025

Audited by: windhustler
etherSky

Contents

1. Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2. Executive Summary

2.1 About Berachain

This is a protocol for managing NFT-based vesting streams and social verification rewards on Berachain through a paymaster implementation.

2.2 Scope

Repository	clique-external-bera-contracts/
Commit Hash	f7d0d50380acdfa64f76a7a27327c37a632b9226

2.3 Audit Timeline

DATE	EVENT
Jan 28, 2025	Audit start
Jan 31, 2025	Audit end
Feb 03, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	1
Medium Risk	1
Low Risk	6
Informational	4
Total Issues	12

3. Findings Summary

ID	DESCRIPTION	STATUS
H-1	Rewards can be generated for blacklisted tokens	Resolved
M-1	The `claim` function in the `ClaimBatchProcessor` is vulnerable to front-running	Acknowledged

L-1	Duplicated `require` statement in `WrappedNFT::onERC1155Received` function	Resolved
L-2	Only EOAs are allowed to claim rewards in StreamingNFT	Acknowledged
L-3	Unpausing StreamingNFT should be prohibited if cliffEndTimestamp is set to 0	Resolved
L-4	The claim function can be called by anyone in Distributor1	Acknowledged
L-5	`WrappedNFT::Unwrap` event emitting incorrect information may cause issues in applications relying on events	Resolved
L-6	`WrappedNFT::unwrap` function can be called when the contract is paused	Resolved
I-1	`WrappedNFT::Pause` and `WrappedNFT::Unpause` events are not used	Resolved
I-2	Use SafeTransferLib to ensure secure and reliable token transfers	Acknowledged
I-3	The vestingDuration should not be 0	Resolved
I-4	Insufficient parameter validation	Acknowledged

4. Findings

4.1 High Risk

A total of 1 high risk findings were identified.

[H-1] Rewards can be generated for blacklisted tokens

Severity: High	Status: Resolved

Target

- StreamingNFT.sol

Severity:

- Impact: High
- Likelihood: Medium

Description: In the `StreamingNFT` contract, certain tokens are `blacklisted` in the `constructor`.

```
constructor(
    address _token, // Use address(0) for native token
    uint256 _vestingDuration,
    uint256 _instantUnlockPercentage,
    uint256 _cliffUnlockPercentage,
    address _credentialNFT,
    uint256 _allocationPerNFT,
    uint256[] memory _blacklistedTokenIds
) Ownable(msg.sender) Transferable(_token) PayMaster(address(this)) {
    allocationPerNFT = _allocationPerNFT;
    for (uint256 i = 0; i < _blacklistedTokenIds.length; i++) {
@->      isBlacklistedTokenId[_blacklistedTokenIds[i]] = true;
    }
    _pause();
}
```

The `createStream` function prevents `reward streams` from being created for these `blacklisted tokens`.

```
function createStream(uint256 tokenId) external nonReentrant
whenNotPaused {
    require(claimedTimestamp[tokenId] == 0, "Stream already created");
@-> require(!isBlacklistedTokenId[tokenId], "TokenId is blacklisted");
}
```

However, the `createBatchStream` function lacks this check, allowing `reward streams` to be created for `blacklisted tokens` through this function.

```
function createBatchStream(uint256[] calldata tokenIds, address
onBehalfOfOwner)
    external
    nonReentrant
    whenNotPaused
{
    bool isPayMaster_ = isPayMaster[tx.origin];

    if (isPayMaster_) {
        gasFee = fee;
        require(gasFee != 0, "Gas fee not set");
    }

    for (uint256 i = 0; i < tokenIds.length; i++) {
        require(claimedTimestamp[tokenIds[i]] == 0, "Stream already
created");

        claimedTimestamp[tokenIds[i]] = cliffEndTimestamp;
        }
}
```

