SMART CONTRACT AUDIT

zokyo

December 6th 2022 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

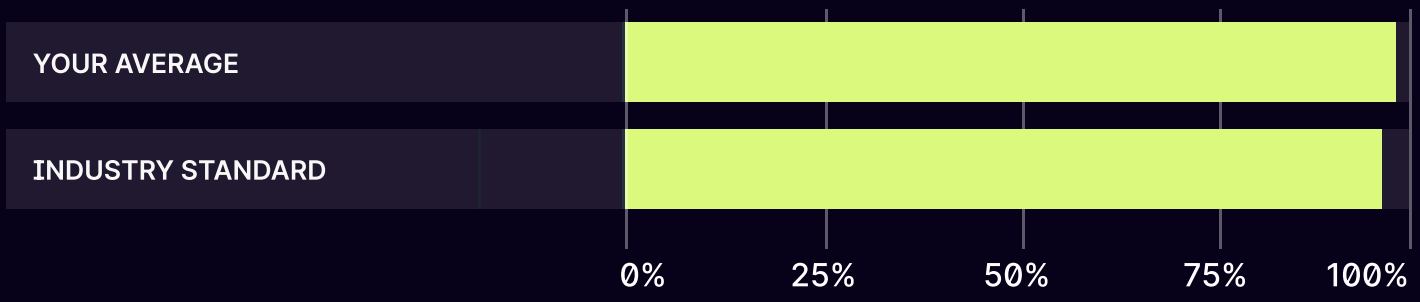
TECHNICAL SUMMARY

This document outlines the overall security of the Symbiosis smart contracts evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Symbiosis smart contract codebase for quality, security, and correctness.

Contract Status

Testable Code



The testable code verified by Zokyo Security is sufficient to cover the industry standard.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Symbiosis team put in place a bug bounty program to encourage further active analysis of the smart contract.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Symbiosis repository:
https://github.com/symbiosis-finance/octopool-audit

Initial commit: 31401e5b3f88205ed1285cb35cb93c1b9d2fdbb9
Final commit: 6aa057f01de274213f600e8c55778f0f1b40f2ee

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- AggregateAccount.sol
- DSMath.sol
- SafeCast.sol
- SignedSafeMath.sol
- Pool.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most resent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Symbiosis smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. Part of this work includes writing a test suite using the Anchor testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.
02	Cross-comparison with other, similar smart contracts by industry leaders.
03	Testing contract logic against common and uncommon attack vectors.
04	Thorough manual review of the codebase line by line.

Executive Summary

During the audit, the Zokyo Security team has reviewed the whole set of contracts and libraries provided by the Symbiosis team. The protocol cosists of the main smart contract, Pool.sol, and a set of helper libraries. The protocol represents a pool for multiple assets that allows users to deposit and swap assets as well as earn fees for each swap.

The goal of the audit was to verify that the contracts are implemented corresponding to the best security and Solidity practises. This includes analyzing the contracts against the auditors' checklist of vulnerabilities and ensuring that the best Solidity practises in terms of gas spendings and optimizations are applied. Part of the audit was to verify the correctness of the implemented bussiness logic in the contracts. In order to fulfill this task, the Zokyo Security team requested the documentation of the protocol. The contracts were carefully checked to correspond the provided documentation. One of the features that is worth mentioning is that the pool's supply of assets might be decreased during swaps. Due to this, at some point, the pool might not have enough supply to pay its liability to users. However, as explained in the documentation, such functinality is a part of the bussiness logic and users should wait until there is enough assets in the pool to pay full liability. We recommende the Symbiosys team to share this documentation with the users of the protocol as it contains a detailed overview of the protocol. It should also be mentioned that Pool.sol is an upgradable contract. The correct work of the contracts depends on the initiate parameters of the pools, such as amplification factor, LP fee, and fee ratio, which are set during the deployment and can be changed at any time by the owner of the contract. We recommend the Symbiosis team to set these parameters with extra caution.

There were some high-severity issues found in the contracts during the audit. The issues were connected to the block of the withdrawal function, sending liquidity to the wrong address, unsafe type casts. Other issues were connected to the bussiness logic verification, the absence of parameters validation, code style, and gas optimization suggestions. All of the issues were successfully fixed by the Symbiosis team. Zokyo Security has also prepared a set of unit tests and additional scenarios. All the tests listed in the report were written by the team of auditors.

