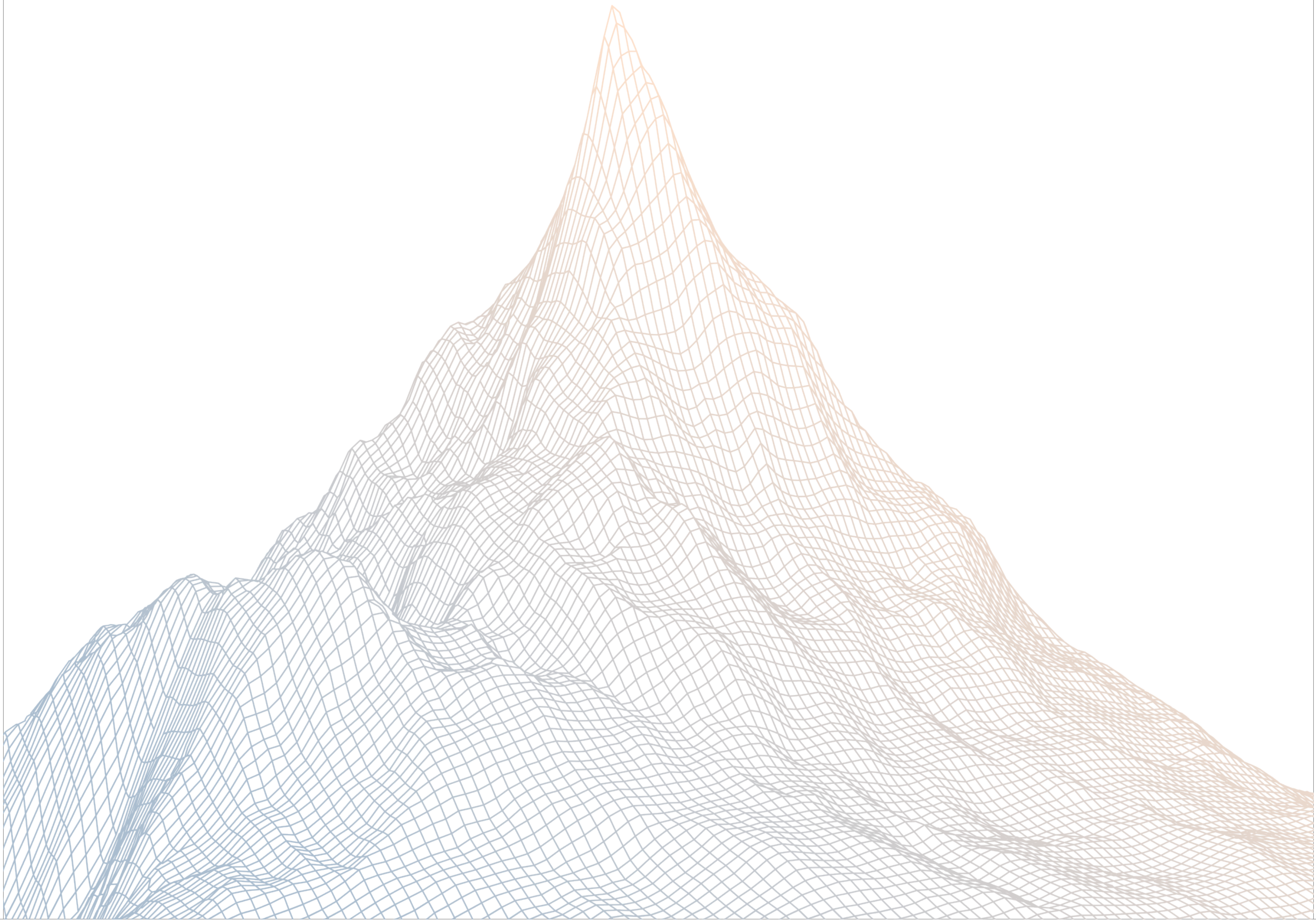
Zenith

Berachain

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: March 11th to March 16th, 2025

AUDITED BY:
ether_sky
matte
peakbolt

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

Berachain is a high-performance EVM-Identical Layer 1 blockchain utilizing Proof-of-Liquidity (PoL) and built on top of the modular EVM-focused consensus client framework BeaconKit.

