ADASWAP

SMART CONTRACT AUDIT

zokyo

November 28th, 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.

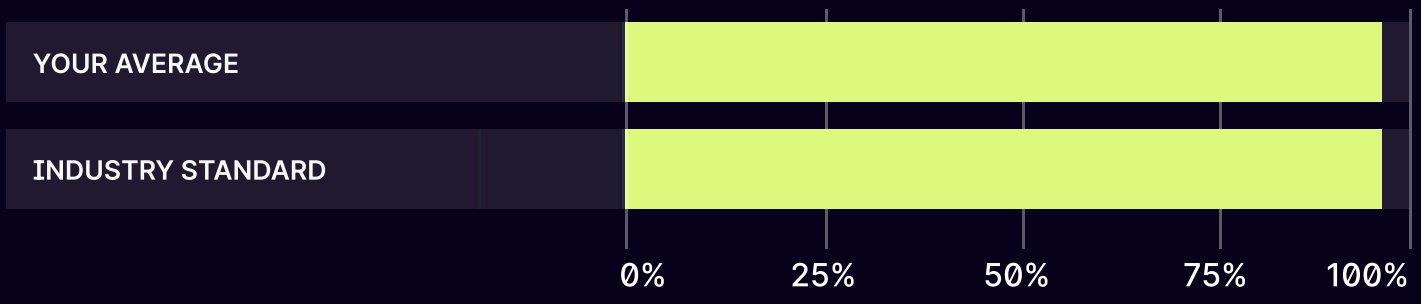
TECHNICAL SUMMARY

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

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

Contract Status

Testable Code



Testable code is sufficient for the security 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 AdaSwap 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

Repository: https://github.com/AdaSwap/adaswap-dex-smart-contracts/tree/farm

Initial commit farm branch, f3b94012aa34426bd73807f1bdfef176d4ab29d0

Final commit: farm branch, e2036dd20f0e7edc159337180029fcc05939d3b2

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

core/contracts
- AdaswapERC20.sol
- AdaswapFactory.sol
- AdaswapPair.sol
- Math.sol, UQ112×112.sol

farm/contracts
- MasterAdaSwap.sol
- Number.sol

periphery/contracts
- AdaswapRouter02.sol
- AdaswapLibrary.sol

Throughout the review process, care was taken to ensure 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 AdaSwap 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. A part of this work included writing a unit test suite using the Hardhat testing framework. In summary, our strategies consisted mostly 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.	03	Testing contract logic against common and uncommon attack vectors.
02	Cross-comparison with other, similar smart contracts by industry leaders.	04	Thorough, manual review of the codebase, line-by-line.

Executive Summary

Adaswap represents fork of UniswapV2 with own rewards distribution system. Adaswap protocol has fork of standard UniswapV2 pair, router, library and math contracts. Auditors checked all files line-by-line and compared with the original code. It was discovered, that the only changes Adaswap forked code contains, are the changes of the naming (Uniswap → Adaswap) and version (upgraded to 0.8.x). Nevertheless, there is one exception: Adaswap version of the Math library contains optimized version of the square root function. It contains two operations less. Though, these were “extra” steps, in some sence the simplification is reasonable. Auditors team has checked the formula and verified that it works in the same way as original one.

The second big part of the Adaswap protocol is their staking with own rewards system. Staking represents standard MasterChief solution with extra modifications. First of all staking has another layer of diversification: pool is divided into different locks. So the user can choose the pool to stake, and choose different time locks (from immidiate reward to 1 year lock). Therefore, staking required another approach in allocation points (internal rewards proportion) calculation. So, each pool receives part of the allocated rewards (per second), and splits it among its locks. The user is not limited to deposit into several pools and even several locks within the pool. Also, each user gets the access to the emergency withdraw function, so they can take tokens at any time (but without caring about the rewards).

Since it is an implementation of pretty standard MasterChief solution, rewards calculation logic and deposit/withdraw flow are standard. Nevertheless, auditors verified it with additional testing rounds.

Adaswap team has fixed several security loopholes: verified the 3rd party dependancy (so the check is passed), fixed allocation points calculation and several low risk issues.
The only issue left unresolved is the SOlidity version. Contracts currently utilize Solidity version which contains several issues with optimizer and compiler (0.8.13), so it needs to be upgraded to the next safe version or last stable one (0.8.17).

The overall security is evaluated as High (and it matches security standards), but the mark reflectes the issue with Solidity version used.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For the ease of navigation, document's sections are arranged from the most critical to the least critical. 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 further explanation from the Customer or remains disregarded by the Customer. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or 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

Allocated points can be updated incorrectly.
MasterAdaSwap.sol: set().

