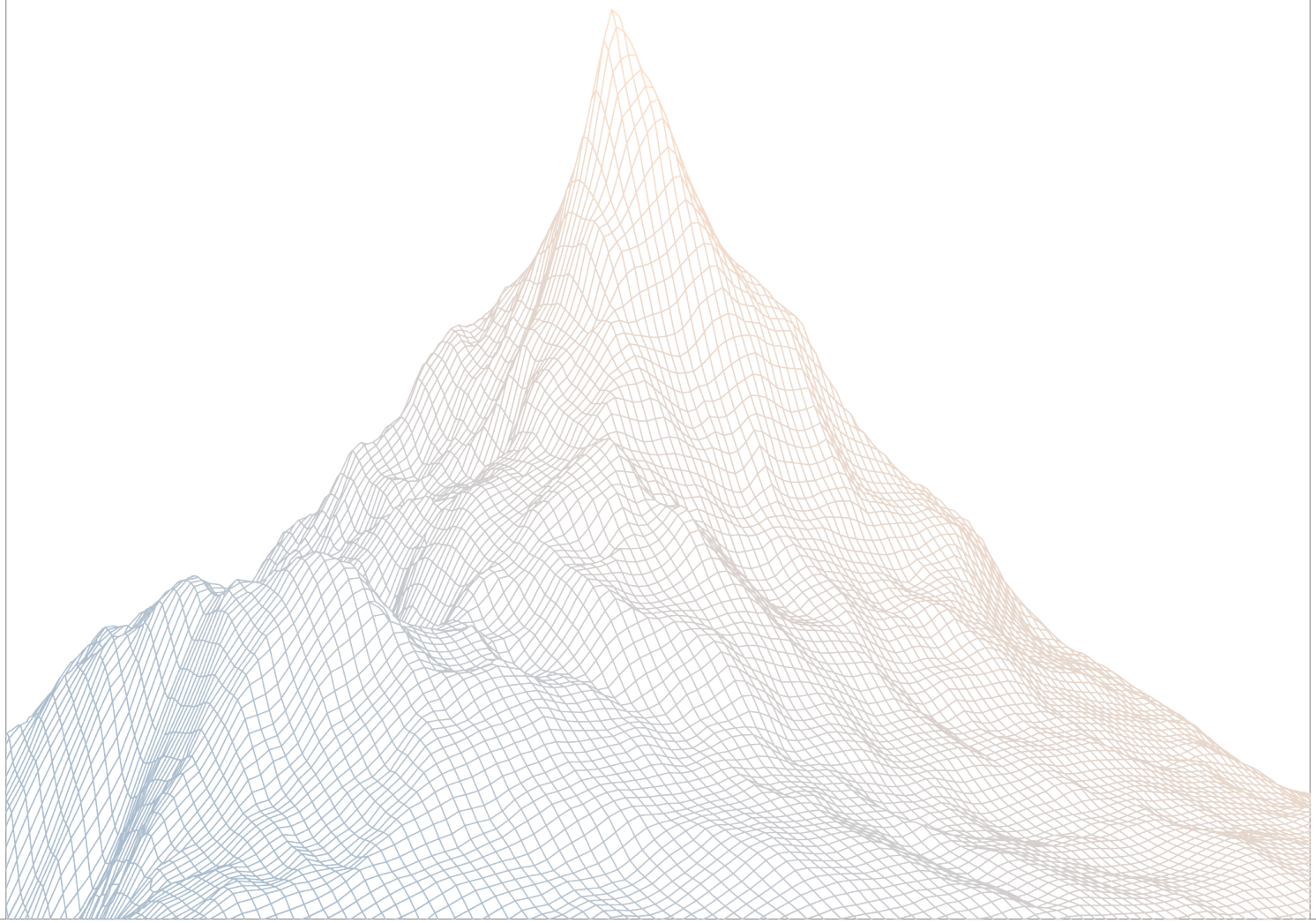
Zenith

Berachain

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: April 9th to April 11th, 2025

AUDITED BY: Ether_sky
Matte

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	`d6488fabbb4a0c0cb43955cec5b2c9b1a0d46d3e`
Files	`PR-606` `PR-610`

2.3 Audit Timeline

April 9, 2025	Audit start
April 11, 2025	Audit end
April 11, 2025	Report published

2.4 Issues Found

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

3

Findings Summary

ID	Description	Status
L-1	The accountIncentives function lacks check to prevent spamming	Resolved
L-2	Invalid reward allocations can be overwritten	Resolved

4

Findings

4.1 Low Risk

A total of 2 low risk findings were identified.

[L-1] The accountIncentives function lacks check to prevent spamming

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

Target

- RewardVault.sol

Severity:

- Impact: Low
- Likelihood: Medium

Description:

The `addIncentive` function allows the `manager` to add `incentives` and modify `incentive rates`. Per the `code comments`, the added amount should exceed `minIncentiveRate` to prevent spamming.