2.2 Scope

The engagement involved a review of the following targets:

Target	contracts-monorepo
Repository	https://github.com/berachain/contracts-monorepo
Commit Hash	`ed9904b02327bf88f09bef1c28d13cff0e04e616`
Files	`Diff in PR-593`

Target	bribe-boost
Repository	https://github.com/berachain/bribe-boost/
Commit Hash	`68f927e03d6303f1a3151eaa36894ae3eb4bfc8b`
Files	`BribeBoost.sol` `BribeBoostFactory.sol`

2.3 Audit Timeline

March 11, 2025	Audit start
March 16, 2025	Audit end
March 18, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	1
Informational	2
Total Issues	**4**

3

Findings Summary

ID	Description	Status
M-1	Setting an arbitrary commission rate for any validator is possible	Resolved
L-1	The reward vault must remain compatible with all supported tokens	Resolved
I-1	The claim function is vulnerable to front-running attacks	Acknowledged
I-2	updateRewardsMetadata() should validate distribution token	Resolved

4 Findings

4.1 Medium Risk

A total of 1 medium risk findings were identified.

[M-1] Setting an arbitrary commission rate for any validator is possible

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: High

Target

- BeraChef.sol

Description:

The `commission rate` for `validators` experiences `delays`. If a `validator` has no `active rate`, the function `defaults` to a `5% commission rate`.

- BeraChef.sol#L433

```
function _getOperatorCommission(bytes calldata valPubkey)
    internal view returns (uint96) {
    CommissionRate memory operatorCommission = valCommission[valPubkey];
    // If the operator commission was never set, default is 5%.
    if (operatorCommission.activationBlock == 0)
    return DEFAULT_COMMISSION_RATE;
    return operatorCommission.commissionRate;
}
```

However, in the `activateQueuedValCommission` function, it is possible to activate an arbitrary `commission rate` due to an error in the `activation block check`. Specifically, the function does not verify whether `blockNumberLast` has been set.

- BeraChef.sol#L261

```
function activateQueuedValCommission(bytes calldata valPubkey) external {
    QueuedCommissionRateChange storage qcr = valQueuedCommission[valPubkey];
    (uint32 blockNumberLast, uint96 commissionRate) = (qcr.blockNumberLast,
    qcr.commissionRate);
    uint32 activationBlock = uint32(blockNumberLast
    + commissionChangeDelay);

@-> if (block.number < activationBlock) {
        CommissionChangeDelayNotPassed.selector.revertWith();
    }

    uint96 oldCommission = _getOperatorCommission(valPubkey);
    valCommission[valPubkey] = CommissionRate({ activationBlock:
    activationBlock, commissionRate: commissionRate });
    emit ValCommissionSet(valPubkey, oldCommission, commissionRate);
    // delete the queued commission
    delete valQueuedCommission[valPubkey];
}
```

As a result, anyone can easily activate any `rate` for a `validator` without a queued `commission rate`.

Recommendations:

```
function activateQueuedValCommission(bytes calldata valPubkey) external {
    QueuedCommissionRateChange storage qcr = valQueuedCommission[valPubkey];
    (uint32 blockNumberLast, uint96 commissionRate) = (qcr.blockNumberLast,
    qcr.commissionRate);
    uint32 activationBlock = uint32(blockNumberLast
    + commissionChangeDelay);

-^^Iif (block.number < activationBlock) {
+^^Iif (blockNumberLast == 0 || block.number < activationBlock) {
        CommissionChangeDelayNotPassed.selector.revertWith();
    }

    uint96 oldCommission = _getOperatorCommission(valPubkey);
    valCommission[valPubkey] = CommissionRate({ activationBlock:
    activationBlock, commissionRate: commissionRate });
    emit ValCommissionSet(valPubkey, oldCommission, commissionRate);
    // delete the queued commission
    delete valQueuedCommission[valPubkey];
}
```

Berachain: Resolved with @229e1e71d4...

Zenith: Verified.

4.2 Low Risk

A total of 1 low risk findings were identified.

[L-1] The reward vault must remain compatible with all supported tokens

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- RewardVault.sol

Description:

In the `_processIncentives` function, `incentive tokens` need to be `approved` before calling the `receiveIncentive` function of the `bgtIncentiveDistributor`. Once the `approval` is granted, the `receiveIncentive` function is invoked.

