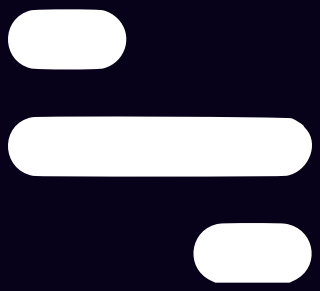


SMART CONTRACT AUDIT

zokyo

September 4th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed the security audit.

TECHNICAL SUMMARY

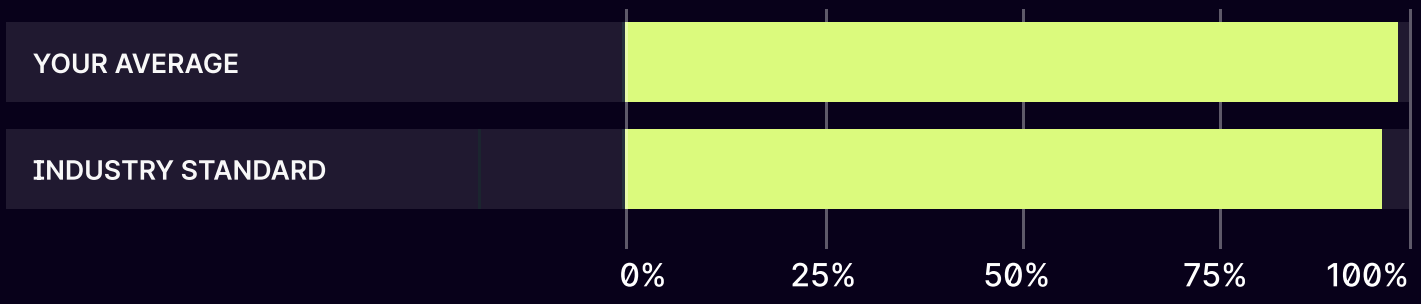
This document outlines the overall security of the Tradable smart contracts evaluated by the Zokyo Security team.

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

Contract Status

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

Testable Code



99.21% 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 contracts 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 Tradable 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 contract was taken from the Tradable repository:
https://github.com/TradableFinanceDevs/tradable-exchange-backend

Last commit - 475d15f5fd7bec47aa7bb0b1869a72992c07e69b

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

- TradableStaking.sol
- TradableSideVault.sol
- StableToken.sol
- TradableMarginHandler.sol
- TradableMarginVault.sol
- TradableSettings.sol
- TradableSettingsMessageAdapter.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 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 Tradable 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 contracts 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 Zokyo team conducted an extensive examination of the Tradable codebase. While the audit did not uncover any critical problems, it did reveal five issues of significant severity, along with several of medium, low severity, and informational nature. A comprehensive breakdown of these findings can be found 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 Tradable team and the Tradable 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	Method withdrawalToAddressRequest and withdrawalToAddressRequestForUser do not send tokens to the `receiver` address	High	Resolved
2	Unsafe uint64 casting may overflow	High	Resolved
3	Lack of proper integration with Gelato relay	High	Resolved
4	Missing onlyGelatoRelay modifier	High	Acknowledge
5	Methods related to Staking on SideVault Contract encode incorrect staking/withdrawal methods for Base chain	High	Resolved
6	No method to withdraw stuck ETH	Medium	Resolved
7	The excessive fee is not refunded to the user	Medium	Resolved
8	Division before multiplication incurs an unnecessary precision loss	Medium	Resolved
9	Transfer/TransferFrom are used instead of their counterparts from SafeERC20	Medium	Resolved
10	Contract accepts ETH unnecessarily	Medium	Resolved
11	Missing implementation of marginAccountDepositFromFundingAccount	Medium	Resolved
12	LayerZero recommendations not followed	Medium	Acknowledged
13	Centralization issue in TradableBalanceVault, TradableSettingsMessageAdapter and TradeableSettings	Medium	Acknowledged
14	Use of single step ownership transfer	Low	Resolved

