SMART CONTRACT AUDIT
Report 4 of 6: Asteroid Farms

zokyo

March 14th 2023 | 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

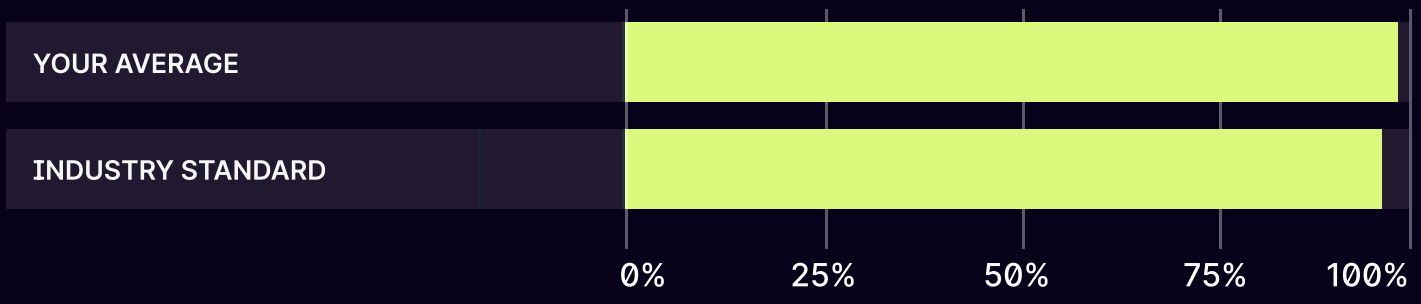
This document outlines the overall security of the TangleSwap smart contracts evaluated by the Zokyo Security team.

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

Contract Status

There were 0 critical issues found during the audit. (See in the Complete Analysis, started from 6 page)

Testable Code



98,14% of the code is testable, which is above the industry standard of 95%.

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 IOTA & Shimmer network's fast-paced and rapidly changing environment, we recommend that the TangleSwap 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 TangleSwap repository: https://github.com/TangleSwap/tangleswap-farming

Last commit: c612d13db2f2748ac201ba6ed84babae9fe45649

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

- Base.sol
- DynamicRange.sol
- FixRange.soll

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 reentrancy attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of TangleSwap 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 Hardhat 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.	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

There was no critical issue found during the audit, but two with high severity, two with medium severity and some informational issues. All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner and the investors interacting with it. They are described in detail in the “Complete Analysis” section.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the TangelSwap team and the TangelSwap team is aware of it, but they have chosen to not solved it. 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

FINDINGS SUMMARY

#	Title	Risk	Status
1	Exposed to reentrancy	High	Resolved
2	Set is not updated after emergencyWithdraw	High	Resolved
3	Unhandled edge case	Medium	Resolved
4	Centralization Risk	Medium	Resolved
5	Lock Solidity version	Informational	Resolved
6	Unnecessary public view visibility	Informational	Resolved
7	Costly error messages	Informational	Acknowledged
8	Transactions can be excluded due to block gas limit while calling collectAllTokens()	Informational	Resolved

HIGH | RESOLVED

Exposed to reentrancy

FixRange.sol & DynamicRange.sol - `emergencyWithdraw` is exposed to reentrancy due to `safeTransfer`. An attacker can reenter this method by having a `external call` implemented in his `onERC721Received` that will call other functions from the same contract context or different contract context. Other way is to reenter other methods like `withdraw` or `collectReward`. This can be highly severe because the state is not updated before the reentrancy

Recommendation:
Apply a wide reentrancy guard using best practices pattern and modifiers.

HIGH | RESOLVED

Set is not updated after emergencyWithdraw

FixRange.sol & DynamicRange.sol - `emergencyWithdraw` does not remove the `tokenId` from the enumerable set `tokenIds`. This will lead to several issues, including a Denial of Service issue to `collectRewards` on the user's other legitimate tokenIds.

Recommendation:
Update `tokenIds` by removing that token from the enumerable set on the emergencyWithdraw transaction.

MEDIUM | RESOLVED

Unhandled edge case

Base.sol - In constructor, in `setRewardPool` - the case in which `token0 = token1` which is believed not to be desired (i.e. according to understanding of the contract). This case is not prevented from occuring and not even considered here, `token0 < token1`.

Recommendation:
add a requirement statement to ensure that `token0 != token1`.