Recommendation:

```
function createBatchStream(uint256[] calldata tokenIds, address
onBehalfOfOwner)
    external
    nonReentrant
    whenNotPaused
{
    bool isPayMaster_ = isPayMaster[tx.origin];

    if (isPayMaster_) {
        gasFee = fee;
        require(gasFee != 0, "Gas fee not set");
```

```
    }

    for (uint256 i = 0; i < tokenIds.length; i++) {
        require(claimedTimestamp[tokenIds[i]] == 0, "Stream already
created");
+       require(!isBlacklistedTokenId[tokenIds[i]], "TokenId is
blacklisted");

        claimedTimestamp[tokenIds[i]] = cliffEndTimestamp;
        }
}
```

Berachain: Fixed with [PR-3]

Zenith: Verified.

4.2 Medium Risk

A total of 1 medium risk findings were identified.

[M-1] The `claim` function in the `ClaimBatchProcessor` is vulnerable to front-running

Severity: Medium	Status: Acknowledged

Target

- ClaimBatchProcessor.sol

Severity:

- Impact: Medium
- Likelihood: Low

Description: The `createBatchStream` function can be called by either the `payMaster` or the `token owner`.

```
function createBatchStream(uint256[] calldata tokenIds, address
onBehalfOfOwner)
    external
    nonReentrant
    whenNotPaused
{
    bool isPayMaster_ = isPayMaster[tx.origin];

    if (isPayMaster_) {
        gasFee = fee;
        require(gasFee != 0, "Gas fee not set");
    }

    for (uint256 i = 0; i < tokenIds.length; i++) {
        require(claimedTimestamp[tokenIds[i]] == 0, "Stream already
created");

        claimedTimestamp[tokenIds[i]] = cliffEndTimestamp;

        onbehalfOf = credentialNFT.ownerOf(tokenIds[i]);
        require(onbehalfOf == onBehalfOfOwner, "Not the owner of the
token");
@->     if (!isPayMaster_ && onbehalfOf != tx.origin) {
            revert InvalidOrigin(tx.origin);
        }
```

```
        }

        transfer(onbehalfOf, instantAmountAccum);
        if (payMasterFeeAccum > 0) {
            transfer(tx.origin, payMasterFeeAccum);
        }
    }
```

The `payMaster` typically calls this function to collect `fees`. Additionally, the `ClaimBatchProcessor` includes a `claim` function that allows the `payMaster` to create batch `streams` for multiple `StreamingNFTs`.

```
    function claim(
        uint256[][] calldata _tokenIds,
        uint256 _amount,
        bytes32[] calldata _proof,
        bytes calldata _signature,
        address _onBehalfOf,
        address[] calldata _nfts
    ) external {
        if (_amount > 0) {
            distributor.claim(_proof, _signature, _amount, _onBehalfOf);
        }

        for (uint256 i = 0; i < _nfts.length; i++) {

    IStreamingNFT(streamingNFTs[_nfts[i]]).createBatchStream(_tokenIds[i],
    _onBehalfOf);
        }
    }
```

Since this is a gas-intensive operation, one `token owner` can front-run the transaction by creating a `reward stream` themselves, causing the original transaction to revert. This results in the `payMaster` losing `gas fees`. Worse, the `owner` of the last `NFT` in the batch can deliberately front-run, further increasing the `payMaster`'s `gas loss`.

Recommendation: Keep it as is if this risk is acceptable, or modify the `claim` function as follows:

```
    function claim(
        uint256[][] calldata _tokenIds,
        uint256 _amount,
        bytes32[] calldata _proof,
```

```
        bytes calldata _signature,
        address _onBehalfOf,
        address[] calldata _nfts
    ) external {
        if (_amount > 0) {
            distributor.claim(_proof, _signature, _amount, _onBehalfOf);
        }

        for (uint256 i = 0; i < _nfts.length; i++) {
-
IStreamingNFT(streamingNFTs[_nfts[i]]).createBatchStream(_tokenIds[i],
_onBehalfOf);
+                try
(IStreamingNFT(streamingNFTs[_nfts[i]]).createBatchStream(_tokenIds[i],
_onBehalfOf)) {
+                } catch {}
        }
    }
```

Berachain: Acknowledged

4.3 Low Risk

A total of 6 low risk findings were identified.

