SMART CONTRACT AUDIT
Report 6 of 6: Investment Hub

zokyo

April 27th 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

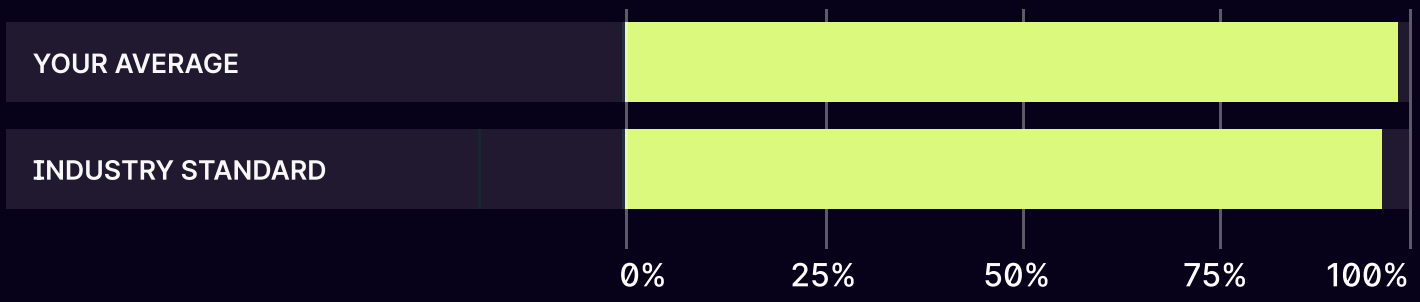
This document outlines the overall security of the TangleSwap smart contract evaluated by the Zokyo Security team.

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 Complete Analysis)

Testable Code



100% 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-investment-hub

Last commit - ad45220a22021a6879b2af5f25885ff75c949e6f

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

- PostAuctionLauncher.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 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 contract by industry leaders.	04	Thorough manual review of the codebase line by line.

Executive Summary

There were no critical issues found during the audit, alongside one with high severity and some medium, low severity, and one informational issue. All the mentioned findings may have an effect only in the 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 TangleSwap team and the TangleSwap 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	Lack of appropriate access control	High	Resolved
2	Lack of safety measures in depositETH() function	Medium	Resolved
3	Centralization risk and possibility of a Denial of Service	Medium	Resolved
4	Possibility of incorrect minting of liquidity via finalize()	Medium	Resolved
5	Missing zero value check for locktime	Low	Resolved
6	Increase test coverage	Informational	Acknowledged

HIGH | RESOLVED

Lack of appropriate access control

The initAuctionLauncher() function lacks appropriate access controls. Anyone can call this initAuctionLauncher() function and initialize himself as an admin.

Recommendation:
It is advised that initAuctionLauncher is called only by the address that has deployed the PostAuctionLauncher rather than relying on quickly calling this init function after deployment of the contract. This will add more security to the initialization step and will reduce any chances of unwanted errors.

MEDIUM | RESOLVED

Lack of safety measures in depositETH() function

Part A) depositETH() allows anyone to deposit ETH(i.e. token1 or token2 if it's a WETH) at any point of time. But it can lead to bypass of checks in _deposit() function for the same token1 or token2. This can result in the bypass of some important require checks, such as -

```
require(!launcherInfo.launched, "Must first launch liquidity");
require(launcherInfo.liquidityAdded == 0, "Liquidity already added");
```

- specifically, in case when depositETH() is used instead of _deposit() (i.e. depositToken1() or depositToken2()). This can lead to incorrect assumptions being made, which may lead to unintended issues.

Recommendation:
It is advised to review the business and operational logic and add the needed checks accordingly.

Part B) Moreover, A malicious admin can withdraw ETH(i.e. WETH that was received to the contract via depositETH() function), especially in case when finalize() has been already been called before.

Recommendation:

To fix this, disallow calling depositETH() function once finalize() has been called or refund the deposited ETH of finalize() has been already called(i.e. when launcherInfo.launched is true).

MEDIUM | RESOLVED

Centralization risk and possibility of a Denial of Service

A malicious admin can call cancelLauncher() and drain all the deposited tokens from the contract.

Recommendation:

It is advised to add a multisig or a governance mechanism to prevent this from happening.