#	Title	Risk	Status
15	The zroPaymentAddress is hardcoded as a zero-address	Low	Resolved
16	Missing zero amount and address checks	Low	Resolved
17	Unlimited public minting for Stable Token	Low	Acknowledged
18	Default Decimal and protocolRewardPercentageDenominator can be set to `0`	Low	Resolved
19	Lack of proper upper limits	Low	Acknowledged
20	Missing sanity checks in TradeableSettings and TradableSideVault	Low	Acknowledged
21	Missing zero address check for endpoint	Low	Resolved
22	For loop over the dynamic array	Low	Acknowledged
23	Admin user can `unstake` without fulfilling the staking period	Low	Resolved
24	Unindexed events Or Missing events	Informational	Resolved
25	Missing or Wrong Natspec comments	Informational	Acknowledged
26	Redundant storage variable update	Informational	Resolved
27	Inconsistent coding pattern in Tradable Settings	Informational	Resolved

HIGH-1 | RESOLVED

Method withdrawalToAddressRequest and withdrawalToAddressRequestForUser do not send tokens to the `receiver` address

In Contract TradableSideVault, methods withdrawalToAddressRequest and withdrawalToAddressRequestForUser are used to withdraw tokens to a `receiver` address.

These methods call the method `marginWithdrawalToAddress` in the contract TradableMarginHandler on the base chain.

Method `marginWithdrawalToAddress` further callback the side-chains side vault contract but without the `receiver` parameter. Instead, it encodes the `user` address, which means withdrawal tokens are sent to the `user` address instead of the `receiver` address.

This defeats the purpose of having methods withdrawalToAddressRequest and withdrawalToAddressRequestForUser for receiving withdrawal tokens in the receiver account.

Recommendation:

Update the method `marginWithdrawalToAddress` in the TradableMarginHandler contract to encode the `receiver` address instead of the `user` address.

HIGH-2 | RESOLVED

Unsafe uint64 casting may overflow

The _stake internal function in TradableStaking contract is responsible for updating and generating staking positions during the deposit of funds. It assigns the calculated values to the Stake struct. However, the function cast amountToBase and userShares value to uint64, which has a maximum possible value of approximately 18.5 ETH. In cases where these computed values surpass the maximal limits, a silent overflow can occur, potentially leading to inaccuracies within the protocol's accounting.

Recommendation:

Do not cast the mentioned values to uint64.

Lack of proper integration with Gelato relay

The protocol employs Gelato for gasless transactions, enabling users to send transactions without a native token balance. However, with off-chain relaying, msg.sender no longer denotes the user initiating the transaction. When relaying a message to a target smart contract function, it becomes crucial for the function to authenticate the origin of the message and verify that it was correctly forwarded by the designated relayer. To address this challenge, Gelato recommends the utilization of an ERC-2771 compliant contract. This type of contract implements data encoding to relay the original _msgSender from off-chain, ensuring that only the trusted forwarder is able to encode this value. However, the TradableSideVault contract uses GelatoRelayContext, which lacks ERC-2771 compatibility. As a result, any function with the onlyGelatoRelay modifier becomes unusable. The use of the _msgSender() function originates from the Context.sol contract which solely returns the msg.sender value. In the situation where a transaction is initiated through the Gelato relayer, invoking _msgSender() will yield the address used by the relayer to forward the transaction, rather than the original user's address

Recommendation:

Implement the Gelato's ERC-2771 complaint logic and follow the security best practices.

Missing onlyGelatoRelay modifier

In Contract TradableSideVault, there are several methods that are “ForUser” meaning those methods will be called on behalf of a user and using _msgSender() for getting the address for the user.

Although these methods are using _msgSender(), they are missing the onlyGelatoRelay modifier. It defeats the purpose of having another method with the only difference of using _msgSender().

```
function stakingAccountDepositForUser(
address token,
uint256 amount
) external {
```

```
function stakeFromMarginAccountForUser(
address token,
uint256 amount
) external {
```

```
function withdrawalRequestForUser(
WithdrawalType withdrawalType,
address token,
uint256 amount
) external {
```

Recommendation:

Add onlyGelatoRelay modifier to the above methods.

Fixed:

Acknowledged and removed these methods.

Methods related to Staking on SideVault Contract encode incorrect staking/withdrawal methods for Base chain

In Contract TradableSideVault, method stakingAccountDepositForUser(...) and method stakingAccountDeposit encodes the keyword, destination base chain contract address, and the destination method along with the parameters. The destination base chain contract is the staking vault address.

```
// message being sent
bytes memory payload = abi.encode(
"md",
dstBaseStakingVault,
abi.encodeWithSignature(
"marginAccountDeposit(address,(address,uint8,bool),uint256)",
msg.sender,
getAcceptedTokenInfo(token),
amount
)
);
```

Whereas it is encoding method `marginAccountDeposit` which is not implemented in the Staking Vault contract. This will fail to stake for the user.