Function set() allows admin to change the proportion of rewards distribution between locks within the pool - via the change of allocPoint. Though, set() function updates the totalAllocPoint, but does not update the pool.allocPoint. In such case there will be incorrect calculation of pending rewards in all main contract flows.

Recommendation:
Verify the functionality and correct the allocation points update for pools.

MEDIUM-1 | UNRESOLVED

Obsolete Solidity version.

core: AdaswapPair.sol, AdaswapERC20.sol, AdaswapFactory.sol, core/libraries, core/interfaces
periphery: AdaswapRouter02.sol, AdaswapLibrary.sol, interfaces
lib: TransferHelper
farm: MasterAdaSwap.sol (this one uses floating version which may also lead to incorrect version used since it gets the project version)

All contracts utilize Solidity version 0.8.13 which is considered obsolete. It is highly recommended to use the latest stable Solidity version, which is currently 0.8.17.

The issue is marked as Medium, because version 0.8.13 contained 2 important bugs connected to the complex data structures. In spite of the rare chance of the bug reproducing, it is still possible to get them before the deployment. The next version (0.8.14) contained patches for those bugs and was released the next day after the 0.8.13. And since the version should be updated anyway, it is recommended to use the latest one, which contains all regular optimizer updates and regular bugfixes.

Recommendation:
Utilize the latest stable version (0.8.17)

LOW-1 | RESOLVED

Variables lack validation.

Some of the variables should be validated, especially the ones which are set in storage only one time.

- MasterAdaSwap.sol: constructor().

Parameters `_adaswapToken`, `_adaswapTreasury` should be validated not to be equal to zero address.

- MasterAdaSwap.sol: add()

Parameter '_lpToken' should be validated not to be zero address, so that an invalid address is used for creating a pool.
Issue is marked as Low, since there is no way to change those variables in case of incorrect value, thus it is recommended to add validation.

Recommendation:
Validate variables.

LOW-2 | RESOLVED

Same lp token can be added to the pools.

MasterAdaSwap.sol: function add().
Though it is mentioned in the commentary section, that same token should not be added more than one time, there is no mandatory validations in the function. As a result of a human error, for example, same token might still be added more than one time. Which is why it is recommended to add mandatory validation in the function.

Recommendation:
Validate that '_lpToken' isn't already added.

INFO-1 | RESOLVED

Gas optimization suggestions.

- MasterAdaSwap.sol: function add(), lines 101, 119.

Two loops can be united into one in order to save gas. Writing to storage struct 'lock' should be performed only when '_allocPoints[i]' is not equal to 0.
Post-audit. Optimization was applied.

INFO-2 | VERIFIED

Possible lack of rewards.

MasterAdaSwap.sol
All rewards are distributed to the users from 'AdaSwapTreasury' address. Since the Treasury contract is out of the current audit, it can't be verified if Treasury will have enough reward tokens for all the stakers, and that harvesting won't be blocked due to this. It is also unclear if Treasury will grant enough allowance to the MasterAdaSwap contract.

Recommendation:
Verify that Treasury will have enough rewards and enough allowance will be granted from Treasury to Pool.

From client.
It will be controlled manually, the treasury has enough rewards to pay.

INFO-3 | VERIFIED

3rd party contract dependency.

MasterAdaSwap.sol, AdaSwapTreasury and ASW.
As comments state, these 2 variables will contain foreign token and treasury contracts addresses. Since these contracts are not present in the scope of the audit and since they play a crucial role in the smart contract logic auditors' team needs the validation from the AdaSwap team on these items. It should be noted that both contracts are present in the "test" directory. But based on the basic structure of those contracts and that they are used as test mocks, auditors still need the validation on these 2 contracts. Otherwise it actually fails the 3rd party dependency check.
The same is applicable to rewarder contract, since it influences the rewards distribution functionality and thus makes the flow impossible to validate fully.The issue is marked as info, since it heavily depends on AdaSwap vision and cannot be classified at this stage.
Recommendation:
Verify treasury and token contracts are the same as in the Test directory or provide the correct code or other validation proof to resolve 3rd party dependency.
From client: the implementation of treasury contract was verified to be identical to farm/contracts/test/Treasury.sol.
The implementation of ASW token is https://explorer-mainnet-cardano-evm.c1.milkomeda.com/address/0xAb033ae98BEeB92C267D1f02E3963A424A02B406/coin-balances

INFO-4 | RESOLVED

Unused code.

Number.sol: functions Uint128.to64(uint128 a), UInt256.to128(uint256 a).
These functions are never used in MasterAdaSwap.sol which is why it is recommended to remove them.

Recommendation:
Remove unused code.

