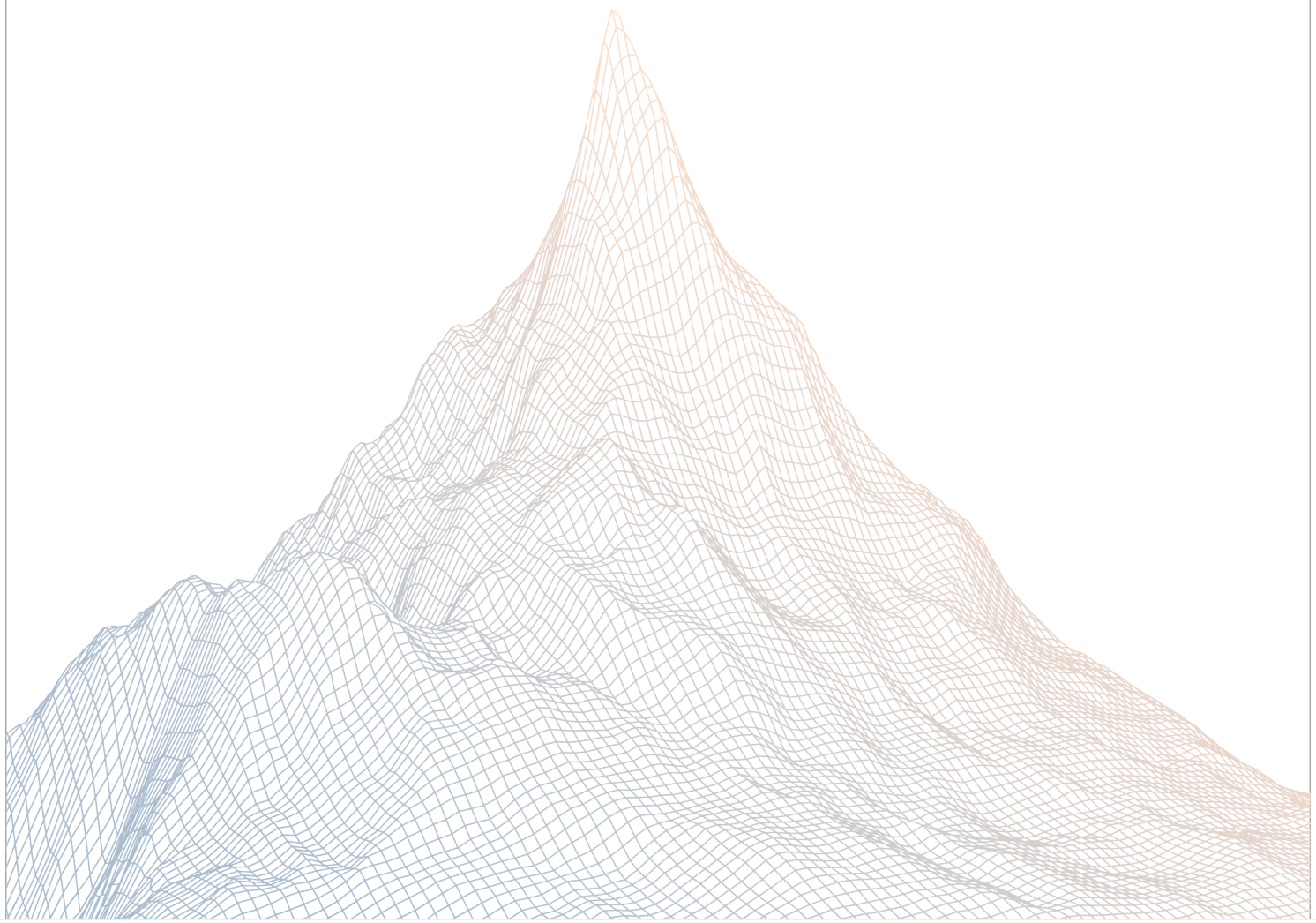
Zenith

Berachain

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: August 25th to August 26th, 2025

AUDITED BY: Matte
ether_sky

Contents

1 Introduction 2

1.1 About Zenith 3

1.2 Disclaimer 3

1.3 Risk Classification 3

2 Executive Summary 3

2.1 About Berachain 4

2.2 Scope 4

2.3 Audit Timeline 5

2.4 Issues Found 5

3 Findings Summary 5

4 Findings 6

4.1 Low Risk 7

4.2 Informational 9

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-internal
Repository	https://github.com/berachain/contracts-internal
Commit Hash	`6f38831f5080ed3081a970d38b3fc5bcd5ba78a1`
Files	`Changes in PR-44`

2.3 Audit Timeline

August 25, 2025	Audit start
August 26, 2025	Audit end
August 28, 2025	Report published

2.4 Issues Found

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

3

Findings Summary

ID	Description	Status
L-1	Legacy withdrawals are allowed when the vault is paused	Resolved
I-1	The maximum deposit validation can be skipped	Resolved

4

Findings

4.1 Low Risk

A total of 1 low risk findings were identified.