Similarly, method withdrawalRequest and method withdrawalRequestForUser encodes the data in the above-mentioned logic but here the destination base chain contract could be the staking vault address (dstBaseStakingVault) based on withdrawal type.

```
// creates messages using user data and send to base chain
address destinationVault = withdrawalType == WithdrawalType.Stake
? dstBaseStakingVault
: dstBaseMarginVault;
bytes memory payload = abi.encode(
"mw",
destinationVault,
abi.encodeWithSignature(
"marginAccountWithdrawal(address,(address,uint8,bool),uint256,address)",
msg.sender,
getAcceptedTokenInfo(token),
amount,
address(this)
)
);
```

Whereas the method encoded is marginAccountWithdrawal which is again not implemented in the staking vault contract. This will fail withdrawal for users.

Recommendation:

Update the encoding to use either the correct destination base chain address or the correct method on the staking vault contract.

MEDIUM-1 | RESOLVED

No method to withdraw stuck ETH

In Contract TradableSettingsMessageAdapter, there is no method to withdraw the stuck ETH in case this contract is not used anymore.

This contract will have ETH as it will be used for paying fees to send messages on across chains using Layer Zero.

Recommendation:

Add a method similar to the method `transferGas(...)` in the TradableSideVault contract to transfer stuck ETH only for admin.

MEDIUM-2 | RESOLVED

Excessive fee is not refunded to the user

The swapLiquidity function in TradableSideVault contract is able to execute swap using the Stargate router. Users can call this function externally, supplying a fee in native tokens. However, in case of excessive fees being provided, the surplus is refunded to the contract address itself, rather than the msg.sender address.

Recommendation:

Set msg.sender address as a refund address.

MEDIUM-3 | RESOLVED

Division before multiplication incurs an unnecessary precision loss

In Solidity, performing division before multiplication can lead to a loss of precision due to integer division truncation. This means that the fractional part of the result is truncated, leading to inaccurate calculations. The calculation of the amount to the base token is performed in the following way:

```
return amount / (10**selectedToken.decimals) *
(10**ITradableSettings(settingsProvider).getDefaultDecimal());
```

or

```
uint256 amountToBase = amount / (10**selectedToken.decimals) *
(10**stakingSettings.baseDecimal);
```

Recommendation:

Avoid division before multiplication and always perform division operations at the end.

MEDIUM-4 | RESOLVED

Transfer/TransferFrom are used instead of their counterparts from SafeERC20

The ERC20.transfer(), ERC20.transferFrom() and ERC20.approve() functions return a boolean value indicating success. Some ERC20 tokens that are not compliant with the specification could return false from the transfer function call to indicate that the transfer fails, but the calling contract would not notice the failure if the return value is not checked.The EIP-20 specification requires checking the return value.

Recommendation:

Consider implementing OpenZeppelin's SafeERC20.

MEDIUM-5 | RESOLVED

Contract accepts ETH unnecessarily

Across the protocol, there are several contracts that accept ETH as it implements the following:

```
receive() external payable {}
```

Although there are several contracts that do not need ETH. The following are the two contracts:

Contract TradableBalanceVault is using TradableSettingsMessageAdapter.sendMessage(...) for sending messages across the chain so it doesn't need ETH.

Contract TradableSettings is using TradableSettingsMessageAdapter.sendMessage(...) for sending messages across the chain so it doesn't need ETH.

Recommendation:

Update the contracts to not accept ETH if not needed, as they can get stuck forever.

Fix 1: The issue was fixed partially by commit 98f10c19c1708f86fb503ff2aaa8d20b6ffc8677. It is recommended to remove both the receive() method and withdrawStuckETH(...) method from contract TradableSettings.
Since the contract TradableSettings is using TradableSettingsMessageAdapter.sendMessage(...) for sending messages across the chain, so it doesn't need ETH.