	AdaswapERC20.sol	AdaswapFactory.sol	AdaswapPair.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

	Number.sol	MasterAdaSwap.sol	AdaSwapRouter02.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 AdaSwap 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 AdaSwap contract requirements for details about issuance amounts and how the system handles these.

MasterAdaSwap: additional scenarios

- ✓ Should deposit and harvest in different locks at the same time (199ms)
- 1) Should set allocation points to the pool correctly
- ✓ Should harvest rewards after withdrawal (136ms)
- ✓ Should harvest same rewards with and without deposit before harvest (224ms)
- ✓ Should harvest same rewards with and without withdraw before harvest (204ms)
- ✓ Should harvest same rewards with and without partial withdraw (118ms)

MasterAdaSwap: additional scenarios

Test 'Deposit' function`

- ✓ Should revert 'MasterAdaSwap: POOL_DOES_NOT_EXIST' (230ms)
- ✓ Should update Pool (172ms)
- ✓ Should change user amount (156ms)
- ✓ Should change user rewardDebt (194ms)
- ✓ Should change user lastDepositTime (177ms)
- ✓ Should change lock.supply (155ms)
- ✓ Should change pool.weight (159ms)
- ✓ Should check if rewarder address not equal ZERO address (151ms)
- ✓ Should change add balance on masterAdaSwap (address(this)) (181ms)
- ✓ Should check if rewarder is address(0) (145ms)

Test 'Harvest' function

- ✓ Should revert 'MasterAdaSwap: FIXED_LOCK_TIME_IS_NOT_OVER' (172ms)
- ✓ Should update Pool (179ms)
- ✓ Should check current Lock Info (165ms)
- ✓ Should check new user reward Debt (175ms)
- ✓ Should emit 'Harvest' (192ms)

Test 'Withdraw' function

- ✓ Should revert 'MasterAdaSwap: FIXED_LOCK_TIME_IS_NOT_OVER' (144ms)
- ✓ Should update Pool (162ms)
- ✓ Should change user amount (225ms)
- ✓ Should check new user reward Debt (183ms)

✓ Should return correct amount lock.supply after withdraw (163ms)
✓ Should return correct amount pool.weight after withdraw (177ms)
✓ Should change balance after withdraw (164ms)
✓ Should check if rewarder is address(0) (161ms)
✓ Should emit 'Withdraw' (198ms)
Test 'WithdrawAndHarvest' function
✓ Should revert 'MasterAdaSwap: FIXED_LOCK_TIME_IS_NOT_OVER' (172ms)
✓ Should update Pool (202ms)
✓ Should check current Lock Info (177ms)
✓ Should check new user reward Debt (176ms)
✓ Should return correct user.amount (192ms)
✓ Should return correct lock.supply (221ms)
✓ Should return correct pool.weight (190ms)
✓ Should check if rewarder is address(0) (185ms)
✓ Should emit 'Harvest' & 'Withdraw' (171ms)
Test 'emergencyWithdraw' function
✓ Should update Pool (166ms)
✓ Should change user amount (172ms)
✓ Should check new user reward Debt (199ms)
✓ Should check lock.supply (162ms)
✓ Should return correct amount pool.weight after withdraw (177ms
✓ Should change balance after withdraw (176ms)
✓ Should check if rewarder is address(0) (161ms)
✓ Should emit 'Withdraw' (175ms)
Test 'nextUnlockedTime' function
✓ Should return 'userInfo' (210ms)
Test 'poolsLength' function
✓ Should return 'poolInfoLength' (142ms)
Test 'lockTimesLength' function
✓ Should return 'lockTimesLength' (131ms)
Test 'lockBitMasked' function
✓ Should return 'mask' (151ms)
Test 'setRewarder' function
✓ Should change rewarder address (172ms)
Test 'pendingAdaSwap' function
✓ Should return pending (136ms)
✓ Should return pending after time travel (149ms)
Test 'setRewarder' function
✓ Should change rewarder address (194ms)
Test 'add' function
✓ Should pass with new allocation points (185ms)
Test 'massUpdatePools' function
✓ Should update pools (223ms)

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by the AdaSwap team

As a part of our work assisting AdaSwap in verifying the correctness of their contract code, our team has checked the complete set of tests prepared by the AdaSwap team.

We need to mention that the original code has a significant original coverage with testing scenarios provided by the AdaSwap team. All of them were also carefully checked by the team of auditors.

