RD AUDITORS

SeaSwap Smart Contract, Code Review and Security Analysis Report

Customer: SeaSwap
Prepared on: 29th September 2025
Platform: Kaspa
Language: Solidity

rdauditors.com

Table of Contents

Disclaimer

This document may contain confidential information about its systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the customer or it can be disclosed publicly after all vulnerabilities are fixed - upon the decision of the customer.

Document

Name	Smart Contract Code Review and Security Analysis Report of SeaSwap
Platform	Kaspa/Solidity
File 1	Aggregator.sol
MD5 hash	0c7e49ead1f9ff97bc04fe89abde3a31
SHA256 hash	efb57c2826ea488c0b29edc816065bb17bcd8643ddbf36c3ff0eb84670a6b3ac
File 2	Auth.sol
MD5 hash	2d5d336d1e32d0f462b536dc94c7b088
SHA256 hash	bf64584bae1288695ee1f4995e4143e3648f5b3feb5aea954ba51b851d2a272d
File 3	Events.sol
MD5 hash	1f52ce4ccff53d7e05899bef98002f91
SHA256 hash	e70bb60cea935f052c88c707667a1e1011855125ff58b22104bf326e1e61ed6c
File 4	Params.sol

MD5 hash	4766d376c6accba65cb159efec54f4b1
SHA256 hash	d0c31880338b3b2b2a6097ab0ff9f04e317a51eb1fc3f18886d5579b42bf2b77
Date	29/09/2025

Introduction

RD Auditors (Consultant) were contracted by SeaSwap (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report represents the findings of the security assessment of the customer`s smart contract and its code review conducted between 22nd - 29th September 2025.

This contract consists of 4 files.

Project Scope

The scope of the project is a smart contract. We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but is not limited to):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

According to the assessment, the customer's solidity smart contract is now **Well-Secured.**

Automated checks are with smartDec, Mythril, Slither and remix IDE. All issues were performed by our team, which included the analysis of code functionality, the manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the AS-IS section and all issues found are located in the audit overview section.

We found the following;

Total Issues	0
Critical	0
High	0
Medium	0
Low	0
Very Low	0

Code Quality

The libraries within this smart contract are part of a logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned to a specific address and its properties/methods can be reused many times by other contracts.

Documentation

The hash of that file is mentioned in the table. As mentioned above, It's recommended to write comments in the smart contract code, so anyone can quickly understand the programming flow as well as complex code logic.

Comments are very helpful in understanding the overall architecture of the protocol. It also provides a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure. Those were based on well known industry standard open source projects and even core code blocks that are written well and systematically.

AS-IS Overview

Aggregator.sol

File And Function Level Report

File 1: Aggregator.sol

Contract: Aggregator

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	setPaused	External	Passed	All Passed	No Issue	Passed
2	setAuthSigner	External	Passed	All Passed	No Issue	Passed
3	setPositiveSlippageBPS	External	Passed	All Passed	No Issue	Passed
4	sweep	External	Passed	All Passed	No Issue	Passed
5	transferOwnership	External	Passed	All Passed	No Issue	Passed
6	setZealousDiscountManager	External	Passed	All Passed	No Issue	Passed
7	setFeeReceiver	External	Passed	All Passed	No Issue	Passed
8	setFactoryAllowed	External	Passed	All Passed	No Issue	Passed
9	setRouterAllowed	External	Passed	All Passed	No Issue	Passed
10	swap	External	Passed	All Passed	No Issue	Passed

11	_swapWithAuth	internal	Passed	All Passed	No Issue	Passed
12	_chargePlatformAndReferralFees	internal	Passed	All Passed	No Issue	Passed
13	_takeFee	internal	Passed	All Passed	No Issue	Passed
14	_executeRoute	internal	Passed	All Passed	No Issue	Passed
15	_executeRouteV1	internal	Passed	All Passed	No Issue	Passed
16	_normalizeIn	internal	Passed	All Passed	No Issue	Passed
17	_acquireInput	internal	Passed	All Passed	No Issue	Passed
18	_consumePermit	internal	Passed	All Passed	No Issue	Passed
19	_handleV3SwapCallback	internal	Passed	All Passed	No Issue	Passed
20	uniswapV3SwapCallback	External	Passed	All Passed	No Issue	Passed
21	kaspaV3SwapCallback	External	Passed	All Passed	No Issue	Passed
22	_capturePositiveSlippage	internal	Passed	All Passed	No Issue	Passed
23	_settleAndEmit	internal	Passed	All Passed	No Issue	Passed
24	_prevalidateV1	internal	Passed	All Passed	No Issue	Passed

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to lost tokens etc.
High	High level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial functions.
Medium	Medium level vulnerabilities are important to fix; however, they cannot lead to lost tokens.
Low	Low level vulnerabilities are most related to outdated, unused etc. These code snippets cannot have a significant impact on execution.
Lowest Code Style/ Best Practice	Lowest level vulnerabilities, code style violations and information statements cannot affect smart contract execution and can be ignored.

Audit Findings

Critical:

No critical severity vulnerabilities were found.

High:

No high severity vulnerabilities were found.

Medium:

No medium severity vulnerabilities were found.

Low:

No low severity vulnerabilities were found.

Very Low:

No very low severity vulnerabilities were found.

Conclusion

We were given a contract file and have used all possible tests based on the given object. So it is ready to go for production. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

The security state of the reviewed contract is “**Well-Secured**”.

Note For Contract Users

Technical auditing does not guarantee the project's ethical side.

Please do your due diligence before investing. Our audit report is never an investment advice.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyse the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

RD Auditors

The smart contracts given for audit have been analysed in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Because the total number of test cases are unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

Email: info@rdauditors.com

Website: www.rdauditors.com