fuzzland

EulerSwap Audit Report

Signature: 1b29845cc5fc8f3f0faad4112b51a9324955689468a924162683fbc839e70932

EulerSwap Audit report

Executive Summary

On Feb 17, 2025, the Euler team engaged Fuzzland to conduct a thorough security audit of their project. The primary objective was identifying and mitigating potential security vulnerabilities, risks, and coding issues to enhance the project's robustness and reliability. Fuzzland conducted this assessment over 6 person days, involving 2 engineers who reviewed the code over a span of 3 day. Employing a multifaceted approach that included static analysis, fuzz testing, formal verification, and manual code review, the Fuzzland team identified no issues across different severity levels and categories.

Scope

Project Name	EulerSwap
Repo	euler-swap
commit	449beb4cc188a3203eaab4a25cf134adec6ea15d
Language	Solidity
Scope	src/*.sol

Disclaimer

The audit does not ensure that it has identified every security issue in the smart contracts, and it should not be seen as a confirmation that there are no more vulnerabilities. The audit is not exhaustive, and we recommend further independent audits and setting up a public bug bounty program for enhanced security verification of the smart contracts. Additionally, this report should not be interpreted as personal financial advice or recommendations.

Auditing Process

- Static Analysis: We perform static analysis using our internal tools and Slither to identify potential vulnerabilities and coding issues.
- Fuzz Testing: We execute fuzz testing with our internal fuzzers to uncover potential bugs and logic flaws.
- Invariant Development: We convert the project into Foundry project and develop Foundry invariant tests for the project based on the code semantics and documentations.
- Invariant Testing: We run multiple fuzz testing tools, including Foundry and ItyFuzz, to identify violations of invariants we developed.
- Formal Verification: We develop individual tests for critical functions and leverage Halmos to prove the functions in question are not vulnerable.
- Manual Code Review: Our engineers manually review code to identify potential vulnerabilities not captured by previous methods.

Vulnerability Severity

We divide severity into three distinct levels: high, medium, low. This classification helps prioritize the issues identified during the audit based on their potential impact and urgency.

- **High Severity Issues** represent critical vulnerabilities or flaws that pose a significant risk to the system's security, functionality, or performance. These issues can lead to severe consequences such as fund loss, or major service disruptions if not addressed immediately. High severity issues typically require urgent attention and prompt remediation to mitigate potential damage and ensure the system's integrity and reliability.
- **Medium Severity Issues** are significant but not critical vulnerabilities or flaws that can impact the system's security, functionality, or performance. These issues might not pose an immediate threat but have the potential to cause considerable harm if left unaddressed over time. Addressing medium severity issues is important to maintain the overall health and efficiency of the system, though they do not require the same level of urgency as high severity issues.
- **Low Severity Issues** are minor vulnerabilities or flaws that have a limited impact on the system's security, functionality, or performance. These issues generally do not pose a significant risk and can be addressed in the regular maintenance cycle. While low severity issues are not critical, resolving them can help improve the system's overall quality and user experience by preventing the accumulation of minor problems over time.

Below is a summary of the vulnerabilities with their current status, highlighting the number of issues identified in each severity category and their resolution progress.

	Number	**Resolved**
High Severity Issues	0	0
Medium Severity Issues	0	0
Low Severity Issues	0	0
Info Severity Issues	2	0

Findings

[Info] Unrestricted concentration parameter range

There is following constraint in the WhitePaper:

> The curve coefficients in these equations, cx, cy ∈[0, 1]

The range of `concentrationX` and `concentrationY` is not checked in the constructor:

```
constructor(..., CurveParams memory curveParams) {
  // No check that concentrationX/Y is ≤1e18
  concentrationX = curveParams.concentrationX;
  concentrationY = curveParams.concentrationY;
}
```

If the parameter exceeds 1e18 (i.e., 1), the curve may fail or produce unexpected behavior.

Recommendation:

Add `require` statements to restrict the parameter range:

```
require(curveParams.concentrationX <= 1e18 && curveParams.concentrationY <=
1e18, "Invalid concentration");
```

Status: Acknowledged

[Info] Rounding issue allows users to steal 1

Since a fee will be charged when calling `swap` (`amount1In = amount1In * feeMultiplier / 1e18;`), when `feeMultiplier` < 1e18, the user can call the `swap` and transfer out one without transferring in any amount.

```
// src/EulerSwap.sol
function swap(uint256 amount0Out, uint256 amount1Out, address to, bytes
calldata data)
    external
    callThroughEVC
    nonReentrant
{
    //...
    if (amount0In > 0) {
        depositAssets(vault0, amount0In);
        amount0In = amount0In * feeMultiplier / 1e18;
    }

    uint256 amount1In = IERC20(asset1).balanceOf(address(this));
    if (amount1In > 0) {
        depositAssets(vault1, amount1In);
        amount1In = amount1In * feeMultiplier / 1e18;
    }
    //...
```