MEDIUM | RESOLVED

Centralization Risk

Admin enjoys too much authority. The general theme of the repo is that admin has power to call several functions like emergencyWithdraw, general setters/mutators. Some functions can be more highly severe to be left out controlled by one wallet more than other functions; depending on the intentions behind the project. Recommendation Apply governance / use multisig wallets.

INFORMATIONAL | RESOLVED

Lock Solidity version

All Contracts, Lock the pragma to a specific version, since not all the EVM compiler versions support all the features, especially the latest one's which are kind of beta versions, So the intended behavior written in code might not be executed as expected. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler, which may have higher risks of undiscovered bugs. Recommendation fix version to 0.8.9

INFORMATIONAL | RESOLVED

Unnecessary public view visibility

In FixRange.sol - `tokenStatusLastTouchAccRewardPerShare` is unnecessarily `public` while it can be limited to `external`. It is one of the best practices to limit the access modifier to what is needed.

INFORMATIONAL | ACKNOWLEDGED

Costly error messages

Error messages in some require statements are long and considered costly in terms of gas.

Recommendation:

- Write shorter error messages
- Use custom errors which is the goto choice for developers since solidity v0.8.4 details about this shown here: soliditylang.

INFORMATIONAL | RESOLVED

Transactions can be excluded due to block gas limit while calling `collectAllTokens()`

In the contract **DynamicRange**, the loop in line 396 iterates over all the tokens owned by the user and collects the rewards. This for loop has no limit on the number of interactions. If the user owns a large number of tokens and calls the functions, the transaction can possibly exceed the block gas limit. Thus, the transaction will never be included in the block.

Recommendation:
Users can alternatively call **collect** function for each token individually. Limit the number of iterations in the loop by adding **from** and **to** parameters to the function call. Using these 2 parameters, the user can decide the range in the array for which the tokens should be collected.

Note #1:
This issue was fixed by providing a method called collect that allows user to collect pending reward one NFT at a time, this covers a majority of use case.
Second, the client also have the intention to put contraints on the frontend to collect no greater than NFT rewards at a time, to mitigate the out of gas risk. Only those who do not respect the constraints on the frontend will run into “out of gas” problem by invoking scripts to collect large batch of NFTs’ rewards will get into this problem.

	Base	FixRange	DynamicRange
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)	Fail	Fail	Fail
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 TangleSwap 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 TangleSwap contract requirements for details about issuance amounts and how the system handles these.

DynamicRange

Testing initial values
✓ tests contract initialization (2800ms)
deposit()
✓ #deposit (3815ms)
✓ #deposit (1912ms)
✓ #deposit (2011ms)
✓ #deposit (1757ms)
✓ #deposit (1780ms)
✓ #deposit (1817ms)
withdraw()
✓ #withdraw (2776ms)
✓ withdraw() (2422ms)
safeTransferEth()
✓ safeTransferEth() (2491ms)
emergenceWithdraw()
✓ #emergenceWithdraw (2306ms)
collectAllTokens()
✓ #collectAllTokens (2402ms)
#collect()
✓ collect() (2449ms)
✓ #collectAll() (2463ms)
#Getters
✓ getOraclePrice(), getMiningContractInfo() (2483ms)

Base

✓ Should be deployed correctly (51ms
✓ Should be able to modify start block (146ms)
✓ Should be able to modify end block (79ms)
✓ Should be able to modify provider (83ms)
✓ Should be able to get token ids (71ms)

✓ Should charge receiver to collect fees (159ms)
✓ Should be able to get pending rewards (77ms)
✓ Should be able to get pending reward (46ms)
✓ Should be able to modify charge receiver (45ms)
✓ Should be able to modify reward per block (86ms)
✓ Should be able to get reward block num (52ms)
✓ Should be able to update global status (91ms

FixRange

✓ Should be deployed correctly (316ms)
✓ Should do on ERCK721Received
✓ Should allow deposits (471ms)
✓ Should allow withdrawals (741ms)
✓ Should be able to get tokenStatus LastTouch AccRewardPerShare (145ms)
✓ Should allow emergency withdrawals (195ms)
✓ Should be able to collect reward (261ms)
✓ Should be able to collect rewards (276ms)
✓ Should able to get mining contract info

36 passing (1m)

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
Base.sol	100	96.3	100	100	
DynamicRange.sol	95.29	81.43	100	95.52	302, 347, 348
Base.sol	100	100	100	100	
FILE	**98.14**	**91.07**	**100**	**98.14**	

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

zokyo