The overall security of the protocol is high enough. The contracts correspond to the documentation. The code is well-written, has good readability, and contains a detailed natspec documentation.

Protocol Overview

Pool.sol

OctoPool is a protocol that allows users to deposit, swap, and withdraw assets to/from Pool Protocol. It has 2 main contracts and 3 contracts responsible for math libraries

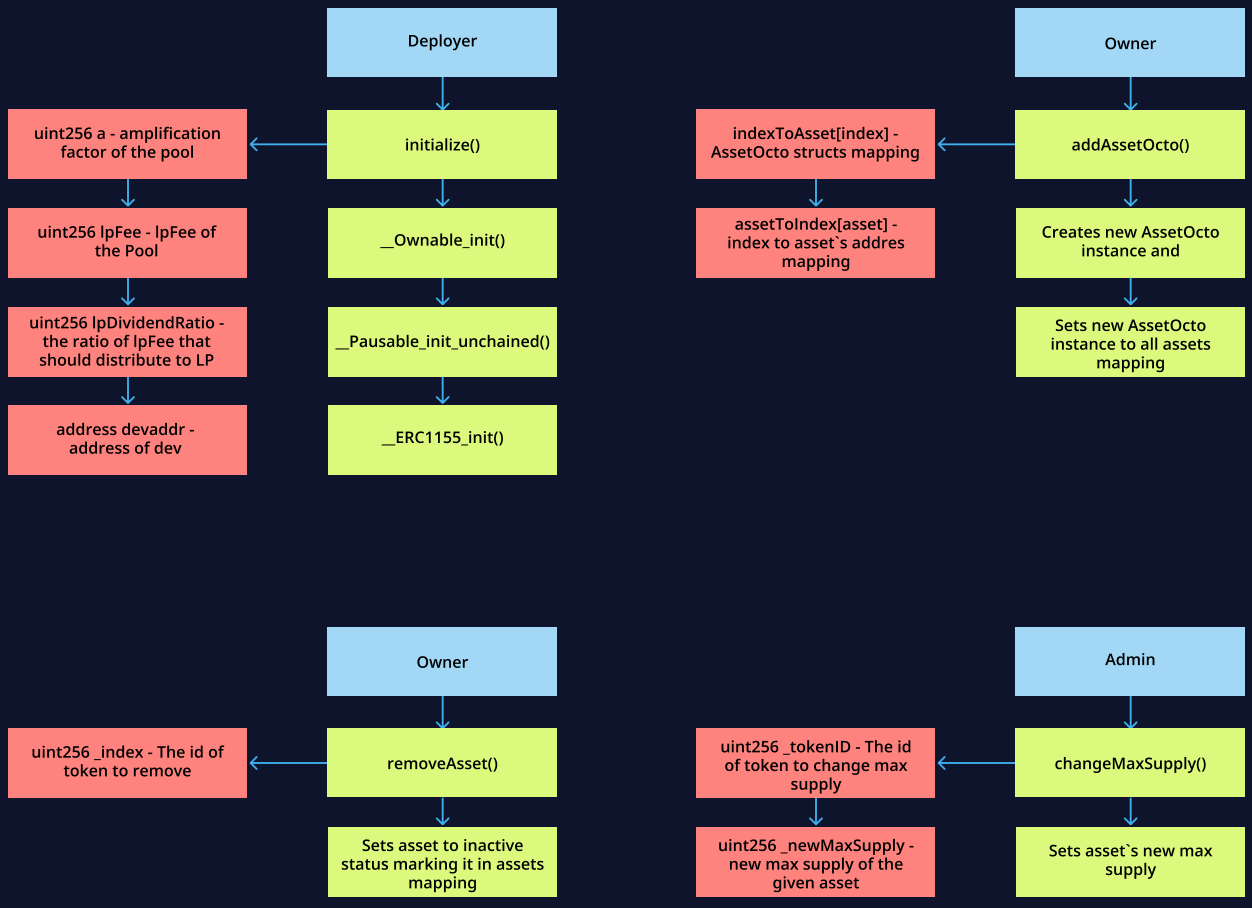
In Octopool users can:
- deposit assets to the Pool (Pool.sol)
- swap assets in the Pool (Pool.sol)
- withdraw assets from the Pool (Pool.sol)

Pool.sol is a contract that manages deposits, withdrawals, and swaps. It holds a mapping of assets and parameters.