MEDIUM-6 | RESOLVED

Missing implementation of marginAccountDepositFromFundingAccount

In the SideMarginFundingAccount contract, the marginAccountDepositFromFundingAccount() function, which is called on line: 132, does not exist in the TradeableSideVault contract. This can lead to the function reverting, always leading to incorrect execution of function than intended and thus leading to Denial of Service.

Recommendation:

It is advised to define the function properly in the expected contract and review business and operational logic.

MEDIUM-7 | ACKNOWLEDGED

LayerZero recommendations not followed

In the **TradableSettingsMessageAdapter** contract, the **Layer Zero** best practices as mentioned here are not followed. It states that-
"It is highly recommended User Applications implement the ILayerZeroApplicationConfig. Implementing this interface will provide you with forceResumeReceive which, in the worse case can allow the owner/multisig to unblock the queue of messages if something unexpected happens"

It is advised that the **TradableSettingsMessageAdapter** implements this *ILayerZeroApplicationConfig*.

Also most importantly, there is missing implementation of **a nonblocking receiver** in the **TradableSettingsMessageAdapter** contract, as recommended here by Layerzero. This again helps to mitigate the issue of blocked queue of messages when using Layerzero. Also be sure to setTrustedRemote() to enable inbound communication on all contracts as recommended. Otherwise it could lead to potential Denial of Service.

Recommendation:

It is advised to follow the above Layerzero recommendations and Best practices and implement them.

Centralization issue in TradableBalanceVault, TradableSettingsMessageAdapter and TradeableSettings

In the **TradableBalanceVault** contract, the **settingsProvider** can be changed anytime by an admin using **setSettingsProvider()** function. A malicious admin can exploit it by changing the settingsProvider to a malicious contract, thereby leading to potential loss of funds of users.

Also, the same **setSettingsProvider()** exists in the **TradableSettingsMessageAdapter** contract and thus the same issue above is applicable.

Similarly, ownerContract can be changed anytime by an admin using the **changeOwnerContract()** function. Setting this to a malicious contract can lead to potential reentrancy issues and loss of funds.

The protocolRewardPercentage and protocolRewardPercentageDenominator can be changed anytime without users noticing. It could also lead to exploitation by a malicious admin, as the can change it to a very high value or very low value. This can be unfair for users if they are unable to notice these changes and unknowingly initiate transactions.

Recommendation:

It is advised to decentralize the usage of these functions by using a multisig wallet with at least 2/3 or a 3/5 configuration of trusted users. Alternatively, a secure governance mechanism can be utilized for the same.

LOW-1 | RESOLVED

Use of single step ownership transfer

The TradableSideVault and TradableSettings contracts implement the setAdminUser function, which allows changing the adminUser address. If the admin's address is set incorrectly, this could potentially result in critical functionalities becoming locked.

Recommendation:
Consider implementing a two-step pattern. Utilize OpenZeppelin's Ownable2Step contract.

Fixed: Issue fixed by commit 98f10c19c1708f86fb503ff2aaa8d20b6ffc8677. The setAdminUser function was removed from TradableSideVault. The TradableSettings contract implement Ownable2Step.

LOW-2 | RESOLVED

The zroPaymentAddress is hardcoded as a zero-address

The sendMessage function in TradableSideVault contract calculates required fees and sends a message to LayerZero endpoint. In accordance with the integration checklist it is advisable to provide *zroPaymentAddress* value as a parameter, allowing for potential future adjustments, rather than hardcoding it as a static zero address value.

Recommendation:
Pass the zroPaymentAddress value to the send function as a parameter.

LOW-3 | RESOLVED

Missing zero amount and address checks

In the **TradableBalanceVault** contract, there is a missing **non-zero** check for the **amount** parameter in the **marginAccountWithdrawal()** function. This could accidentally lead to the withdrawal of 0 amount being processed, which could lead to unnecessary transactions and gas wastage.
Also, there is a missing **zero address** check for the **receiver**. This could lead to a loss of funds for users if a zero address is accidentally used by users for the receiver parameter.
In Contract TradableSideVault, the sendMessage(...) method has a `destination` parameter which is not checked for address(0). It is advised to check if the `destination` parameter is address(0) and revert if it is.

In Contract TradableSettings.sol, the constructor sets the `adminUser` which is used all across the protocol contracts and does not validate if the given parameter _adminUser is address(0) or not. In case, the admin user is set address(0), the contract may be needed to be deployed again.