[L-1] Duplicated `require` statement in `WrappedNFT::onERC1155Received` function

Severity: Low	Status: Resolved

Target

- WrappedNFT.sol#L35

Severity:

- Impact: Low
- Likelihood: Low

Description: At the beginning of the `WrappedNFT::onERC1155Received` function, there is a check to verify that `msg.sender` is equal to the `oriToken` address. However, after calling the `WrappedNFT::_wrap` function, the same check is performed again. This makes the initial check in `WrappedNFT::onERC1155Received` unnecessary and removing it would reduce gas costs.

Recommendation: Remove the redundant `require` statement in `WrappedNFT::onERC1155Received`:

```
function onERC1155Received(address, address from, uint256 id, uint256
value, bytes calldata)
    external
    nonReentrant
    whenNotPaused
    returns (bytes4)
{
-   require(msg.sender == oriToken, "Invalid sender");

    _wrap(from, id, value);
    return this.onERC1155Received.selector;
}
```

Client: Fixed with PR-4

Zenith: Verified.

[L-2] Only EOAs are allowed to claim rewards in StreamingNFT

Severity: Low	Status: Acknowledged

Target

- StreamingNFT.sol

Severity:

- Impact: Low
- Likelihood: Medium

Description: Currently, `token owners` can `claim rewards` only if they are `EOAs`.

```
function _claimVestedRewards(uint256 streamId, uint256
_vestingEndTimestamp) internal {
    uint256 claimableAmount = _getClaimableRewards(streamId,
_vestingEndTimestamp);
    address beneficiary = credentialNFT.ownerOf(streamId);

@-> if (tx.origin != beneficiary) {
        revert InvalidOrigin(tx.origin);
    }
}
```

However, `paymasters` can create `reward streams` regardless of whether the `owners` are `EOAs`.

```
function createStream(uint256 tokenId) external nonReentrant
whenNotPaused {
    require(claimedTimestamp[tokenId] == 0, "Stream already created");
    require(!isBlacklistedTokenId[tokenId], "TokenId is blacklisted");

    claimedTimestamp[tokenId] = cliffEndTimestamp;

    address onbehalfOf = credentialNFT.ownerOf(tokenId);
    bool isPayMaster_ = isPayMaster[tx.origin];

@-> if (!isPayMaster_ && onbehalfOf != tx.origin) {
        revert InvalidOrigin(tx.origin);
    }
}
```

It is unclear whether this behavior is part of the intended design.

Recommendation: Enforce the same rule for these two operations.

Berachain: Acknowledged - we ensure EOAs only.

[L-3] Unpausing StreamingNFT should be prohibited if cliffEndTimestamp is set to 0

Severity: Low	Status: Resolved

Target

- StreamingNFT.sol

Severity:

- Impact: High
- Likelihood: Negligible

Description: The `owner` can `unpause` `StreamingNFT` at any time, regardless of whether `cliffEndTimestamp` is 0.

```
function unpause() external onlyOwner {
    _unpause(); }
```

This could allow anyone to create a `reward stream` indefinitely and claim the entire balance of `StreamingNFT`.

```
function createStream(uint256 tokenId) external nonReentrant
whenNotPaused {
@-> require(claimedTimestamp[tokenId] == 0, "Stream already created");
    require(!isBlacklistedTokenId[tokenId], "TokenId is blacklisted");

@-> claimedTimestamp[tokenId] = cliffEndTimestamp;

    transfer(onbehalfOf, instantAmount);
    emit StreamCreated(tokenId, onbehalfOf, allocationPerNFT, gasFee);
}
```

While `unpausing` when `cliffEndTimestamp` is 0 is unlikely, it is still advisable to prevent this scenario for security and reliability

Recommendation:

```
function unpause() external onlyOwner {
+   require(cliffEndTimestamp != 0, '');
    _unpause(); }
```

Berachain: Fixed with PR-7

Zenith: Verified.