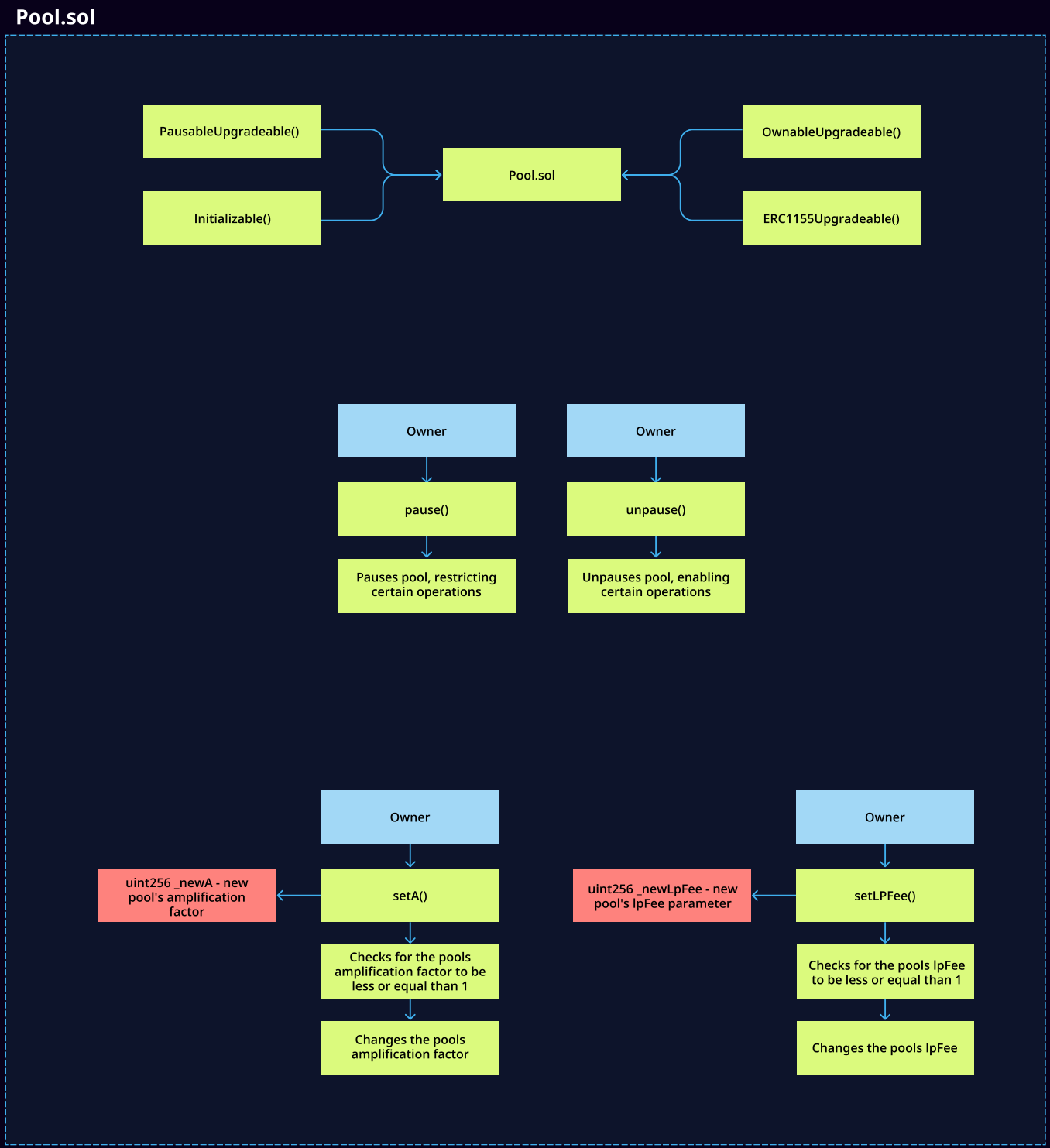
AggregateAccount.sol is a contract that represents groups of assets

SignedSafeMath.sol is a contract responsible for mathematical operations

SafeCast.sol is a contract responsible for converting a signed int256 into an unsigned uint256 and the other way.