- RewardVault.sol#L467-L468

```
function _processIncentives(bytes calldata pubkey, uint256 bgtEmitted)
    internal {
    for (uint256 i; i < whitelistedTokensCount; ++i) {
    if (amount > 0) {
            // Transfer the remaining amount of the incentive to the
    bgtIncentiveDistributor contract for
            // distribution among BGT boosters.
            // give the bgtIncentiveDistributor the allowance to transfer the
    incentive token.
            bytes memory data = abi.encodeCall(IERC20.approve,
    (bgtIncentiveDistributor, amount));
            (bool success,) = token.call(data);
            if (success) {
                // reuse the already defined data variable to avoid stack too
    deep error.
                data
    = abi.encodeCall(IBGTIncentiveDistributor.receiveIncentive, (pubkey,
    token, amount));
```

```
                (success,) = bgtIncentiveDistributor.call(data);
                if (success) {
                    amountRemaining -= amount;
                    emit BGTBoosterIncentivesProcessed(pubkey, token,
    bgtEmitted, amount);
                } else {
                    emit BGTBoosterIncentivesProcessFailed(pubkey, token,
    bgtEmitted, amount);
                }
            }
            // if the approve fails, log the failure in sending the incentive
    to the bgtIncentiveDistributor.
            else {
                emit BGTBoosterIncentivesProcessFailed(pubkey, token,
    bgtEmitted, amount);
            }
        }
        incentive.amountRemaining = amountRemaining;
    }
}
```

However, if the external call to the `receiveIncentive` function fails, the `approved` tokens remain unused. In such cases, future calls to `_processIncentives` will revert if the `incentive token` behaves like `USDT`, where `approvals` from a non-zero value are reverted.

Recommendations:

Reset the `approval` to `0` if the call fails.

Berachain: Resolved with PR-596

Zenith: Verified

4.3 Informational

A total of 2 informational findings were identified.

[I-1] The claim function is vulnerable to front-running attacks

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- BGTIncentiveDistributor.sol

Description:

The `claim` function can be called by anyone, making it vulnerable to front-running attacks.

- BGTIncentiveDistributor.sol#L140

```
function claim(Claim[] calldata _claims)
    external nonReentrant whenNotPaused {
    uint256 cLen = _claims.length;

    if (cLen == 0) InvalidArray.selector.revertWith();

    for (uint256 i; i < cLen;) {
        _claim(_claims[i].identifier, _claims[i].account, _claims[i].amount,
    _claims[i].merkleProof);

        unchecked {
            ++i;
        }
    }
}
```

For instance, if a user intends to claim rewards for 10 users, an attacker could front-run the transaction by claiming the last user's rewards. As a result, the `_claim` function would revert because the `lifeTimeAmount` would have already been updated.

- BGTIncentiveDistributor.sol#L172

```
function _claim(bytes32 _identifier, address _account, uint256 _amount,
    bytes32[] calldata _merkleProof) private {
    Reward memory reward = rewards[_identifier];

    if (reward.merkleRoot == 0) InvalidMerkleRoot.selector.revertWith();
    if (reward.activeAt > block.timestamp)
    RewardInactive.selector.revertWith();

    uint256 lifeTimeAmount = claimed[_identifier][_account] + _amount;

    // Verify the merkle proof
    if (
        !MerkleProof.verifyCalldata(
            _merkleProof, reward.merkleRoot,
    keccak256(abi.encodePacked(_account, lifeTimeAmount))
        )
    ) InvalidProof.selector.revertWith();
}
```

Consequently, the original caller would lose funds in the form of gas fees.

Berachain: Acknowledged

[I-2] `updateRewardsMetadata()` should validate distribution token

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- BribeBoost.sol#L87-L108

Description:

`updateRewardsMetadata()` can be used to update an existing reward metadata with new values for the `merkleRoot` and `activeAt` based on the `distribution.identifier`.

To prevent any accidental update to the wrong distribution, it is suggested to validate the `reward.token == distribution.token`, and revert otherwise.

Recommendations:

Consider the following change,

```
function updateRewardsMetadata(Common.Distribution[]
calldata _distributions)
    external
    onlyRole(DEFAULT_ADMIN_ROLE)
{
    //@audit-ok non-zero distributions
    uint256 dLen = _distributions.length;

    if (dLen == 0) revert Errors.InvalidDistribution();

    //@audit-ok allow claims 3hrs after update
    uint256 activeAt = block.timestamp + activeTimerDuration;

    for (uint256 i; i < dLen;) {
        // Update the metadata and start the timer until the rewards will
be active/claimable
        Common.Distribution calldata distribution = _distributions[i];
        Reward storage reward = rewards[distribution.identifier];
        reward.merkleRoot = distribution.merkleRoot;
        reward.proof = distribution.proof;
        reward.activeAt = activeAt;

        // Should only be set once per identifier
        if (reward.token == address(0)) {
            reward.token = distribution.token;
        }
       else if(reward.token != distribution.token)
          revert Error.InvalidDistributionToken();


        emit RewardMetadataUpdated(
            distribution.identifier, distribution.token,
distribution.merkleRoot, distribution.proof, activeAt
        );

        unchecked {
            ++i;
        }
    }
}
```

Berachain: Fixed in @81b1c78fe33f...

Zenith: Verified