Recommendation:
It is advised to add a missing zero amount check as well as a zero address check for the mentioned methods.

LOW-4 | ACKNOWLEDGED

Unlimited public minting for Stable Token

In Contract StableToken, the method mintTo(...) is public and allows anyone to mint any amount of tokens. The contract has an owner so it is advised to allow only the owner to mint the tokens. We understand this contract is just for testing purposes so this is just a low severity issue.

Recommendation:
It is advised to refrain from using this contract in production. Also, update the mintTo(...) to be used only by admin.

LOW-5 | RESOLVED

Default Decimal and protocolRewardPercentageDenominator can be set to `0`

In Contract TradableSideVault, the method setDefaultDecimal allows the defaultDecimal value to be set `0` which will cause the method `convertAmountToTokenDecimal` or any method using defaultDecimal to return unexpected results.

In Contract TradableSettings, the method setProtocolSettings allows the owner to set protocolRewardPercentageDenominator to be 0 which will result into divide by 0 wherever it is used since it is a denominator value.

Recommendation:
Add a `require` statement to ensure that defaultDecimal and protocolRewardPercentageDenominator are not set to 0.

Fix 1: The issue was fixed partially by commit 98f10c19c1708f86fb503ff2aaa8d20b6ffc8677. It is recommended to to add the check to ensure that protocolRewardPercentageDenominator is not set to 0 as well in the method `setProtocolRewardSettings`.

LOW-6 | ACKNOWLEDGED

Lack of proper upper limits

In the Tradeable Settings contract, **protocolRewardPercentage** and **protocolRewardPercentageDenominator** should have maximum limits. Otherwise, it could accidentally lead to assigning inconsistent and arbitrary numbers that could lead to values that do not conform to the percentage calculations intended.

Recommendation:
It is advised to add maximum limits or relative limits in order to mitigate this issue.

Missing sanity checks in TradeableSettings and TradableSideVault

In the **TradeableSettings** contract, there are missing sanity checks in **setMarketMakerMarginSettings()** function for **minMarginDeposit, liquidationPeriod**, and **protocolLiquidationPercentage**.

This could lead to issues, for example, **minMarginDeposit** can be set to **0**, which would mean that one does not need to deposit anything at all for the margin account. The **liquidationPeriod** could also be set to very low values, leading to more issues.

Also, the **protocolLiquidationPercentage** could be set to a very high value leading to inconsistent and impractical percentage calculations.

In addition to this, in the **TradableSideVault** contract, there are missing sanity checks in the function **setStakingSettings()**. A missing non-zero check for **minStakingDeposit** could mean that the user can stake **0** tokens, which are logically incorrect and could introduce more issues.
It is advised to add at least a non-zero check for the **minStakingDeposit**. The same is advised for **minStakingWithdrawal.**

Recommendation:
It is advised to add proper required checks and sanity checks for all the above parameters in the **TradeableSettings** as well as the **TradableSideVault** contract.

Comments:
The team added the required sanity checks for **setMarketMakerMarginSettings()**, but decided not to add checks to the **setStakingSettings()** function to conform to the working principle of the system.

LOW-8 | RESOLVED

Missing zero address check for endpoint

In the **TradableSettingsMessageAdapter** contract, there is a missing zero address check for the endpoint in the constructor.
Similarly, in the **TradableSideVault** contract, there is a missing zero address check for the endpoint in the constructor. It is important to add a zero address check here because the endpoint can be set only once.
In addition to this, in the **TradableSideVault** contract, there is a missing zero address check for adminUser in **setAdminUser()** function.

Recommendation:
It is advised to add a proper require check for the same.

LOW-9 | ACKNOWLEDGED

For loop over the dynamic array

There exists a for loop on **line: 199** which loops over **sideVaultsList**. This can be problematic because it is a dynamic array and could lead to out of gas issues if its length increases a lot. It also exists on **line: 395** and **line: 307**. Alternatively, a **removeSideVaultsList** function can be added to remove sideVaults if the looping leads to out of gas issues. But this would introduce the risk of existing vaults with funds being removed from the list, which could lead to more issues like Denial of Service for users.

Recommendation:
It is advised to put a max limit on the sideVaultsList in order to mitigate this issue.