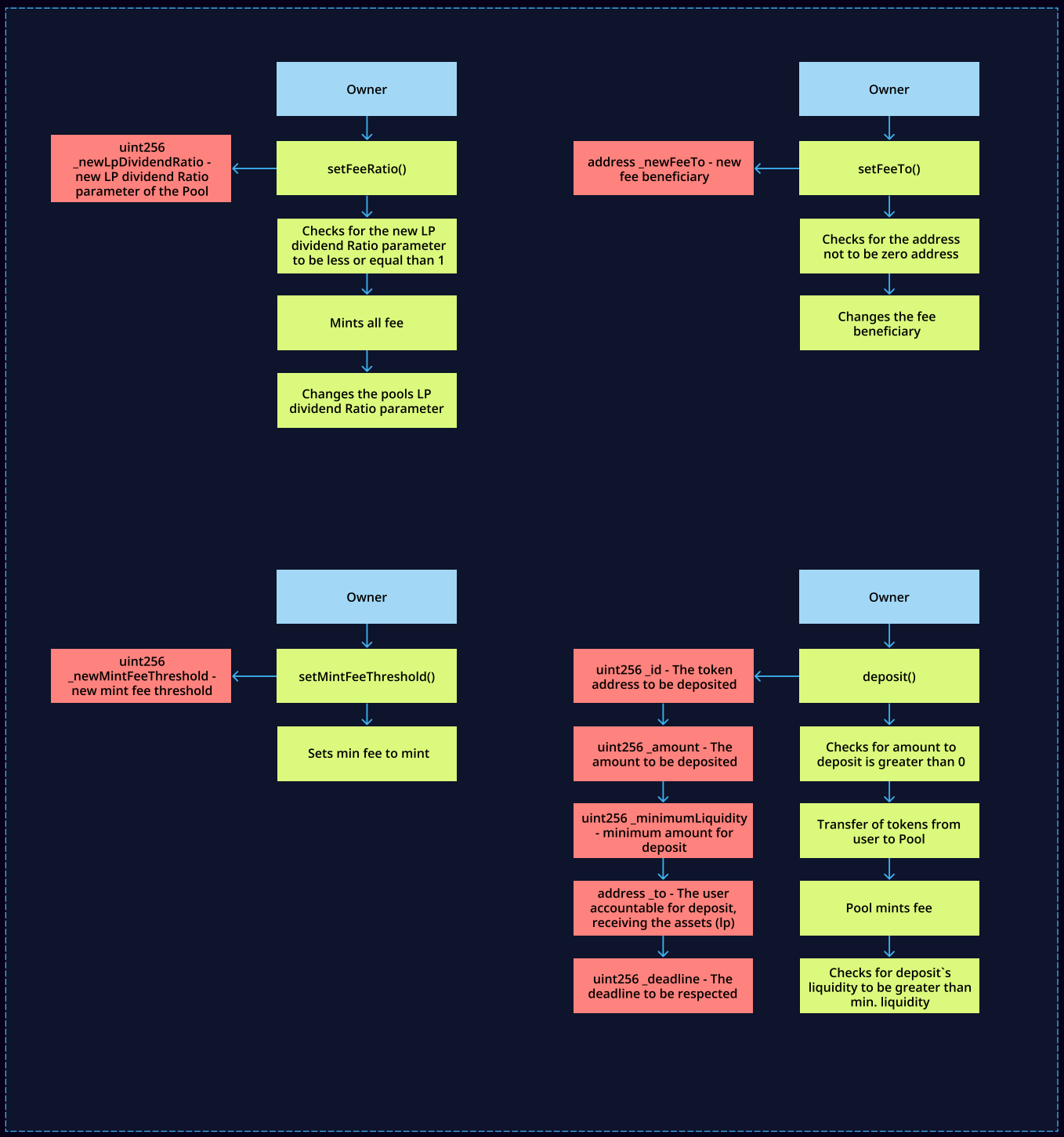


Protocol Overview



Protocol Overview

Pool.sol

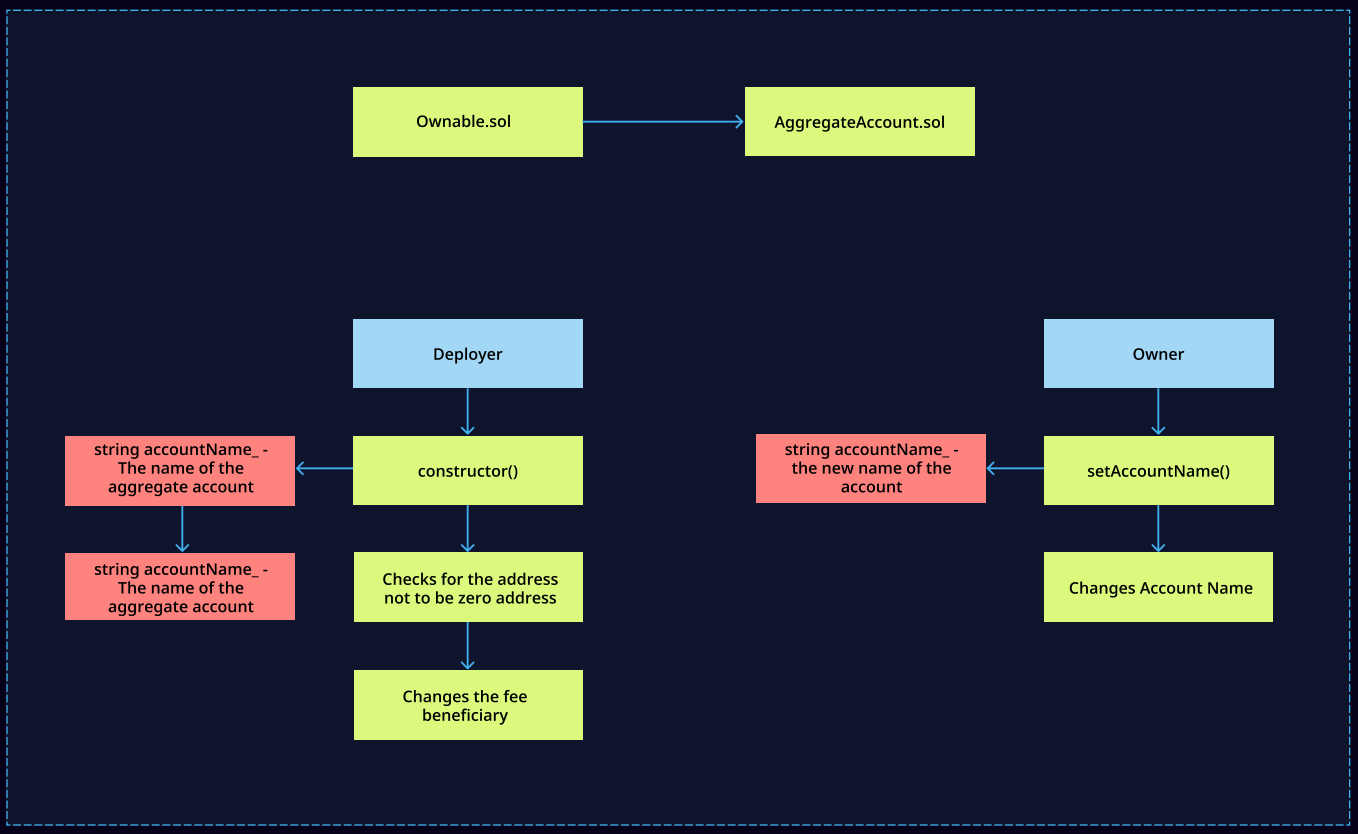


Protocol Overview

Pool.sol

Protocol Overview

AggregateAccount.sol



STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, sections are arranged from the most to the least critical one. Issues are tagged as “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

HIGH-1 | RESOLVED

Users can deposit, withdraw, and swap assets that were removed by the owner.

Pool.sol, deposit(), withdraw(), swap()
In AssetOcto struct, there is a field for marking whether the asset is active. After initializing this parameter in addAssetOcto(), it is might be changed by calling the owner`s removeAsset() method, which is set to false. Yet, in the deposit(), withdraw(), and swap() methods, there are no checks for the status of the asset, and users can perform actions even after “removing” the asset by the owner.

Recommendation:
Add checks in deposit(), withdraw(), and swap() for validating that the asset is active.

Post-audit.
A validation for not allowing actions with removed assets was added

HIGH-2 | RESOLVED

Minting LP tokens and their withdrawal from the Pool to the wrong address.

Pool.sol, deposit(), withdraw()
*deposit(): When calling deposit method parameter `_to` is given by input parameters as a receiver of liquidity but LP tokens are minted to msg.sender instead of a provided destination address.
*withdraw(): When calling withdraw method parameter `_to` is given by input parameters as a receiver of tokens but tokens are transferred from Pool to msg.sender.

