Auditors
Craig - Lead Security Researcher
0xf4ld3 - Lead Security Researcher

October 22, 2025

Contents

1 Introduction

1.1 About ChainCheck Audits

ChainCheck Audits is a blockchain security marketplace that connects leading security researchers and auditing specialists with projects seeking robust security assurance

1.2 Disclaimer

ChainCheck Audits provides an in-depth assessment of a project's security posture based on the code available at a specific point in time. While every effort is made to identify potential security vulnerabilities and implementation risks, the review cannot guarantee that all issues will be uncovered or that the codebase will remain immune to every possible attack vector. This evaluation applies only to the exact code version and commit that were reviewed. Any subsequent modifications may introduce new vulnerabilities not covered in this report. Therefore, projects are strongly advised to request a follow-up review after making any changes to the code. Please note that this assessment should not be considered a substitute for continuous security practices such as penetration testing, automated vulnerability scanning, and periodic internal or external audits.

1.3 Risk Assessment

Severity	Description
Critical	Must be fixed immediately (especially if the system is already deployed).
High	Can result in significant asset loss (>10%) or major impact on most users.
Medium	May cause limited losses (<10%) or affect only a few users but remains unacceptable.
Low	Minor issues causing limited or temporary disruption; includes griefing or inefficiency risks.
Gas Optimization	Recommendations for improving gas efficiency and reducing operational costs.
Informational	Non-critical insights, best practices, or suggestions for improving code readability and maintainability.

1.3.1 Severity Classification

Each issue identified during the review is classified according to its potential impact and likelihood of exploitation.
- Critical vulnerabilities pose an immediate and severe threat; these must be resolved urgently.
- High severity issues are easily exploitable or highly incentivized and should be addressed as soon as possible.
- Medium severity issues are plausible under specific conditions or with moderate incentive and should be remediated promptly.
- Low severity findings require unlikely conditions to exploit or pose minimal incentive but should still be fixed for completeness.
- Gas Optimization and Informational findings do not directly impact security but represent meaningful improvements to performance, efficiency, and code quality.

2. Korvbox Smart Contracts Security Review

Audit Overview :

Project: Korvbox Smart Contracts
Chain: Base Mainnet
Audit Duration: October 20 - 21, 2025
Auditor: ChainCheck Audits
Total Issues: 5 (1 Low, 4 Informational)
Contract Version: Solidity ^0.8.20

Summary
LendManager is a Morpho Integration Contract that facilitates the deposit and withdrawal of assets into the Morpho protocol, as well as the claiming of accrued rewards. It acts as an intermediary layer managing user liquidity and reward interactions with the underlying lending markets.

3. Detailed Findings

3.1 Low Risk Findings

3.1.1 [L-01] Fees Lost When feeRecipient Not Set

Severity: Low

Description:

The contract allows deposits and fee-charging operations before the feeRecipient address is initialized. This introduces a potential loss of protocol revenue because any fees calculated and transferred while feeRecipient is set to address(0) are irrecoverably sent to the zero address, effectively burning funds.

Root Cause:

- The initialize() function does not set a valid feeRecipient.
- The internal _chargeFee() function does not validate that feeRecipient is non-zero before executing transfers.

Impact:
While user funds remain unaffected, the protocol permanently loses its intended revenue from these transactions. This issue becomes more critical if multiple operations or epochs are executed before initialization is completed.

Recommendation:

- Require feeRecipient != address(0) in _chargeFee().
  OR
- Initialize feeRecipient properly during deployment .

Example Fix:

```
require(feeRecipient != address(0), "Invalid fee recipient");
```

3.2 Informational Findings

3.2.1 [I-01] Missing Zero Address Validation in setFeeRecipient

Severity: Informational

Description:

The setFeeRecipient() function allows assignment of a zero address as the new recipient. If the admin mistakenly sets feeRecipient to address(0), subsequent fee transfers would burn funds, leading to lost protocol revenue.

Impact:

No direct vulnerability exists if used correctly, but misconfiguration can result in lost protocol fees.

Root Cause:

Lack of a require(_newFeeRecipient != address(0)) validation in the setter function.

Recommendation:

Add input validation in setFeeRecipient() to prevent assignment of the zero address.

Example Fix:

```
function setFeeRecipient(address _newFeeRecipient) external onlyOwner {
   require(_newFeeRecipient != address(0), "Invalid recipient");
   feeRecipient = _newFeeRecipient;
}
```

3.2.2 [I-02] Unreadable Error Messages Using Dynamic Hashes

Severity: Informational

Description:

Several revert statements display hashed keccak256 values or cryptic identifiers instead of clear, human-readable revert reasons. This complicates debugging during testing or integration, especially when external contracts or frontends interface with the system.

Root Cause:

Error messages implemented as keccak256 hashes (e.g., keccak256("VAULT_ZERO_ADDRESS")) rather than descriptive revert strings or custom errors.

Impact:

- Reduced transparency in revert causes.
- Increased debugging time for developers and auditors.

Recommendation:

Option 1: Use descriptive error strings or Solidity 0.8.x custom error types for improved clarity.

Example Fix:

```
error NoZeroAddress(string field);
require(_vault != address(0) && _asset != address(0), Errors.NoZeroAddress("Vault or Asset"));
```

Option 2: Separate error messages

Example Fix:

```
require(_vault != address(0), Errors.VaultIsZeroAddress());
require(_asset != address(0), Errors.AssetIsZeroAddress());
```

3.2.3 [I-03] getCurrentShareValue() Reverts Instead of Returning Zero

Severity: Informational

Description:

The getCurrentShareValue() function reverts when a user has no shares (userShares == 0). This behavior can break frontends or integrators that query the value of users who haven't interacted with the contract yet.

Root Cause:

A strict require(userShares > 0) statement forces a revert rather than handling the zero-share case gracefully.

Recommendation:

Return a default value (e.g., 0) when a user has no shares, or clearly document this revert behavior in the function's NatSpec comments.

Example Fix:

```
if (userShares == 0) return 0;
```

3.2.4 [I-04] Missing Storage Gap in LendManager.sol Contract

Severity: Informational

Description:

The LendManager contract is designed to be upgradeable (likely using UUPS or Transparent proxy pattern). However, it lacks a reserved storage gap to accommodate future variable additions without causing storage layout collisions.

Root Cause:

Absence of a reserved storage gap at the end of the contract.

Impact:

Future upgrades could unintentionally overwrite existing state variables, leading to corrupted state, lost balances, or broken functionality.

Recommendation:

Include a storage gap as per OpenZeppelin's upgradeable contract recommendations.

Example Fix:

```
uint256[50] private __gap;
```

This audit report was generated for the LendManager Morpho Integration Contract. All findings should be reviewed and mitigated prior to mainnet deployment.