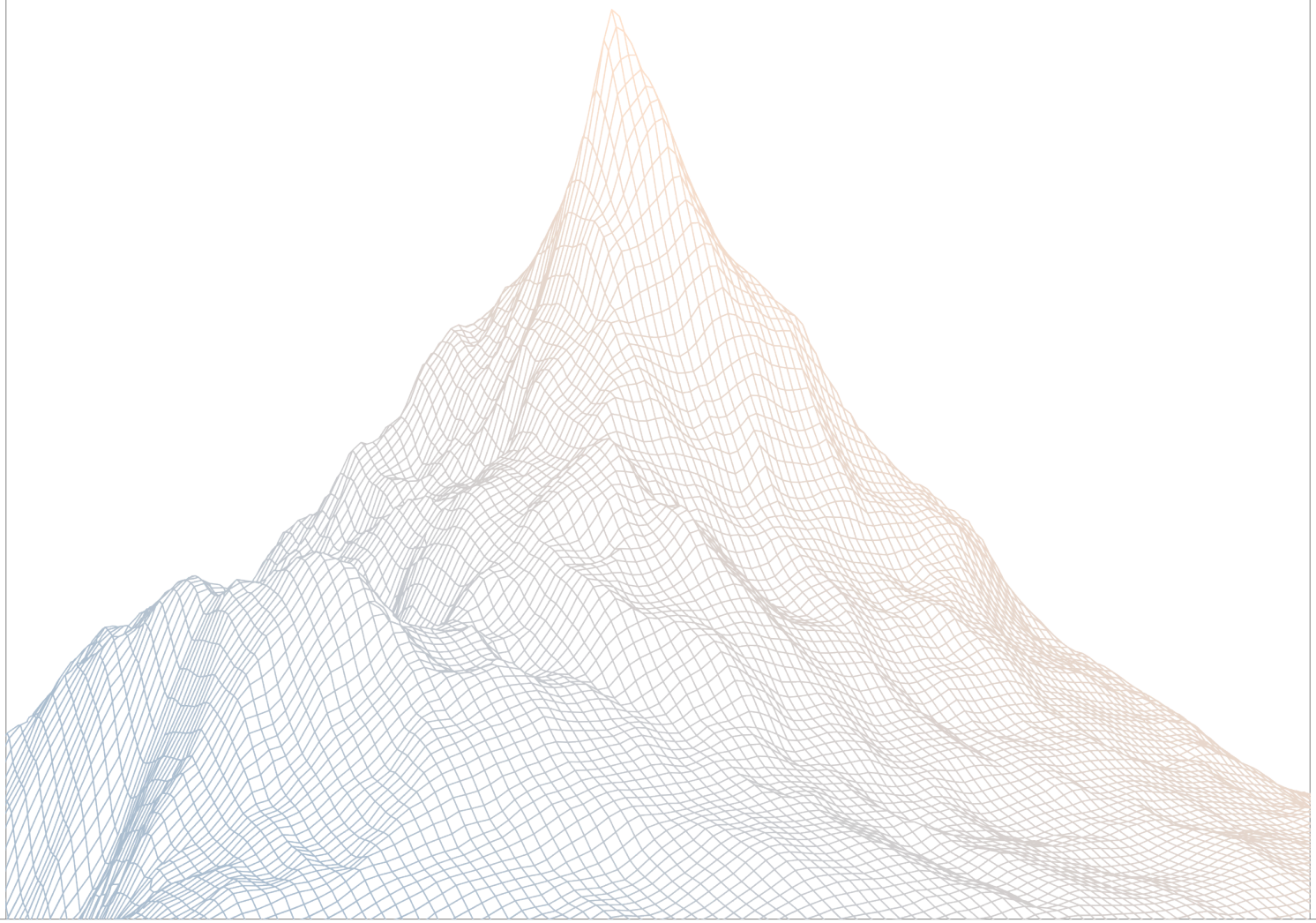
Zenith

Berachain

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: June 25th to June 27th, 2025