Recommendation:
Replace “msg.sender” with “_to” in the deposit and withdraw methods.

Post-audit.
The `_to` parameter is used in both methods now.

HIGH-3 | RESOLVED

Possible int256 overflow when converting uint256 to int256.

Pool.sol, multiple methods
In the internal/private methods, when performing mathematical operations, there are numerous conversions from uint256 to int256 without a proper safety check, which may result in int256 overflow.

Recommendation:
Add safety checks so that there is no int256 overflow.

Post-audit.
SafeCast library is used now.

HIGH-4 | RESOLVED

Possible blockage of the assets withdrawal.

Pool.sol, withdraw()
When removing assets by setting their status to inactive, the status can't be changed in the future. In other words, the asset's status can't be set to active again, and due to the validation that only allows for active assets withdrawal, its withdrawal is blocked.

Recommendation:
Leave the possibility to withdraw assets from the removed pools.

Post-audit.
Withdrawals are not blocked now. The Pool can also be set as active or inactive anytime.

MEDIUM-1 | RESOLVED

maxSupply of assets is never used for the safety checks in methods.

Pool.sol, deposit(), swap()
The checks for not exceeding the asset's maxSupply are missing in the deposit() and swap() methods. The maxSupply field of the AssetOcto struct might be increased or decreased by the owner but maxSupply field is not used in methods of depositing assets.

Recommendation:
Add the checks for not exceeding maxSupply when depositing assets to the Pool.

Post-audit.
Safety checks were added in the deposit() and swap() methods to ensure that assets do not exceed maxSupply when added to the Pool.

LOW-1 | RESOLVED

Missing validation checks in multiple methods.

Pool.sol, initialize(), addAssetOcto(), changeMaxSupply(), removeAsset(), withdraw()
*In initialize(): the _a and _lpFeech, arguments are not validated despite the fact that there is a check for argument <= WAD in the setter for _lpFee.
*In addAssetOcto(): the checks for whether the asset by the given token was already added and whether the given token address does not equal to zero are missing.
*In changeMaxSupply(): the checks for whether the given _tokenId is valid and for validating _newMaxSupply are missing.
*In removeAsset(): the check for whether the given _tokenId is valid is missing.
*In withdraw(): the check for whether the user owns not less LP tokens that they want to burn is missing.

Recommendation:
Add validation checks for input parameters.

Post-audit.
All validations were applied.

LOW-2 | RESOLVED

Wrong variable is validated in the setter.

Pool.sol, setA()
In the given setter, there is requirement for the old current “a” value, but not for the one that was set (“_newA”)

Recommendation:
Change the validation of the old value to the new value.

Post-audit.
Validation of the new value was implemented.

LOW-3 | RESOLVED

Possible mess with the owner rights and devaddr rights.

Pool.sol, initialize()
In the initialize() method, the devaddr variable is initialized with the msg.sender value as it is the same as the owner variable. After the initialization, devaddr can be changed by calling the setDev() method, but from the deployment till that moment, the owner and devaddr have the same value but different roles.

Recommendation:
Isolate devaddr from the owner.

Post-audit.
The devaddr role was removed.

LOW-4 | RESOLVED

The validation check for the correct liquidity value in deposit() is redundant.

Pool.sol, deposit()
In the deposit() method, the check for the liquidity to be greater than `0` (line 374) is useless since there is later another check for the liquidity to be greater than _minimumLiquidity. And as _minimumLiquidity is of uint256, then first “require” is useless.

Recommendation:
Remove the check for the liquidity to be greater than `0`

Post-audit:
Redundant validation check was removed.

LOW-5 | RESOLVED

The same asset can be added more than once.

Pool.sol, addAssetOcto()
In the addAssetOcto method, there is no check whether this asset is already added to Octo.

Recommendation:
Add a validation check for not adding the same asset more than once.

Post-audit:
The validation was implemented.

INFO-1 | RESOLVED

Code style issues.

Pool.sol, multiple methods
*According to the code style recommendations from the Solidity documentation, the order of the functions should be the following: constructor, fallback function (if exists), external, public, internal, private. In the Pool contract, the methods of all visibility are messed up.
*All of the internal and private methods' names start with "_" but not in mintAllFee(), which is marked as internal.