MasterAdaSwap

- ✓ 0. testMasterAdaSwap: Pool should exist (44ms)
- ✓ 1. testMasterAdaSwap: PendingAdaSwap should equal ExpectedAdaSwap (413ms)

Deposit

- ✓ 2. testMasterAdaSwap: Deposit 1 lptoken (86ms)
- ✓ 3. testMasterAdaSwap: Deposit does not exist (58ms)

Withdraw

- ✓ 4. testMasterAdaSwap: Withdraw 10 amount lock time is no over (88ms)
- ✓ 5. testMasterAdaSwap: Withdraw 10 mount (97ms)
- ✓ 6. testMasterAdaSwap: Withdraw 0 mount (55ms)

Harvest

- ✓ 7. testMasterAdaSwap: Harvest lock time is no over (52ms)
- ✓ 8. testMasterAdaSwap: Should give back the correct amount of reward (145ms)
- ✓ 9. testMasterAdaSwap: Harvest with empty user balance (73ms)

Emergency Withdraw

- ✓ 10. testMasterAdaSwap: Lock time is not over (83ms)
- ✓ 11. testMasterAdaSwap: Lock time is over (85ms)
- ✓ 12. testMasterAdaSwap: Transfer amount is zero (48ms)

Withdraw And Harvest

- ✓ 13. testMasterAdaSwap: Lock time is not over (67ms)
- ✓ 14. testMasterAdaSwap: Withdrawal of nonzero amount (103ms)
- ✓ 15. testMasterAdaSwap: Withdrawal of zero amount (119ms)

Update Pool

- ✓ 16. testMasterAdaSwap: Total LP supply is zero
- ✓ 17. testMasterAdaSwap: Total LP supply is nonzero (105ms)

Add

- ✓ 18. testMasterAdaSwap: Should add pool with corresponding allocation points (53ms)

MasterAdaSwap

Check rewards

✓ 19. testMasterAdaSwap: Stop Reward counting (410ms)

Fake operations

✓ 20. testMasterAdaSwap: Should revert if not Onlyowner

✓ 21. testMasterAdaSwap: Should revert if 0 Address

✓ 22. testMasterAdaSwap: Should revert if fake account

MasterAdaSwap Use Cases

✓ 23. testMasterAdaSwap: One staker already in pool, 2 entered and then exit positions (234ms)

✓ 24. testMasterAdaSwap: Two stakers already in pool, 1 entered and then first two exit positions (317ms)

✓ 25. testMasterAdaSwap: Two users are already staking and harvesting in different periods of time (231ms)

✓ 26. testMasterAdaSwap: Two users are staking, then other two users start staking. After that first two increase their staking amount (357ms)

MasterAdaSwap

Creating pools with different time lock

✓ 27. testMasterAdaSwap: Pool with time lock - 14 days (276ms)

✓ 28. testMasterAdaSwap: Pool with time lock - 60 days (337ms)

✓ 29. testMasterAdaSwap: Pool with time lock - 365 days (258ms)

✓ 30. testMasterAdaSwap: Pool with diffrent time lock in same pool (224ms)

31 passing (27s)

AdaswapERC20

✓ name, symbol, decimals, totalSupply, balanceOf, DOMAIN_SEPARATOR, PERMIT_TYPEHASH (410ms)

✓ approve (81ms)

✓ increaseAllowance (102ms)

✓ decreaseAllowance (119ms)

✓ transfer (85ms)

✓ transfer:fail

✓ transferFrom (150ms)

✓ transferFrom:max (160ms)

✓ permit (525ms)

AdaswapFactory

✓ feeTo, feeToSetter, allPairsLength (41ms)

✓ createPair (337ms)

✓ createPair:reverse (328ms)

✓ createPair:gas

✓ setFeeTo (50ms)

✓ setFeeToSetter (73ms)

✓ pairCodeHash (72ms)

AdaswapPair

- ✓ mint (334ms)
- ✓ getInputPrice:0 (966ms)
- ✓ getInputPrice:1 (212ms)
- ✓ getInputPrice:2 (137ms)
- ✓ getInputPrice:3 (124ms)
- ✓ getInputPrice:4 (114ms)
- ✓ getInputPrice:5 (119ms)
- ✓ getInputPrice:6 (105ms)
- ✓ swap:token0 (115ms)
- ✓ swap:token1 (105ms)
- ✓ swap:gas
- ✓ burn (122ms)
- ✓ price{0,1}CumulativeLast (152ms)
- ✓ feeTo:off (136ms)
- ✓ feeTo:on (147ms)

31 passing (11s)

	Stmts	Branch	Funcs
MasterAdaSwap.sol	96.77	80.95	94.44
AdaswapERC20.sol	100	58.33	100
AdaswapFactory.sol	100	78.57	100
AdaswapPair.sol	93.59	66.67	91.67

We are grateful for the opportunity to work with the AdaSwap 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 AdaSwap team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo