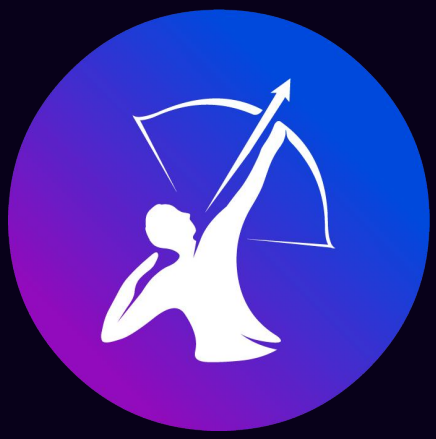

SMART CONTRACTS REVIEW

zokyo

September 19th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed security review.

TECHNICAL SUMMARY

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

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

Contract Status

There were 0 critical issues found during the audit. (See Complete Analysis)

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

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contracts was taken from the TigrisTrade repository: https://github.com/Tigris-Trade/Contracts/blob/main/contracts

Last commit - acf0538e974832eeebb86ea541c1c1c19abaf724fd

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

- Trading.sol
- TradingExtension.sol
- TradingLibrary.sol

During the audit, Zokyo Security ensured that the contract(s):

- 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 recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of TigrisTrade 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. 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	Thorough manual review of the codebase line by line.

Executive Summary

Although the review did not uncover any critical vulnerabilities, it did reveal concerns of different degrees of severity, including both high and medium levels, along with one informational issue. Further details regarding these findings can be found in the section labeled "Comprehensive Analysis."

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 TigrisTrade team and the TigrisTrade team is aware of it, but they have chosen to not solve 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	Logic edge case allow to bypass deposit and mint StableVault tokens	High	Resolved
2	Inaccurate update of limitPrice	Medium	Acknowledged
3	Possible underflow in decimals calculations	Medium	Acknowledged
4	Use safeTransferFrom and safeTransfer from SafeERC20	Medium	Resolved
5	Redundant use of uint256	Informational	Resolved

COMPLETE ANALYSIS

HIGH-1 | RESOLVED

Logic edge case allow to bypass deposit and mint StableVault tokens

In contract Trading.sol, function _handleDeposit at line #769, unchecked data might lead to undefined behavior. Given that _permitData is an input value from an external function, the field `usePermit` can be set to false, therefore avoiding the ERC20Permit function call. This way the execution continues to the next block, where the marginAsset is transferred to the Trading contract from the `_trader` and the result from the transferFrom function is unchecked so if the token will not revert and returns false the execution will continue, in conlusion even if the _marginAsset has not been approved by the trader, the contract still calls the StableVault `deposit` function which mints stable vault tokens.

Recommendation:
Add whitelisting for tokens and don't whitelist tokens with empty fallback functions if they don't have permit functionality, and also add the SafeERC20 library and refactor from transferFrom to safeTransferFrom function.

MEDIUM-1 | ACKNOWLEDGED

Inaccurate update of limitPrice

In contract TradingExtension in function "_limitClose" at lines #94 and #102 the variable "_limitPrice" is updated using "_trade.tpPrice" or "_trade.slPrice" fields. The assignment of these variables happens if all the previous checks passed, but given those checks' logic, it might be inaccurate to update the _limitPrice using the _trade fields if the price have spiked considerable more then the "tpPrice" or "slPrice"

Recommendation:
Use the _price variable to update the _limitPrice.

MEDIUM-2 | ACKNOWLEDGED

Possible underflow in decimals calculations

In contract Trading in function “_handleDeposit” at line #775 the “_marginDecMultiplier” is the result of calculating the multiplier based on the “_marginAsset” decimals. The calculation assumes that the token's decimals are always smaller or equal than 18, but this is not always the case. Using the hardcoded value can lead to an underflow in the following expression “18 - ERC21(_marginAsset).decimals”. This reverts, but it still posses a risk for the trading contract as it makes a _marginAsset unusable due to the calculations always failing.

Recommendation:
Don't use a hardcoded value and handle such cases when margin assets might not follow the standard 18 decimals or less, that most tokens follow.

MEDIUM-3 | RESOLVED

Use safeTransferFrom and safeTransfer from SafeERC20

In contract Trading.sol, there are multiple places where the IERC20 interface is used for the interaction with several tokens. However, there are also cases where the result of functions is not checked, for example ‘transfer’ function at line 804 which can result in weird edge cases as there are a lot of ERC20 tokens who do not behaves as developers usually expect.

Recommendation:
Add the usage of SafeERC20 library to ensure safe token transfers, in line with best practices.

INFORMATIONAL-1 | RESOLVED

Redundant use of uint256

In contract TradingExtension, function “getVerifiedPrice” at line #143, the function argument “_withSpreadIsLong” is declared as uint256 even it acts as a flag and takes small values. This is also found in function “initiateLimitOrder” for “_orderType”

Recommendation:
Use uint8 for such variables that act as flags

	Trading.sol TradingExtension.sol TradingLibrary.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

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

zokyo