LOW-10 | RESOLVED

Admin user can `unstake` without fulfilling the staking period

In contract TradableStaking, the method _unstake(...) implemented the following logic

```
if (user != ITradableSettings(settingsProvider).getAdminUser()) {
uint256 timeDiff = block.timestamp - (uint256(_stakes.timestamp));
require(
timeDiff > uint256(stakingSettings.stakingPeriod),
"period"
);
// console.log("timeDiff: %d", timeDiff);
}
```

Here it means that the admin user can unstake without fulfilling the staking period.

Recommendation:
Use a multi-sig account for the admin user.

INFORMATIONAL-1 | RESOLVED

Unindexed events Or Missing events

In Contract TradableSideVault, none of the events have indexed parameters. Most user actions are initiated from this contract only so it is advised to make the `user` parameter indexed in events. For example:

```
event SideChainWithdrawalInitiated(
address user,
address token,
uint256 amount
);
```

In Contract TradableSideVault, the method deactivateAcceptedToken(...) is not emitting an event. It is advisable to emit an event for this update as well.

Recommendation:
Index the `user` parameter in Side Vault events.

Missing or Wrong Natspec comments

In Contract TradableSideVault, the constructor has several parameters but the natspec comment only describes one parameter. It is advised to add a description for all parameters.

In Contract TradableSideVault, update the natspec comment of `setDestinationConfiguration` method.

In Contract TradableSideVault, method _nonblockingLzReceive(...) have comments which mention the keywords for many actions which are possible to be called from base chain to side chain vaults. It is missing descriptions for keywords such as “ms”, “ss”, “sdd”. It is advised to add the explanation for these keywords as well.

In Contract Tradable Balance Vault, the method transferWithdrawableBalance(...) allows the admin to transfer the withdrawable balance from its own account to the receiver account. Although the comment mentions, this method should allow users the ability to transfer their balance to other users. It is advised to update the comment as per the implemented method.

In Contract TradableStaking, struct Stake has 4 variables in which 2nd and 3rd variables are 32 bytes whereasthe 4th variable size is 5 bytes contrary to comments.

Recommendation:
Update/Add the natspec comments for the above-mentioned contracts’ methods.

INFORMATIONAL-3 | RESOLVED

Redundant storage variable update

In Contract TradableSettings, the method activateTradingFeeTier is using storage for `selectedTier` variable, which updates the state of the `tradingFeeTierMap` map. There is no need for the last line in the method logic.

```
tradingFeeTierMap[price] = selectedTier;
```

It is also the same for the method `deactivateTradingFeeTier` in the TradableSettings contract.

Recommendation:
Remove the redundant state update statement.

INFORMATIONAL-4 | RESOLVED

Inconsistent coding pattern in Tradable Settings

The internal function **sendMessage() of Tradeable Settings** contract is used only in some places, while in other places the complete function is written instead of calling it internally.. It can also be used on **lines: 503, 510** and **517**

Recommendation:
It is advised to use the sendMessage internal function at all places or use the function definition so that the coding pattern is consistent.

	TradableStaking.sol TradableSideVault.sol StableToken.sol TradableMarginHandler.sol TradableMarginVault.sol TradableSettings.sol TradableSettingsMessageAdapter.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 Tradable in verifying the correctness of their contracts 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 Tradable contracts requirements for details about issuance amounts and how the system handles these.

TradableMargin

TradableMargin

setSettingsProvider

✓ should set (74ms)
✓ should revert with unauthorized (68ms)
✓ should revert with invalid-address (49ms)

setAllowedStakingProvider

✓ should set (48ms)
✓ should revert with unauthorized (69ms)
✓ should revert with invalid-address (43ms)

setVault

✓ should set (61ms)
✓ should revert with unauthorized (45ms)
✓ should revert with invalid-address (45ms)

changeUserType

✓ should revert with invalid-change (58ms)
✓ should revert with invalid-change unauthorized-caller

attemptLiquidation

✓ should update position liquidation (82ms)

volAcc: % 0

1000

✓ should update position liquidation (288ms)
✓ should revert with market-makers-only (208ms)

depositValidation

✓ should validate deposit (102ms)
✓ should revert with min-deposit (81ms)
✓ should revert with unauthorized-contract (65ms)
✓ should revert with token-not-active (54ms)

