Smart Contract Security Assessment

Final Report

Kame.ag

28 July 2025

Table of contents

1. Overview

This report has been prepared for Kame.ag. Sotatek provides a user-centered examination of smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

A comprehensive examination has been performed, utilizing Cross Referencing, Static Analysis, In-House Security Tools, and line-by-line Manual Review. The auditing process pays special attention to the following considerations:

- Ensuring contract logic meets the specifications and intentions of the

client without exposing the user's funds to risk.

- Testing the smart contracts against both common and uncommon attack vectors.
- Inspecting liquidity and holders' statistics to inform the status to both users and clients when applicable.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Verifying contract functions that allow trusted and/or untrusted actors to mint, lock, pause, and transfer assets.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Summary

Project Name	Kame Ag
Type	Defi
Platform	SEI
Language	Solidity
Auditors	Sotatek
Timeline	Jul 20, 2025 – Jul 27, 2025
Description	Kame is the premier native DEX aggregator built specifically for the Sei Network, designed to find users the most efficient on-chain swap routes across major liquidity venues. Kame emphasizes ultra-low friction and high performance, allowing traders to quickly access deep liquidity in under a second across the Sei ecosystem.

1.1 Audit Summary

Delivery Date	Jul 27, 2025
Audit Methodology	Static Analysis, Manual Review

1.2 Vulnerability Summary

Vulnerability	Total	Pending	Resolved	Acknowledged
Critical	0	0	0	0
Major	0	0	0	0
Medium	2	0	0	2
Informational	2	0	0	2

Classification Of Issues

Severity	Description
Critical	Exploits, vulnerabilities, or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Major	Bugs or issues with that may be subject to exploitation, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible
Medium	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax, or style best practices. Generally, pose a negligible level of risk, if any.

1.3 Audit Scope

Delivery Date	July 27, 2025
Audit Methodology	Static Analysis, Manual Review

The following files were made available during the review:

File	Interfaces
pyxis-sc/blob/main/src/AggregationExecutor.sol	1
pyxis-sc/blob/main/src/AggregationRouter.sol	1
Totals	2

2. Finding

1.1 KAME-01 | Mixed solidity versions

Category	Severity	Location	Status
Volatile Code	Medium	Multiple location	Acknowledged

Description

The smart contract codebase uses multiple Solidity compiler versions (0.8.13, 0.8.20, and 0.8.21) across different files. This inconsistency can lead to several problems:

1. **Compatibility Issues**: Different versions may have incompatible features or behaviors, potentially causing unexpected interactions between contracts.
2. **Security Risks**: Older versions might lack important security fixes or optimizations present in newer versions.
3. **Maintenance Challenges**: Managing multiple compiler versions complicates the development and deployment processes.
4. **Audit Complexity**: Inconsistent versions make it harder to conduct a thorough security audit, as each version needs to be considered separately.

Recommendation

Standardize the Solidity compiler version across all smart contracts. We recommend using the most recent stable version (0.8.20 in this case) for all contracts, unless there's a specific reason to use an older version. If upgrading all contracts to the latest version is not immediately feasible, consider the following:

1. Document the reasons for using different versions.
2. Plan a gradual migration to a single, up-to-date version.
3. Ensure thorough testing of inter-contract interactions, especially between contracts using different versions.

1.2 KAME-02 | Missing events

Category	Severity	Location	Status
Volatile Code	Information	src/AggregationExecutor.sol	Acknowledged

Description

rescueFunds functions that change critical contract parameters/addresses/state should emit events and consider adding timelocks so that users and other privileged roles can detect upcoming changes (by offchain monitoring of events) and have the time to react to them.

Recommendation

Add events to all possible flows (some flows emit events in callers) and consider adding timelocks to such onlyAdmin functions.

1.3 KAME-03 | Missing Natspects

Category	Severity	Location	Status
Volatile Code	Information	src/AggregationExecutor.sol	Acknowledged

Description

The rescueFunds function lacks NatSpec comments, which are recommended for all public or external functions. NatSpec improves:

Recommendation

Add NatSpec comments to clarify intent and parameters

1.4 KAME-04 | Centralization

Category	Severity	Location	Status
Volatile Code	Medium	https://github.com/kitelabs-io/pyxis-sc/blob/5c94dd2358e3329b7eeb2c190c8cfb60a4773159/src/AggregationExecutor.sol#L72C14-L72C25	Acknowledged

Description

The rescueFunds function allows the contract owner to withdraw any ERC20 token from the contract to an arbitrary recipient. While this is a common utility function used to recover stuck tokens, it introduces a centralized control risk.

The function is gated by the onlyOwner modifier, meaning a single private key (EOA) could unilaterally withdraw funds.

Recommendation

To mitigate this centralization risk and align with decentralized security best practices:

- The owner should be a multisig wallet (e.g., Gnosis Safe) to reduce single-point-of-failure risks.

- Alternatively, integrate a timelock contract to delay sensitive operations, allowing time for community response.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e., incorrect usage of private or deleted.