[L-1] Legacy withdrawals are allowed when the vault is paused

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

Target

- WBERAStakerVault.sol

Description:

All operations, including `deposits` and `queued withdrawals`, use the `nonReentrant` and `whenNotPaused` modifiers.

- WBERAStakerVault.sol#L211-L212

```
function queueWithdraw(
    uint256 assets,
    address receiver,
    address owner
)
    external
    nonReentrant
    whenNotPaused
    returns (uint256, uint256)
```

However, `legacy withdrawals` do not have these modifiers.

Recommendations:

```
function _withdraw(
    address caller,
    address receiver,
    address owner,
```

```
    uint256 assets,
    uint256 shares
)
    internal
    override
     nonReentrant
     whenNotPaused
```

Berachain: Resolved with @9528dbebb4...

Zenith: Verified.

4.2 Informational

A total of 1 informational findings were identified.

[I-1] The maximum deposit validation can be skipped

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

Target

- WBERAStakerVault.sol

Description:

The `WBERAStakerVault` is an `ERC4626 vault` that enforces a `maximum deposit check` when `assets` are `deposited`. This check is important: while the `maximum deposit amount` is currently set to `type(uint256).max`, the functionality could be updated in the future. For that reason, the `validation` is preserved across all functions.

- WBERAStakerVault.sol#L151-L154

```
function depositNative(uint256 assets, address receiver)
    public payable whenNotPaused returns (uint256) {
    if (msg.value ≠ assets) InsufficientNativeValue.selector.revertWith();
    uint256 maxAssets = maxDeposit(receiver);
    if (assets > maxAssets) {
        revert ERC4626ExceededMaxDeposit(receiver, assets, maxAssets);
    }

    uint256 shares = previewDeposit(assets);
    _depositNative(_msgSender(), receiver, assets, shares);
    return shares;
}
```

However, this check can effectively be bypassed by users. For example:

- `User A` has a `maximum deposit limit` of `100`.
- `User B` has a `maximum deposit limit` of `1000` and `deposited 500 assets`.
- `User B` approves `User A` to use their `balance`.

- `User A` queues a `withdrawal` of `500 assets` using the `queueWithdraw` function.
- WBERAStakerVault.sol#L394

```
function _queueWithdraw(
    address caller,
    address receiver,
    address owner,
    uint256 assets,
    uint256 shares
)
    internal
    returns (uint256)
{
    reservedAssets += assets;

    // Mint the withdrawal request NFT
    uint256 requestId = withdrawalRequests721.mint(caller, receiver, owner,
    assets, shares);
}
```

- An `NFT` is created for `User A`.
- WBERAStakerVaultWithdrawalRequest.sol#L165

```
function _mintWithdrawalRequest(
    address caller,
    address receiver,
    address owner,
    uint256 assets,
    uint256 shares
)
    internal
    returns (uint256)
{
    _safeMint(caller, newRequestId);
    withdrawalRequests[newRequestId] = newRequest;
}
```

- `User A` then cancels this `withdrawal` using the `cancelQueuedWithdrawal` function.
- WBERAStakerVault.sol#L243

```
function cancelQueuedWithdrawal(uint256 requestId)
    external nonReentrant whenNotPaused {
    if (msg.sender ≠
    IERC721(address(withdrawalRequests721)).ownerOf(requestId)) {
```

```
        OnlyNFTOwnerAllowed.selector.revertWith();
    }
    uint256 assets = request.assets; // cache the assets
    uint256 newSharesToMint = previewDeposit(assets);

    reservedAssets -= assets;
    // mint the shares to the caller
    _mint(msg.sender, newSharesToMint);
}
```

At this point, the `500 assets` are `re-deposited` into the `vault` on behalf of `User A`, allowing `User A` to hold more than their `100-asset limit`.

Recommendations:

```
function cancelQueuedWithdrawal(uint256 requestId)
    external nonReentrant whenNotPaused {
    // only NFT owner can cancel the withdrawal request.
    // ownerOf reverts with ERC721NonexistentToken if the requestId does not
    exist.
    if (msg.sender ≠
    IERC721(address(withdrawalRequests721)).ownerOf(requestId)) {
        OnlyNFTOwnerAllowed.selector.revertWith();
    }
    // get the request from the withdrawalRequests721 contract.
    IWBERAStakerVaultWithdrawalRequest.WithdrawalRequest memory request =
        withdrawalRequests721.getRequest(requestId);
    // burn the NFT and delete the request from the mapping
    withdrawalRequests721.cancel(requestId);
    // mint the new shares to the NFT owner based on locked assets during the
    queuing of the request.
    uint256 assets = request.assets; // cache the assets

    uint256 maxAssets = maxDeposit(msg.sender);
    if (assets > maxAssets) {
        revert ERC4626ExceededMaxDeposit(msg.sender, assets, maxAssets);
    }

    uint256 newSharesToMint = previewDeposit(assets);

    reservedAssets -= assets;
    // mint the shares to the caller
    _mint(msg.sender, newSharesToMint);
    emit WithdrawalCancelled(msg.sender, request.owner, assets,
```

```
        request.shares, newSharesToMint);
    }
```

Berachain: Resolved with @de31ac452e...

Zenith: Verified.