stakeFromMarginBalance
✓ should stake from margin balance (180ms)
✓ should revert with insufficient-balance (101ms)
✓ should revert with unauthorized-caller (118ms)
createMarketMakerPosition
✓ should create market maker position (152ms)
✓ Should revert with insufficient-balance (122ms)
✓ should create market maker position (155ms)
✓ should revert with market-makers-only (48ms)
✓ should revert with unauthorized-caller (65ms)
changeUserType
✓ should change user type to market maker (56ms)
marginWithdrawalToAddress
✓ marginWithdrawalToAddress (250ms)
depositIntoMarketMakerPosition
✓ should deposit into market maker position (251ms)
✓ Should revert with insufficient-balance (220ms)
✓ should revert with market-makers-only (236ms)
✓ should revert with unauthorized-caller (209ms)
withdrawFromMarketMakerPosition
volAcc: % 0
1000
✓ should wuthdraw from margin balance (533ms)
✓ market-makers-only (302ms)
✓ should revert with unauthorized-caller (283ms)
withdrawalValidation
✓ should validate withdrawal data (436ms)
✓ should revert with invalid-address (304ms)
✓ should revert with unauthorized-contract (336ms)
getMarketMakerPosition
✓ should get user (49ms)
✓ should get position (206ms)
✓ should revert with market-makers-only (260ms)
✓ should get market maker position (364ms)
✓ should revert with market-makers-only (419ms)
✓ should set settings provider (58ms)

TradableSideVault

getAdminUser
✓ should return current admin user (42ms)
setAdminUser
✓ should update admin user with new address
✓ should revert with unauthoized-caller (63ms)
✓ should revert with invalid-address

marginAccountDepositFromUserAccount
✓ should receive deposited amount (625ms)
✓ should revert with token-not-accepted (577ms)
✓ should revert with min-deposit (890ms)
✓ marginAccountDepositFromUserAccount (876ms)
setters
✓ should set destination configuration (907ms)
✓ should set default decimal
✓ should withdraw tokens (591ms)
✓ should revert with invalid-transaction (918ms)
✓ tokenExists (905ms)
✓ tokenExists (852ms)
✓ getAcceptedTokenInfo invalid address (828ms)
✓ getAcceptedTokenInfo invalid address (745ms)
✓ should set user margin settings (913ms)
✓ should set staking settings (957ms)
✓ should deactivate an accepted token (914ms)
✓ DeactivateAcceptedToken: should revert with invalid-address (859ms)
✓ ActivateAcceptedToken: should activate an accepted token (783ms)
✓ ActivateAcceptedToken: should revert with invalid-address (831ms)
✓ DeleteAcceptedToken: should delete acepted token (903ms)
✓ DeleteAcceptedToken: should revert with invalid-address (920ms)
stakingAccountDeposit
✓ should recieve staker token (994ms)
✓ should revert staking-not-allowed (890ms)
✓ should revert with token-not-accepted (799ms)
✓ should revert min-deposit (926ms)
stakingAccountDepositForUser
✓ should receive tokens and send message (996ms)
✓ should revert with staking-not-allowed (939ms)
✓ should revert with token-not-accepted (813ms)
✓ should revert with min-deposit (1019ms)
stakeFromMarginAccount
✓ should send message to messageAdapter (1004ms)
✓ stakeFromMarginAccount (967ms)
✓ should revert with token-not-accepted (864ms)
✓ should revert with min-deposit (987ms)
stakeFromMarginAccountForUser
✓ should encode and then send message (1001ms)
✓ should revert with staking-not-allowed (968ms)
✓ should revert with token-not-accepted (867ms)

✓ should revert with min-deposit (964ms)

stakeFromMarginAccountForUser

✓ should encode message and send (1028ms)

marginAccountDepositForUser

✓ should receive token and encode message (1146ms)
✓ Should revert with token reverted (1037ms)
✓ Should revert with min deposit (1069ms)
✓ Should revert with onlyGelatoRelay (1059ms)

withdrawalRequest

✓ should encode withdrawal request message and send (1238ms)
✓ token-not-accepted (1017ms)
✓ should revert with insufficient-side-vault-balance (1170ms)
✓ should revert with min-withdrawal (1215ms)

withdrawalRequestForUser

✓ should encode withdrawal request data and send message (1372ms)
✓ should revert with token-not-accepted (1070ms)
✓ should revert with insufficient-side-vault-balance (1129ms)
✓ should revert with min-withdrawal (1150ms)