[L-4] The claim function can be called by anyone in Distributor1

Severity: Low	Status: Acknowledged

Target

- Distributor1.sol

Severity:

- Impact: Medium
- Likelihood: Medium

Description: Anyone can call the `claim` function to receive the `fee`.

```
function claim(bytes32[] calldata _proof, bytes calldata _signature,
uint256 _amount, address _onBehalfOf)
    external
{
    if (balance() < _amount) {
        revert InsufficientBalance();
    }
    if (claimed[_onBehalfOf]) revert AlreadyClaimed();
    if (!active) revert NotActive();

    claimed[_onBehalfOf] = true;

    _rootCheck(_proof, _amount, _onBehalfOf);
    _signatureCheck(_amount, _signature, _onBehalfOf);

    uint256 amount = _amount;

@-> if (tx.origin != _onBehalfOf) {
        uint256 _fee = fee;
        require(_fee != 0, "Gas fee not set");
        amount -= _fee;
@->     transfer(tx.origin, _fee);
    }

    transfer(_onBehalfOf, amount);

    emit AirdropClaimed(tx.origin, amount, _onBehalfOf);
}
```

If the `fee` exceeds the `transaction cost`, users may be incentivized to call the function on behalf of others. However, this introduces a risk of front-running, where multiple callers compete to execute the function first, potentially leading to wasted gas fees for unsuccessful transactions.

Recommendation: Restrict access to paymasters, as implemented in `StreamingNFT`.

Berachain: Acknowledged. The function is protected by `signature`. The other cannot call this function unless `_onBehalfOf` actively share the signature with them.

[L-5] `WrappedNFT::Unwrap` event emitting incorrect information may cause issues in applications relying on events

Severity: Low	Status: Resolved

Target

- WrappedNFT.sol#L89

Severity:

- Impact: High/Medium/Low
- Likelihood: High/Medium/Low

Description: The `WrappedNFT::Unwrap` event is defined as `event Unwrap(address indexed collection, uint256 indexed id)`, where the first argument represents the collection address. However, when a user calls the `WrappedNFT::unwrap` function, the event is emitted with `msg.sender` as the first argument. This means the user's address is recorded as the collection address in the event, which can lead to incorrect data in applications that depend on these events.

Recommendation: Ensure the event is emitted with the correct collection address:

```
function unwrap(uint256[] calldata tokenId, address onBehalfOf) external
{
    require(tokenId.length >= 1, "At least one token must be unwrapped");

    for (uint256 i = 0; i < tokenId.length; i++) {
        require(msg.sender == ownerOf(tokenId[i]), "Must be owner");
        _burn(tokenId[i]);
        IERC1155(oriToken).safeTransferFrom(address(this), onBehalfOf,
tokenId[i], 1, bytes(""));
-       emit Unwrap(msg.sender, tokenId[i]);
+       emit Unwrap(oriToken, tokenId[i]);
    }
}
```

Berachain: Fixed with PR-8

Zenith: Verified.

[L-6] `WrappedNFT::unwrap` function can be called when the contract is paused

Severity: Low	Status: Resolved

Target

- WrappedNFT.sol#L82

Severity:

- Impact: Low
- Likelihood: Low

Description: The `whenNotPaused` modifier is not applied to the `WrappedNFT::unwrap` function. This allows users to unwrap their NFTs into ERC1155 tokens even when the contract is paused.

Recommendation: Add the `whenNotPaused` modifier to the `WrappedNFT::unwrap` function:

```
- function unwrap(uint256[] calldata tokenId, address onBehalfOf)
external {
+ function unwrap(uint256[] calldata tokenId, address onBehalfOf)
external whenNotPaused {
```

Berachain: Fixed with PR-9

Zenith: Verified.

4.4 Informational

A total of 4 informational findings were identified.

