Smart Contract
Security Audit Report

Prepared for Euler Labs Limited

Prepared by Supremacy

June 13, 2025

Contents

1 Introduction 3
1.1 About Client 4
1.2 Audit Scope 4
1.3 Changelogs 4
1.4 About Us 4
1.5 Terminology 5
2 Findings 6
2.1 Informational 7
3 Disclaimer 9

1 Introduction

Given the opportunity to review the design document and related codebase of the Euler Finance's IRMLinearKinky, we outline in the report our systematic approach to evaluate potential security issues in the smart contract(s) implementation, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About Client

Euler is a flexible platform for decentralized lending and borrowing, designed to adapt and grow with the evolving world of DeFi.

Item	Description
Client	Euler Labs Limited
Project	IRMLinearKinky
Type	Smart Contract
Languages	Solidity
Platform	EVM-compatible

1.2 Audit Scope

In the following, we show the Git repository of reviewed file and the commit hash used in this security audit:

- Repository: https://github.com/euler-xyz/evk-periphery/blob/a647325c329cb1ed398fab7cac9b8e31108c282e/src/IRM/IRMLinearKinky.sol
- Commit Hash: a647325c329cb1ed398fab7cac9b8e31108c282e

Below are the files in scope for this security audit and their corresponding MD5 hashes.

Filename	MD5
`./src/IRM/IRMLinearKinky.sol`	`1B56EC55623D2C90B8171330BC09A8D3`

1.3 Changelogs

Version	Date	Description
0.1	June 12, 2025	Initial Draft
1.0	June 13, 2025	Final Release

1.4 About Us

Supremacy is a leading blockchain security firm, composed of industry hackers and academic researchers, provide top-notch security solutions through our technology and innovative research.

We are reachable at X (https://x.com/SupremacyHQ), or Email (contact@supremacy.email).

1.5 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- Likelihood represents the likelihood of a finding to be triggered or exploited in practice
- Impact specifies the technical and business-related consequences of a finding
- Severity is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Severity

Impact	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

Likelihood

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

2 Findings

The table below summarizes the findings of the audit, including status and severity details.

ID	Severity	Description	Status
1	Informational	Potential loss of precision	Acknowledged
2	Informational	Insufficient constructor parameter validation	Acknowledged
3	Informational	Lack of comment	Fixed

2.1 Informational

1. Potential loss of precision [Informational]

`Status: Acknowledged`

Description

The `IRMLinearKinky` contract's `computeInterestRateInternal()` function implements a non-linear interest rate formula when utilization exceeds the `kink`:

```
83      ir += slopeUtilizationOverKink * kinkRemaining * (1 + shape) /
    utilizationRemaining - slopeUtilizationOverKink * shape;
```

IRMLinearKinky.sol

This formula, aligned with the provided math model, involves integer division by `utilizationRemaining` (i.e., `type(uint32).max - utilization`). When utilization approaches `type(uint32).max` (corresponding to u⟶1), `utilizationRemaining` becomes very small, potentially leading to significant precision loss due to integer division truncating results. For example, if `utilizationRemaining` is in a borderline case, the division may produce a disproportionately large or truncated result, causing the calculated interest rate (ir) to deviate from the expected continuous curve in the math model. This could lead to inaccurate interest rates.

Recommendation

Revise the code logic accordingly.

Feedback

We acknowledge. The precision loss in this case is accepted.

2. Insufficient constructor parameter validation [Informational]

`Status: Acknowledged`

Description

The `IRMLinearKinky` contract's `constructor` accepts critical parameters (`baseRate_`, `slope_`, `shape_`, `kink_`, `cutoff_`) that define the behavior of the interest rate model. However, it lacks sufficient validation for these parameters, which could lead to incorrect or unsafe configurations of the interest rate curve. Specifically, these variables are not constrained as required by the comments.

Recommendation

Revise the code logic accordingly.

Feedback

We acknowledge. Our IRM factory (not in scope of this audit) will be verifying the input parameters.

3. Lack of comment [Informational]

`Status: Fixed`

Description

Throughout the codebase there are numerous functions missing or lacking documentation. This hinders reviewers' understanding of the code's intention, which is fundamental to correctly assess not only security, but also correctness. Additionally, comments improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios under which they can fail, the roles allowed to call them, the values returned and the events emitted.

Recommendation

Consider thoroughly documenting all functions (and their parameters) that are part of the smart contracts' public interfaces. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing comments, consider following the Ethereum Natural Specification Format (NatSpec).

Feedback

Fixed in `ff2a4e9`.

3 Disclaimer

This security audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. This security audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues, also cannot make guarantees about any additional code added to the assessed project after the audit version. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contract(s). Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.