AUDITED BY: Mario Poneder
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-meta-aggregator
Repository	https://github.com/berachain/contracts-meta-aggregator
Commit Hash	`ae303d3c438a53c5c7d1b23c4479f767cdf60ffb`
Files	`Changes in the PR-8`

2.3 Audit Timeline

June 25, 2025	Audit start
June 27, 2025	Audit end
July 5, 2025	Report published

2.4 Issues Found

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

3

Findings Summary

ID	Description	Status
I-1	Silent truncation of inputData.amountIn to uint160 in Permit2 approval	Resolved
I-2	Removing PERMIT2_EXPIRATION could simplify the logic	Resolved

4

Findings

4.1 Informational

A total of 2 informational findings were identified.

[I-1] Silent truncation of `inputData.amountIn` to `uint160` in Permit2 approval

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

Target

- src/MetaAggregatorExecutor.sol#L56

Description:

In `MetaAggregatorExecutor.execute`, the `inputData.amountIn` value is cast directly to `uint160` when calling `IPermit2(PERMIT2_ADDRESS).approve`. If `inputData.amountIn` exceeds `type(uint160).max`, this will silently truncate the value, potentially resulting in an insufficient approval amount. This could cause swaps to fail or behave unexpectedly, especially for tokens with very high decimals or large transfer amounts.

Recommendations:

It is recommended to use `toUint160` from `SafeCastLib` to safely cast and revert on overflow. Alternatively, add an explicit check and define a custom error `AmountInOverflow()`: “‘solidity if (inputData.amountIn > type(uint160).max) revert AmountInOverflow(); “‘

Berachain: Resolved with @ae303d3c438...

Zenith: Verified.

[I-2] Removing PERMIT2_EXPIRATION could simplify the logic

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

Target

- MetaAggregatorExecutor.sol

Description:

When `swapping` uses the `PERMIT2` contract, the `expiration` is currently set to `10 seconds`.

- MetaAggregatorExecutor.sol#L22C29-L22C47

```
uint48 private constant PERMIT2_EXPIRATION = 10 seconds;

function execute(
    SwapInputData calldata inputData,
    SwapOutputData calldata outputData,
    AggregatorData calldata aggregatorData
)
    external
    returns (uint256 amountOut)
{
    if (inputData.isPermit2Approval) {
        inputData.tokenIn.safeApprove(PERMIT2_ADDRESS, inputData.amountIn);
        if (inputData.permit2SpenderAddress == address(0))
    revert ZeroAddress();
        IPermit2(PERMIT2_ADDRESS).approve(
            inputData.tokenIn,
            inputData.permit2SpenderAddress,
            uint160(inputData.amountIn),
            uint48(block.timestamp + PERMIT2_EXPIRATION)
        );
    } else {
        inputData.tokenIn.safeApprove(aggregatorData.aggregator,
    inputData.amountIn);
    }
    if (inputData.isPermit2Approval) {
        inputData.tokenIn.safeApprove(PERMIT2_ADDRESS, 0);
```

```
    } else {
        inputData.tokenIn.safeApprove(aggregatorData.aggregator, 0);
    }
}
```

However, since the `swap` and `PERMIT2 approval` occur within the same transaction and the `approval` is reset at the end, setting a `10-second expiration` is unnecessary. Setting the `expiration` to the current time should be sufficient.

Recommendations:

```
uint48 private constant PERMIT2_EXPIRATION = 10 seconds;

IPermit2(PERMIT2_ADDRESS).approve(
    inputData.tokenIn,
    inputData.permit2SpenderAddress,
    uint160(inputData.amountIn),
    uint48(block.timestamp + PERMIT2_EXPIRATION)
    uint48(block.timestamp)
);
```

Berachain: Resolved with @ae303d3c438...

Zenith: Verified.