[I-1] `WrappedNFT::Pause` and `WrappedNFT::Unpause` events are not used

Severity: Informational	Status: Resolved

Target

- WrappedNFT.sol#L26-L27

Severity:

- Impact: Low
- Likelihood: Low

Description: The `WrappedNFT::Pause` and `WrappedNFT::Unpause` events are declared but never used in the `WrappedNFT` contract. Since `WrappedNFT` inherits from `Pausable`, the `Pausable` contract already provides `Paused` and `Unpaused` events, which are emitted when the contract is paused and unpaused.

Recommendation: Remove the unused events:

```
contract WrappedNFT is Pausable, ONFT721, IERC1155Receiver,
ReentrancyGuard {
    ...
-   event Pause();
-   event Unpause();
    ...
}
```

Berachain: Fixed with PR-5

Zenith: Verified.

[I-2] Use SafeTransferLib to ensure secure and reliable token transfers

Severity: Informational	Status: Acknowledged

Target

- Transferable.sol

Severity:

- Impact: Low
- Likelihood: Low

Description: In the `Transferable` contract, the return value of the `token transfer` is checked.

```
function transfer(address recipient, uint256 amount) internal {
    if (token == address(0)) {
        (bool success,) = recipient.call{value: amount}("");
        require(success, "Native token transfer failed");
    } else {
        require(IERC20(token).transfer(recipient, amount), "ERC20
transfer failed");
    }
}
```

However, some tokens do not return a `boolean` value in their `transfer` function. As a result, `Transferable` does not support these types of tokens.

Recommendation: Use `SafeTransferLib` to ensure secure and reliable token transfers

Berachain: Acknowledged. Zero address is applied in this deployment

[I-3] The vestingDuration should not be 0

Severity: Informational	Status: Resolved

Target

- StreamingNFT.sol

Severity:

- Impact: Low
- Likelihood: Low

Description: The `vestingDuration` should not be 0. If set to 0, `vestingEndTimestamp` and `cliffEndTimestamp` will be the same, preventing owners from claiming vested `rewards`.

```
function _getClaimableRewards(uint256 streamId, uint256
vestingEndTimestamp_) internal view returns (uint256) {
    require(block.timestamp > cliffEndTimestamp, "Vesting has not started
yet");

    uint256 lastClaimedTimestamp = claimedTimestamp[streamId];
    uint256 lastClaimedAmount = claimedAmount[streamId];

    require(lastClaimedTimestamp != 0, "Stream not created");

    if (lastClaimedTimestamp >= vestingEndTimestamp_) {
@->      return 0;
    }
}
```

Recommendation: Add the following check in the constructor to enforce this constraint.

```
constructor(
        address _token, // Use address(0) for native token
        uint256 _vestingDuration,
        uint256 _instantUnlockPercentage,
        uint256 _cliffUnlockPercentage,
        address _credentialNFT,
        uint256 _allocationPerNFT,
        uint256[] memory _blacklistedTokenIds
) Ownable(msg.sender) Transferable(_token) PayMaster(address(this)) {
+   require(_vestingDuration > 0, ''); }
```

Berachain: Fixed with PR-6

Zenith: Verified

[I-4] Insufficient parameter validation

Severity: Informational	Status: Acknowledged

Target

- StreamingNFT.sol
- Distributor1.sol

Severity:

- Impact: Low
- Likelihood: Low

Description:

- The paymaster fee should always be smaller than the `instantUnlockAmount` in the `StreamingNFT` contract. Otherwise, the `createStream` and `createBatchStream` functions will revert when calculating `instantAmount -= gasFee`.
- The paymaster fee should be less than each amount in the leaf of the Merkle tree in the `Distributor1` contract. Otherwise, the `claim` function will revert when subtracting the fee from the amount.

Recommendation: Add validation checks for the fee parameter in the `StreamingNFT` contract and make sure the fee is smaller than each amount in the leaf of the merkle tree in the `Distributor1` contract.

Berachain: Acknowledged