Kann Audits

Mystic Finance Security Review

13/02/2026

Contents

1 About Kann Audits

Kann Audits is a top-tier Web3 security audit company, trusted by leading projects and providing comprehensive audits and expert guidance to secure the most critical blockchain protocols.

Check out our previous work or reach out on Twitter @KannAudits.

2 Disclaimer

A security audit can never guarantee the complete absence of vulnerabilities. Audits are a time, resource, and expertise-bound effort in which we aim to identify as many issues as possible.

Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users
- **Medium** - leads to a moderate material loss of assets in the protocol or moderately harms a group of users
- **Low** - leads to a minor material loss of assets in the protocol or harms a small group of users

Likelihood

- **High** - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost

- **Medium** - only a conditionally incentivized attack vector, but still relatively likely
- **Low** - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive

3 About Mystic Finance

Mystic Finance is a DeFi lending and borrowing protocol that allows users to supply assets, borrow against collateral, and leverage positions, with support for liquid staking and tokenized real-world assets.

4 Executive Summary

Protocol Summary

Project Name	Mystic Finance
Protocol Type	Vaults / 0x Deployment
Timeline	12/02/2026 - 13/02/2026
Review commit hash:	
`139a4505e23028e7d6875fdbca294487eae4fdd4`	
Fixes review commit hash:	

Scope

`deployment/scripts/deploy-optimized.js`

5 Findings

Findings count

Summary of findings

Severity	Amount
Critical	0
High	0
Medium	0
Low	1
Informational	0
Total findings	**1**

ID	Title	Severity
[L-01]	Incorrect Constructor Arguments Used in ZeroExOptimized Verification	Low

5.1 Low Severity

5.1.1 [L-01] Incorrect Constructor Arguments Used in ZeroExOptimized Verification

Description: In the deployment script, the ‘ZeroExOptimized‘ contract is deployed using the bootstrapper address obtained from ‘fullMigration.getBootstrapper()‘.

```
    const zeroExOptimized = await deployContract(ZeroExOptimizedFactory, [await
        fullMigration.getBootstrapper()]);
    console.log("ZeroExOptimized deployed to:", zeroExOptimized.address);
```

However, during the verification step, the script incorrectly uses the ‘FullMigration‘ contract address as the constructor argument.

```
 await verifyContract(hre, deploymentResults.contracts.ZeroExOptimized, [
     deploymentResults.contracts.FullMigration]);
```

This mismatch causes verification failures because the constructor arguments provided to the verification function do not match those used during deployment.

Root cause: The verification logic does not account for the actual constructor parameter used when deploying ‘ZeroExOptimized‘. Instead of using the bootstrapper address returned by ‘fullMigration.getBootstrapper()‘, it mistakenly uses ‘fullMigration.address‘, resulting in incorrect verification data.

Recommendation: Pass bootstrapper as the constructor argument during verification.

```
 const bootstrapper = await fullMigration.getBootstrapper();
deploymentResults.bootstrapper = bootstrapper;
await verifyContract(hre, deploymentResults.contracts.ZeroExOptimized, [
    deploymentResults.bootstrapper])
```

Resolution: Fixed