MEDIUM | RESOLVED

Possibility of incorrect minting of liquidity via finalize()

Depending upon the logic of TangleSwapCallingParams.mintParams(which is out of scope of this audit) if the parameters-

uint160 sqrtPriceX96,
int24 tickLower,
int24 tickUpper

- to the finalize() function are provided incorrect values, which are not corresponding to the getTokenAmounts() values (i.e. lesser than the expected amount), it may lead to lesser liquidity getting minted or added in the liquidity pool. The rest of the liquidity would remain in the contract and which will require an admin to intervene by removing the remaining liquidity via withdrawDeposits(), and then add it manually to the TangleSwap pool, which introduces centralization risk of a malicious admin draining and removing all the remaining liquidity.

Recommendation:

Therefore, it is advised to check if the following scenarios can arise and introduce specific measures such as an access control mechanism to finalize function so that this issue is prevented.

LOW | RESOLVED

Missing zero value check for locktime

Missing zero value, check for _locktime. It is possible that the _locktime is accidentally set to zero. This will result in the unlocktime same as the time when the pool was launched using finalize().

Recommendation:

To avoid this issue, it is advised to add a non-zero requirement check as well for locktime in the initAuctionLauncher() function.

INFORMATIONAL | ACKNOWLEDGED

Increase test coverage

It is advised to have more test coverage and integration tests for the contract. This is because the contracts with which the PostLauncher is interacting (which are out of the scope of this audit) might have been modified and can be different from Miso's original contracts.

Recommendation:

Thus, more thorough testing is advised to check whether the same PostLauncher logic holds true for the contracts with which PostLauncher is interacting.

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

PostAuctionLauncher

- ✓ Should be able to init auction launcher (1055ms)
- ✓ Should allow eth to be deposited (616ms)
- ✓ Should allow token1 to be deposited (252ms)
- ✓ Should allow token2 to be deposited (226ms)
- ✓ Should check if market is connected (149ms)
- ✓ Should be able to withdraw nft (373ms)
- ✓ Should be to cancel launcher (206ms)
- ✓ Should be able to set wallet (191ms)
- ✓ Should be able to withdraw deposits (286ms)
- ✓ Should be able to finialize (994ms)
- ✓ Should be able to finalize with pair already present (1002ms)
- ✓ Should be able to get lp token address (168ms)
- ✓ Should be able to init launcher (213ms)
- ✓ Should ensure weth is deposited when non-weth address sends eth (433ms)
- ✓ Should be able to get token amounts (200ms)

15 passing (19s)

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

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	% UNCOVERED LINES
PostAuctionLauncher.sol	100	95.24	100	100	
All Files	**100**	**95.24**	**100**	**100**	

Zokyo Assertion over the TangleSwap Investment-Hub repository:

In the TangleSwap iteration, which has been forked from Sushiswap Miso, only a select few disparities have been observed that were not encompassed within a previous Halborn audit report. It is essential to note that, apart from these identified deviations, the remaining files have preserved their original integrity and contextual logic. Consequently, no potential security vulnerabilities or risks are anticipated to stem from the forked nature of TangleSwap in relation to Sushiswap Miso.

Contract PostAuctionLauncher is very identical to Sushiswap Miso PostAuctionLauncher except that TangleSwap introduces the following modifications to suit their use case (thus extended it to be able to handle NFTs). The TangleSwap team introduced these new state variables, interfaces and function modifications to the contract.

1. **fee global variable:** This is introduced into the contract by TangleSwap.
2. **IHubAuction:** This interface is just TangleSwap version of IMisoAuction with only one additional function fee().
3. **finalize(uint160 sqrtPriceX96, int24 tickLower, int24 tickUpper):** The finalize function takes 3 arguments. These arguments are specific to TangleSwap. The parameters "sqrtPriceX96" is used to initialize the pool contract, whereas tickLower and tickUpper also passed to TangleSwap nonfungiblePositionManager contract mint function to mint new position. These parameters do not directly alter the state of PostAuctionLauncher.
4. **initAuctionLauncher():** Function also sets the necessary state variables in order to properly initialize the PostAuctionLauncher contract. Except that this contract sets tokenPair variable by calling TangleSwap's factory contract getPool function. The getPool function in TangleSwap factory contract is also safe as it only gets the value stored in a mapping in the factory contract.
5. **withdrawNft():**This function does something similar to withdrawLPToken() in Miso PostAuctionLauncher except that it is extended to handle withdrawals of NFT. This function does enough checks to ensure that the nft is approved to the contract before it's allowed transferred to the wallet stored in contract.

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