Recommendation:
Follow the Solidity style guide recommendations. Consider changing the name of mintAllFee() to _mintAllFee()

Post-audit:
The order of functions was changed according to the Solidity documentation.

INFO-2 | RESOLVED

Reentrancy: State variables are written after external calls.

Pool.sol, deposit()
*Reentrancy in Pool.deposit(uint256,uint256,uint256,address,uint256)
(Pool.sol#344-390) External calls:
IERC20Upgradeable(indexToAsset[_id].token).safeTransferFrom(address(msg.send
er),address(this),_amount) (Pool.sol#359-363)
State variables written after the call(s):
- _mintFee
    - indexToAsset[id].liability += liabilityToMint
    - indexToAsset[id].cash += lpDividend
- indexToAsset[_id].cash += _amount.toWad(indexToAsset[_id].decimals)
- indexToAsset[_id].liability += liabilityToMint
- indexToAsset[_id].totalSupply += liquidity
*Reentrancy in Pool.deposit(uint256,uint256,uint256,address,uint256)
(Pool.sol#344-390) External calls:
IERC20Upgradeable(indexToAsset[_id].token).safeTransferFrom(address(msg.send
er),address(this),_amount) (Pool.sol#359-363)
State variables written after the call(s):
- _mint(msg.sender,_id,liquidity)
**The issue is marked as informational because assets can be added only by the owner of the contract.

Recommendation:
Pay attention to the assets that you add to the Pool.

Post-audit:
State variables are now written before the external calls.

INFO-3 | RESOLVED

Events' arguments are not standardized.

Pool.sol, multiple events
When declaring, some of the events addresses are marked as indexed, while others have no declarations at all.

Recommendation:
Mark all addresses as indexed in the events declaration.

INFO-4 | RESOLVED

Variable's visibility should be changed to immutable.

AggregateAccount.sol, isStable
isStable is initialized in the constructor and cannot be set again, so marking this variable as immutable is preferable.

Recommendation:
Consider marking isStable variable as immutable.

INFO-5 | RESOLVED

Incrementation of lastIndex in addAssetOcto is redundant.

Pool.sol, addAssetOcto()
When creating new struct instance of indexToAsset, there is the “lastIndex + 1” indexing. After that, when adding data to the assetToIndex mapping, there is a pre-incrementation of the “++lastIndex” index.

Recommendation:
Consider pre-incrementing “lastIndex” when setting values in theindexToAsset struct instance.

Post-audit:
Pre-incrementation was implemented.

INFO-6 | RESOLVED

Some of the methods from math libraries are never used.

*SafeCast.sol: toInt256(), toUint256()
*SignedSafeMath.sol: fromWad(), sqrt(), toUint(), toWad()

Recommendation:
Remove methods that are never used.

Post-audit:
Unused methods were removed or are used now.

INFO-7 | RESOLVED

Multiple to-do comments in pre-production code.

Pool.sol, multiple methods
Pre-production contracts should not have marks for “to-do” features.

Recommendation:
Implement “to-do” features.

Post-audit:
“to-do” features were implemented.

INFO-8 | RESOLVED

The event is not emitted when executing the corresponding method.

Pool.sol, removeAsset()
In the removeAsset method, the corresponding declared event, AssetRemoved, is not emitted.

Recommendation:
Emit the AssetRemoved event in the removeAsset method.

Post-audit:
Event emitting was added.

INFO-9 | VERIFIED

Loss of the Pool share due to external swaps.

Pool.sol, swap()
When swapping assets, holders of “assetTo” get their assets decreased, while no LP tokens are moved and no “assetFrom” LP tokens are added to “assetTo” holders.
Flow example:

1) user1 deposits asset1 of the N amount, user2 deposits asset2 of the M amount
2) user1 swaps asset1 to asset2 sending asset2 to user1, but neither asset1 nor LP tokens for asset1 are sent to user2

This way, the amount of the Pool share as the amount of asset2 owned by user2 is decreased permanently. Thus, it is unclear why user2 should deposit assets to the Pool as the user is just going to receive less assets than deposited.

Recommendation:
Consider this a permanent loss of user’s assets that were deposited to the Pool.

Post-audit:
After the Symbiosis team has provided a documentation on Octopool, it was verified that such functionality is a part of the bussiness logic. Users need to wait until the Pool can pay out full liability.

Users cannot withdraw assets.

