TangleSwap

SMART CONTRACT AUDIT
Report 3 of 6: NFT Docking Stations

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

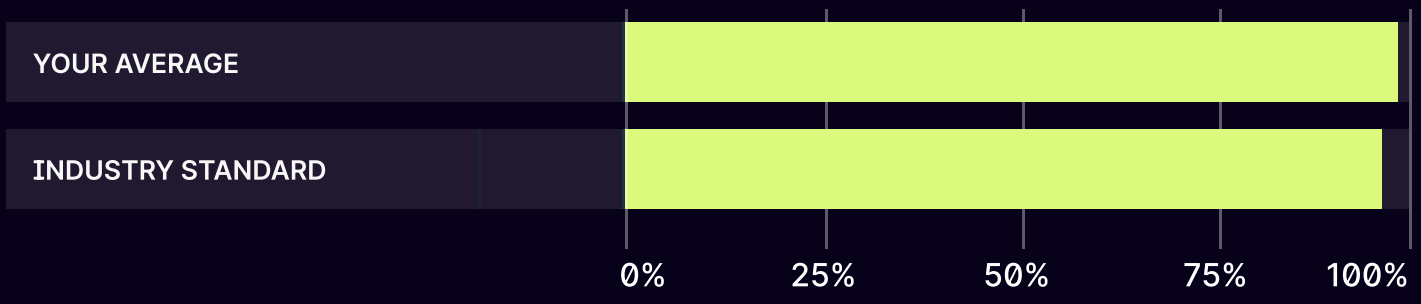
This document outlines the overall security of the TangleSwap smart contracts evaluated by the Zokyo Security team.

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



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-nft-staking/

Last commit: `0c3ab79b8811c4d3850d713789dc6c72fb6d1e95`

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

- TangleStaking.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 contracts by industry leaders.	04	Thorough manual review of the codebase line by line.

Executive Summary

The contract is well written and structured. There was no critical issue found during the audit, but were found one issue with high severity and some medium and low severity and a couple of 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 possible DoS due to gasLimit	High	Resolved
2	No validation of apy	Medium	Resolved
3	Centralization Risk	Medium	Resolved
4	Use SafeERC20	Medium	Resolved
5	Unsafe ERC721 transfer	Low	Resolved
6	Lost ERC20 rewards	Low	Resolved
7	Error objects to save gas	Informational	Acknowledged
8	Limit access modifier	Informational	Resolved
9	No way back for allowList	Informational	Resolved
10	Setter applied with no event emitted	Informational	Resolved

HIGH | RESOLVED

Exposed to possible DoS due to gasLimit

TangleStaking.sol - in `_unstake`

```
uint256 index = Array.indexOf(stakedNftById[_nftAddress][_msgSender()], _tokenId);
uint256 length = stakedNftById[_nftAddress][_msgSender()].length;
for (uint256 i = index; i < length - 1; i++) {
    stakedNftById[_nftAddress][_msgSender()][i] = stakedNftById[_nftAddress][_msgSender()][i + 1]
}
stakedNftById[_nftAddress][_msgSender()].pop();
```

For loop in `unstake` is costly since it includes a storage operation also. The high gas cost is an issue by itself but what makes this highly severe is that it is susceptible to Denial of Service. An investor might be denied to receive his token if he holds too many staked tokens in TangleStaking and seemingly there is no way around it.

Recommendation:
Might be a better option to use enumerable set to represent the token ids in `stakedNftById`. Or adopt a different operation to remove the token Id if preserving order is unnecessary.

MEDIUM | RESOLVED

No validation of apy

In `setApys`, there is no validation check for the input value. This issue severity is higher than it is usually due to the impact a mistake in new value can have on the tokenomics of the ecosystem.

Recommendation:
`require` statement to check input new value.

MEDIUM | RESOLVED

Centralization Risk

There are multiple instances where the contract owner can change protocol parameters that affect the behavior of the contracts. These include calls to several functions like updating rewarding details and general setters/mutators. More severely though is the action to with draw ERC20 asset from the contract, this alleviates the concern, hence recommendation on how to deal with it should be followed.

Recommendation:
Consider using multisig wallets or having a governance module

MEDIUM | RESOLVED

Use SafeERC20

It is recommended to utilize `SafeERC20` in order to do the transfers of tokens (i.e. as `safeTransfer`). Devs preferred to utilize transfer wrapped in a `require` statement which appears safe, but it is not the proper way to deal in case we have "Bad ERC20" that `SafeERC20` takes account for in its implementation.

Recommendation:
Refactor the usage of transfer with safeTransfer.

LOW | RESOLVED

ERC721 transfer does not respect check-effect-interact pattern

In _stake and _unstake methods the transfer of nft might represent a re-entrancy risk because of the hook of onERC721Received, it would be best to protect from this possibility by implementing a proper check-effect-interact pattern by making the external calls to the transfer functions as the final call in the contract.

Recommendation:
Refactor to implement a proper check-effect-interact pattern

LOW | RESOLVED

Lost ERC20 rewards

In `unstakeNft` - if `totalFunding` is less than amount or reward, the investor is worthy to receive. The investor would not receive rewards, which is obvious anyway, but `stakeNftRewardInfo` record for that investor for the staked NFT is deleted. Hence, no chance for the investor to receive the rewards even later.

INFORMATIONAL | ACKNOWLEDGED

Error objects to save gas

Starting from solidity 0.8.4 (project uses 0.8.7) - solidity developers are recommended to use custom error objects to save gas and also better error information as explained here in solidity blog.

Recommendation:
Use custom errors instead of require statements to revert.

INFORMATIONAL | RESOLVED

Limit access modifier

Most public methods are not invoked internally in contract, hence it is recommended to limit the access modifier to external for those methods.

INFORMATIONAL | RESOLVED

No way back for allowList

In `setAllowList` - this method sets the allow list on nft contracts to true. On the other hand there is no way in this contract to revert that change and set it to false. This issue can be regarded as a note as in case this feature is intentional the issue shall be irrelevant.

Recommendation:
a method needed to enable admin to remove from allowList

INFORMATIONAL | RESOLVED

Setter applied with no event emitted

In `setAllowList` & `setApys` - according to the general theme of the project, it is noticed that events are emitted after calling the setter methods in this contract. While for `setAllowList` no event describing this mutation taking place is emitted.

Recommendation:
emit event for setAllowList and setApys.

	TangleStaking.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.

TangleStaking

- ✓ Should be deployed correctly (296ms)
- ✓ Should be able to set base apy (194ms)
- ✓ Should be able to set lock period
- ✓ Should be able to set base reward principal
- ✓ Should be able to get erc20 balance (93ms)
- ✓ Should be able to get nft balance (98ms)
- ✓ Should allow owner to pause contract (125ms)
- ✓ Should allow owner to unpause contract (156ms)
- ✓ Should be able to set allow list (154ms)
- ✓ Should be to stake nft (314ms)
- ✓ Should be to unstake nft (614ms)
- ✓ Should allow interest to be claimed (401ms)
- ✓ Should be able to set claim duration
- ✓ Should be able to get staked nft by id (122ms)
- ✓ Should getTotalLockedPerRate (151ms)
- ✓ Should be able to fund contract (86ms)
- ✓ Should be able to withdraw (88ms)
- ✓ Should be able to get reward info (205ms)
- ✓ Should be able to get reward info (218ms)
- ✓ Should be able to get claimable amount (168ms)

20 passing (20s)

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

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	% UNCOVERED LINES
contracts/	100	97.73	100	100	
TangleStaking.sol	100	97.73	100	100	
FILE	**100**	**97.73**	**100**	**100**	

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