PoC:

```
// test/EulerSwap.t.sol
function test_basicSwap_onlyOut() public {
    uint256 amountOuto = 1;
    EulerSwap.swap(amountOuto, 0, address(this), "");
    console.log("assetTST.balanceOf(address(this))",
assetTST.balanceOf(address(this)));
}

// output
forge test --mt test_basicSwap_onlyOut -vvv
INT ✘ ▯ system Node
[⠊] Compiling...
[⠒] Compiling 100 files with Solc 0.8.27
[⠘] Solc 0.8.27 finished in 9.96s
Compiler run successful!

Ran 1 test for test/EulerSwap.t.sol:EulerSwapTest
[PASS] test_basicSwap_onlyOut() (gas: 288381)
Logs:
    assetTST.balanceOf(address(this)) 1
```

Recommendation:

Consider adding a minimum amount limit.

Status: Acknowledged

Additional Unit Tests

```
// test/AltDecimals.t.sol
function test_alt_decimals_6_18_in() public {
    createMaglev(50e6, 50e18, 0.01e18, 1e18, 1e6, 0.9e18, 0.9e18);//@note test 0.01e18
    skimAll(maglev, true);
    logState(address(maglev));

    uint256 amount = 1e6;
    uint256 q = periphery.quoteExactInput(address(maglev), address(assetTST), address(assetTST2), amount);
    assertApproxEqAbs(q, 1e18, 0.02e18);

    assetTST.mint(address(this), amount);
    assetTST.transfer(address(maglev), amount);

    {
        uint256 qPlus = q + 1;
        vm.expectRevert();
        maglev.swap(0, qPlus, address(this), "");
    }

    maglev.swap(0, q, address(this), "");
    logState(address(maglev));
}
```

```solidity
// test/EulerSwap.t.sol
function test_basicSwap_InAndOut() public monotonicHolderNAV {
    // logState(address(maglev));
    uint256 amountOut0 = 10e18;
    uint256 amountIn0 = periphery.quoteExactOutput(address(maglev), address(assetTST), address(assetTST2), amountOut0);
    console.log("amountIn0",amountIn0);

    uint256 amountOut1 = 10e18;
    uint256 amountIn1 = periphery.quoteExactOutput(address(maglev), address(assetTST), address(assetTST2), amountOut1);
    console.log("amountIn1",amountIn1);

    assetTST.mint(address(this), amountIn0);
    assetTST.mint(address(this), amountIn1);

    assetTST.transfer(address(maglev), amountIn0);
    assetTST.transfer(address(maglev), amountIn1);

    maglev.swap(amountOut0, amountOut1, address(this), "");

    console.log("amountOut0",amountOut0 + amountOut1);
console.log("assetTST.balanceOf(address(this))",assetTST.balanceOf(address(this

console.log("assetTST2.balanceOf(address(this))",assetTST2.balanceOf(address(th
    // logState(address(maglev));

    uint256 amountOut10 = 1e18;
    uint256 amountIn10 = periphery.quoteExactOutput(address(maglev),
address(assetTST), address(assetTST2), amountOut10);
    console.log("amountIn10",amountIn10);
}

function test_basicSwap_onlyOut() public {
    maglev.swap(1, 0, address(this), "");

console.log("assetTST.balanceOf(address(this))",assetTST.balanceOf(address(this
}

function test_priceAfterFeeGain() public {
    uint256 amountIn = 1e18;
    uint256 firstAmountOut;
    uint256 lastAmountOut;

    for (uint256 i = 0; i < 1002; i++) {
        assetTST.mint(address(this), amountIn);

        uint256 currentOut = periphery.quoteExactInput(
            address(maglev),
            address(assetTST),
            address(assetTST2),
            amountIn
        );

        if (i == 0) {
            firstAmountOut = currentOut;
            console.log("First swap amountOut:", currentOut);
        }
        if (i == 1001) {
            lastAmountOut = currentOut;
            console.log("102nd swap amountOut:", currentOut);
        }

        assetTST.transfer(address(maglev), amountIn);
        maglev.swap(0, currentOut, address(this), "");
    }
}

function test_invariant_bidirectionalSwap(uint256 amount) public {
    console.log("begin test_invariant_bidirectionalSwap");
    int256 origNAV = getHolderNAV();

    amount = bound(amount, 0.1e18, 10e18);
    performSwap(amount, true);

    performSwap(amount, false);


    assertGe(getHolderNAV(), origNAV);
}


function performSwap(uint256 amount, bool dir) internal {
    TestERC20 tIn = dir ? assetTST : assetTST2;
    TestERC20 tOut = dir ? assetTST2 : assetTST;

    tIn.mint(address(this), amount);
    uint256 quoted = periphery.quoteExactInput(address(maglev), address(tIn),
address(tOut), amount);

    tIn.transfer(address(maglev), amount);
    if (dir) {
        maglev.swap(0, quoted, address(this), "");
    } else {
        maglev.swap(quoted, 0, address(this), "");
    }
}
```