- RewardVault.sol#L372

```
function addIncentive(
    address token,
    uint256 amount,
    uint256 incentiveRate
)
    external
    nonReentrant
    onlyWhitelistedToken(token)
{
    // The incentive amount should be equal to or greater than the
    `minIncentiveRate` to avoid spamming.
    // If the `minIncentiveRate` is 100 USDC/BGT, the amount should be at
```

```
    least 100 USDC.
    if (amount < minIncentiveRate)
    AmountLessThanMinIncentiveRate.selector.revertWith();
}
```

However, the `accountIncentives` function lacks this validation, enabling the `manager` to add arbitrary amounts without restriction.

- RewardVault.sol#L405-L406

```
function accountIncentives(address token, uint256 amount)
    external nonReentrant onlyWhitelistedToken(token) {
    Incentive storage incentive = incentives[token];
    (uint256 incentiveRateStored, uint256 amountRemainingBefore,
    address manager) =
        (incentive.incentiveRate, incentive.amountRemaining,
    incentive.manager);

    // Only allow the incentive token manager to account for cumulated
    incentives.
    if (msg.sender ≠ manager) NotIncentiveManager.selector.revertWith();

    uint256 incentiveBalance = IERC20(token).balanceOf(address(this));
    if (amount > incentiveBalance - amountRemainingBefore)
    NotEnoughBalance.selector.revertWith();

    incentive.amountRemaining += amount;

    // TBD: may need another type of event to distinguish between the two
    operations.
    emit IncentiveAdded(token, msg.sender, amount, incentiveRateStored);
}
```

Recommendations:

Add the same check to the `accountIncentives` function.

Berachain: Resolved with @466029238a...

Zenith: Resolved

[L-2] Invalid reward allocations can be overwritten

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

Target

- BeraChef.sol

Description:

Suppose the `maxWeightPerVault` is set to `70%` and one `validator` has valid `reward allocation` like below: `{ vault1: 50%, vault2: 50% }` The `validator` queues new valid `reward allocation` (`{ vault1: 65%, vault2: 35% }`) using `queueNewRewardAllocation` function. The `owner` updates the `maxWeightPerVault` to `60%` and the `defaultRewardAllocation` is still valid under this `60%`.

- BeraChef.sol#L157

```
function setMaxWeightPerVault(uint96 _maxWeightPerVault)
    external onlyOwner {
    if (_maxWeightPerVault == 0 || _maxWeightPerVault > ONE_HUNDRED_PERCENT)
    {
        InvalidMaxWeightPerVault.selector.revertWith();
    }

    // Note: no need to check `_maxWeightPerVault *
    maxNumWeightsPerRewardAllocation ≥ ONE_HUNDRED_PERCENT`
    // since a _maxWeightPerVault too low would invalidate any valid default
    reward allocation.

    maxWeightPerVault = _maxWeightPerVault;

    // Check if the change could invalidate the default reward allocation
    if (!_checkIfStillValid(defaultRewardAllocation.weights)) {
        InvalidateDefaultRewardAllocation.selector.revertWith();
    }

    emit MaxWeightPerVaultSet(_maxWeightPerVault);
}
```

After the `rewardAllocationBlockDelay` expires, the previously queued `reward allocation` is activated using `activateReadyQueuedRewardAllocation` function.

- BeraChef.sol#L298

```
function activateReadyQueuedRewardAllocation(bytes calldata valPubkey)
    external onlyDistributor {
    if (!isQueuedRewardAllocationReady(valPubkey, block.number)) return;
    RewardAllocation storage qra = queuedRewardAllocations[valPubkey];
    uint64 startBlock = qra.startBlock;
    activeRewardAllocations[valPubkey] = qra;
    emit ActivateRewardAllocation(valPubkey, startBlock, qra.weights);
    // delete the queued reward allocation
    delete queuedRewardAllocations[valPubkey];
}
```

This new `reward allocation({ vault1: 65%, vault2: 35% })` is invalid under `60% maxWeightPerVault` but it overwrites the old valid `reward allocation({ vault1: 50%, vault2: 50% })`. Consequently, the `getActiveRewardAllocation` function returns `defaultRewardAllocation` and the `distributor` distributes `BGT` based on this `allocation`.

Recommendations:

```
function activateReadyQueuedRewardAllocation(bytes calldata valPubkey)
    external onlyDistributor {
    if (!isQueuedRewardAllocationReady(valPubkey, block.number)) return;
    RewardAllocation storage qra = queuedRewardAllocations[valPubkey];
    uint64 startBlock = qra.startBlock;

+^^Iif (_checkIfStillValid(qra.weights)) {
        activeRewardAllocations[valPubkey] = qra;
        emit ActivateRewardAllocation(valPubkey, startBlock, qra.weights);
+^^I}

    // delete the queued reward allocation
    delete queuedRewardAllocations[valPubkey];
}
```

Berachain: Acknowledged, we consider this as expected behaviour. in any case we expect changes to this parameter to be well anticipated and validators to have plenty of time to adapt their reward allocation. Basically having 2 * rewardAllocationBlockDelay will be enough to let them activate a possibly will-be-invalid allocation and change to a compliant one. Since changing maxWeightPerVault will require a governance proposal this will take way longer than 2 * rewardAllocationBlockDelay to become effective.