Pool.sol, withdraw()
In the withdraw method, users' funds are being withdrawn with .safeTransferFrom by spending tokens from the address of the Pool contract with no prior approval from the Pool itself.

Recommendation:
Change .safeTransferFrom to .safeTransfer.

Post-audit.
.safeTransferFrom was replaced with .safeTransfer.

Note. The issue was original marked as critical, since it prevents users from withdrawing their funds. However, according to the client's clarification, this bug was from the previous version of the contracts and was removed during the audit. Nevertheless, the issue is still present in the report as an informational, since auditors have still verified that the issue has no impact on the contract's ability to operate.

	AggregateAccount.sol	SignedSafeMath.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	DSMath.sol	SafeCast.sol	Pool.sol
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions / Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting Symbiosis in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Hardhat testing framework.

The tests were based on the functionality of the code, as well as a review of the Symbiosis contract requirements for details about issuance amounts and how the system handles these.

Contract: Pool

Pool setting: initialization, pause/unpause, setters

✓ .initialize() - Correct (81ms)
✓ .initialize() - Should not initialize twice
✓ .pause() - Admin should pause
✓ .pause() - Not admin should not pause
✓ .unpause() - Admin should not unpause if not paused
✓ .unpause() - Admin should unpause if paused
✓ .pause() - Contract don`t work if it is paused (116ms)
✓ .setDev() Admin should set new Dev (111ms)
✓ .setDev() Admin should not set AddressZero as new Dev
✓ .setA() Admin should set new correct amplification factor (110ms)
✓ .setA() Admin should not set new uncorrect amplification factor (test should pass in correct code)
✓ .setLPFee() Admin should set new correct LPFee (109ms)
✓ .setLPFee() Admin should not set uncorrect LPFee
✓ .setFeeRatio() Admin should set new correct feeRatio (109ms)
✓ .setFeeRatio() Admin should not set new uncorrect feeRatio
✓ .setFeeTo() Admin should set new correct address to fee (109ms)
✓ .setFeeTo() Admin should not set AddressZero to fee address
✓ .setMintFeeThreshold() Admin should set new mint fee threshold (112ms)
✓ .setVeSISAddress() Admin should set new VeSIS (109ms)
✓ .setVeSISAddress() Admin should not set ZeroAddress to VeSIS (test should pass in correct code)

Work with assets: adding, change, remove

✓ .addAssetOcto() - Admin should add new asset to pool
✓ .changeMaxSupply() - Admin should change max supply in existing asset
✓ .changeMaxSupply() - Admin should not change max supply in non-existing asset
(test should pass in correct code)
✓ .removeAsset() - Admin should remove existing asset

✓ .removeAsset() - Admin should not remove non-existing asset (test should pass in correct code)
✓ .removeAsset() - Admin should not remove removed existing asset (test should pass in correct code)

Deposit and Withdraw

✓ .deposit() - User should do correct deposit (112ms)
✓ .deposit() - User should not do deposit with amount = 0 (56ms)
✓ .deposit() - User should not do deposit with past deadline (55ms)
✓ .deposit() - User should not do deposit he set more minimum liquidity (85ms)
✓ .withdraw() - User should correct withdraw (123ms)
✓ .withdraw() - User should not withdraw too low amount (157ms)
✓ .withdraw() - User should not withdraw with liquidity = 0 (99ms)

Swap

✓ .swap() - User should swap with correct parameters (218ms)
✓ .swap() - User should not swap tokens what hasn`t (235ms)
✓ .swap() - User should not swap if he hasn`t deposits (105ms)
✓ .swap() - User should not swap token for itself (55ms)
✓ .swap() - User should not swap zero tokens (207ms)

Other functions and cases

✓ .globalEquilCovRatio() - Correct work (103ms)
✓ .spreadAccumulatedError() - Correct work (with amount = 0) (122ms)
✓ .spreadAccumulatedError() - Error if amount > accumulated error) (112ms)
✓ .mintFee() - Correct transfer fee (323ms)
✓ .mintFee() - Not transfer fee if dividend = 0 (294ms)
✓ .mintFee() - If fee not collected - early return (94ms)

44 passing (12s)

FILE	% STMTS	% BRANCH	% FUNCS
Pool.sol	97.58	78.33	100

We are grateful for the opportunity to work with the Symbiosis team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Symbiosis team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo