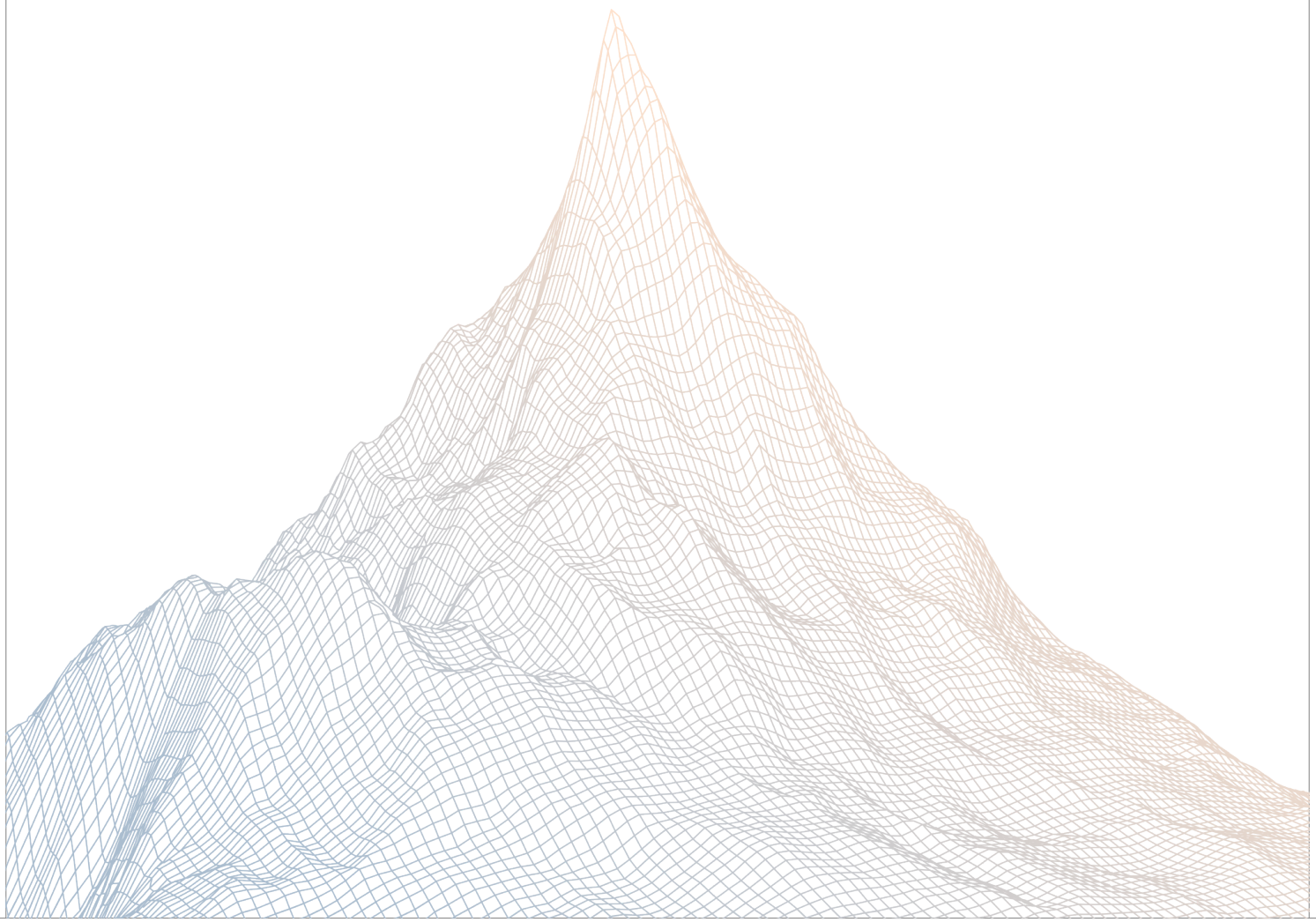
Zenith

Berachain

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: May 16th to May 20th, 2025

AUDITED BY: Matte
ether_sky

Contents

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

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	`ba521fbed47e15195255f3dce219376ef987525f`
Files	`PR-624`

Target	contracts-monorepo
Repository	https://github.com/berachain/contracts-monorepo
Commit Hash	`e2040ef21f275bf4fe7172778898dbe6978f94ed`
Files	`PR-627`

2.3 Audit Timeline

May 16, 2025	Audit start
May 20, 2025	Audit end
May 23, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	1
Informational	1
Total Issues	**3**

3

Findings Summary

ID	Description	Status
M-1	Funds can be permanently stuck in HoneyFactoryPythWrapper due to recapitalization amount adjustments	Resolved
L-1	Any remaining ETH should be refunded after the Pyth update is completed	Resolved
I-1	lastRewardDurationChangeTimestamp should be set in the initialize function	Resolved

4

Findings

4.1 Medium Risk

A total of 1 medium risk findings were identified.

[M-1] Funds can be permanently stuck in HoneyFactoryPythWrapper due to recapitalization amount adjustments

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

Target

- HoneyFactoryPythWrapper.sol

Description

The `HoneyFactoryPythWrapper` contract contains a vulnerability in its `recapitalize()` function that can lead to user funds being permanently stuck in the contract.

When a user calls `recapitalize()` on the `HoneyFactoryPythWrapper`, the function:

1. Updates Pyth price feeds via `_updatePyth(updateData)`
2. Transfers assets from the user to the wrapper contract via `_getForFactory(asset, amount)`
3. Calls `recapitalize(asset, amount)` on the `HoneyFactory` contract

The issue occurs because the `HoneyFactory.recapitalize()` function performs several checks and may adjust the amount of assets it actually processes:

- It verifies the current balance against a target balance threshold
- It checks if the asset is pegged
- Most critically, it adjusts `amount` downward if `currentBalance + amount > targetBalance`

However, the `HoneyFactoryPythWrapper` does not account for this potential adjustment. When the `HoneyFactory` reduces the `amount` parameter, it only processes this smaller amount, but the wrapper has already transferred the full original amount from the user.

This discrepancy results in the excess assets remaining trapped in the `HoneyFactoryPythWrapper` contract with no mechanism to retrieve them.

For example, if a user attempts to recapitalize 100 tokens but the `HoneyFactory` only needs 60 to reach its target balance, the remaining 40 tokens will be stuck in the wrapper contract.

Recommendations

Modify the `recapitalize()` function in `HoneyFactoryPythWrapper` to handle potential amount adjustments and return any excess funds to the user.

Berachain: Resolved with @6b825985d6...

Zenith: Verified.

4.2 Low Risk

A total of 1 low risk findings were identified.

[L-1] Any remaining ETH should be refunded after the Pyth update is completed

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

Target

- HoneyFactoryPythWrapper.sol

Description:

In the `HoneyFactoryPythWrapper`, users are required to update `Pyth` before performing operations like `minting`.

- HoneyFactoryPythWrapper.sol#L51

```
function mint(
    bytes[] calldata updateData,
    address asset,
    uint256 amount,
    address receiver,
    bool expectBasketMode
)
    external
    payable
    returns (uint256 minted)
{
    _updatePyth(updateData);
}
```

To do this, they must `deposit` some `ETH` to cover the `update fee`.

- HoneyFactoryPythWrapper.sol#L173

```
function _updatePyth(bytes[] memory updateData) internal {
    uint256 fee = IPyth(pyth).getUpdateFee(updateData);
    if (address(this).balance < fee)
    InsufficientBalanceToPayPythFee.selector.revertWith();
    IPyth(pyth).updatePriceFeeds{ value: fee }(updateData);

    emit PythOracleUpdated(fee);
}
```

However, there is currently no logic in place to `refund` any `unused ETH` after the `update`.

Recommendations:

Refund any remaining ETH to `msg.sender` within the `_updatePyth` function.

Berachain: Resolved with @007f8d11c7...

Zenith: Verified.

4.3 Informational

A total of 1 informational findings were identified.

[I-1] lastRewardDurationChangeTimestamp should be set in the initialize function

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

Target

- RewardVault.sol

Description:

`lastRewardDurationChangeTimestamp` is not being set in the `initialize` function

- RewardVault.sol#L106

```
function initialize(
    address _beaconDepositContract,
    address _bgt,
    address _distributor,
    address _stakingToken
)
    external
    initializer
{
    __FactoryOwnable_init(msg.sender);
    __Pausable_init();
    __ReentrancyGuard_init();
    __StakingRewards_init(_stakingToken, _bgt, 7 days);
    maxIncentiveTokensCount = 3;
    // slither-disable-next-line missing-zero-check
    distributor = _distributor;
    beaconDepositContract = IBeaconDeposit(_beaconDepositContract);
    emit DistributorSet(_distributor);
    emit MaxIncentiveTokensCountUpdated(maxIncentiveTokensCount);
}
```

Recommendations:

```
function initialize(
    address _beaconDepositContract,
    address _bgt,
    address _distributor,
    address _stakingToken
)
    external
    initializer
{
+^^IlastRewardDurationChangeTimestamp = block.timestamp;
}
```

Berachain: Resolved with @a5ecff0aa5...

Zenith: Verified.