withdrawalToAddressRequest

✓ should encode withdrawal request data and send a message (1240ms)
✓ should revert with token-not-accepted (1106ms)
✓ should revert with invalid-receiver (1242ms)
✓ should revert with insufficient-side-vault-balance (760ms)
✓ should revert with min-withdrawal (780ms)

withdrawalToAddressRequestForUser

✓ should encode withdrawal request data and send message to destination chain (893ms)
✓ should revert with onlyGelatoRelay (758ms)
✓ should revert with token-not-accepted (845ms)
✓ should revert with invalid-receiver (768ms)
✓ should revert with insufficient-side-vault-balance (1171ms)
✓ should revert with min-withdrawal (1249ms)

createMarketMakerPositionForUser

✓ should create market maker position (1348ms)
✓ should revert with onlyGelatoRelay (1278ms)

createMarketMakerPosition

✓ should create market maker position (1388ms)

depositIntoMarketMakerPosition

✓ should encode and send deposit message (1500ms)

depositIntoMarketMakerPositionForUser

✓ should encode and send deposit message (1443ms)
✓ should revert with onlyGelatoRelay (1402ms)

withdrawFromMarketMakerPosition

✓ should encode deposit request and send message (1518ms)

depositIntoMarketMakerPositionForUser
✓ should encode deposit request data and send message (1392ms)
withdrawFromMarketMakerPosition
✓ should encode withdrawal request data and send message (978ms)
withdrawFromMarketMakerPositionForUser
✓ should encode withdrawal request data and send messae (993ms)
✓ should revert with onlyGelatoRelay (1249ms)
transferGas
✓ should transfer gas (1151ms)
✓ should revert with unauthorized-caller (1027ms)
✓ should revert with Failed to send gas (1305ms)
nonblockingLzReceive
✓ aat (69ms)
✓ aat++ (134ms)
✓ aat- (121ms)
✓ dat (296ms)
✓ mw (225ms)
✓ ms (756ms)
✓ ss (1032ms)
✓ sdd (743ms)
✓ cd (1076ms)
✓ cd (783ms)
swapLiquidity
✓ should swap liquidity (426ms)
✓ should revert with unauthorized-caller (77ms)

TradableStaking Contract

setProtocolRewardSettings
✓ Should update settings provider (387ms)
✓ Should revert with unauthorized (120ms)
✓ Should revert with invalid-address (464ms)
✓ should set allowed margin provider (130ms)
✓ should revert with margin-provider-already-set (237ms)
✓ should revert with invalid-address (134ms)
✓ should revert with unauthorized (388ms)
mintTo
✓ mintTo (60ms)
setVault
✓ should revert with unauthorized (480ms)
✓ should revert with invalid-address (115ms)
depositValidation
✓ should revert with min-deposit (609ms)
✓ should validate deposit (361ms)

✓ should revert with unauthorized-caller (443ms)

withdrawalValidation

✓ should validate withdrawal without claimable rewards (457ms)
✓ should validate withdrawal with claimable rewards (581ms)
✓ should revert with unauthorized-caller (590ms)
✓ should revert with min-withdrawal (270ms)

getClaimableReward

✓ should return current claimable rewards (508ms)
✓ should return claimable rewards (211ms)

stakeFromMarginAccount

✓ should successfully stake from margin account (332ms)
✓ stakeFromMarginAccount (587ms)
✓ should revert with unauthorized-margin-provider (187ms)
✓ should revert with invalid-address (170ms)

Getters

✓ should get user shares (413ms)
✓ should return total shares available (57ms)
✓ should get user's stake (372ms)

updateStakingVaultWithdrawableBalance

✓ should increase staking vault Withdrawable balance (113ms)
✓ should reduce staking vault Withdrawable balance (118ms)
✓ should revert with unauthorized (126ms)
✓ should revert with unauthorized (166ms)
✓ should revert with insufficient-balance (172ms)

updateReward

✓ should update reward for user (579ms)
✓ should return if zero adress is passed (62ms)
✓ should return if cumulativeRewardPerTokenStored == 0 (847ms)

sendMessage

✓ should revert with invalid-vault-information (187ms)
✓ should send message (322ms)

Tradable Balance Vault

✓ Should be able to deploy (60ms)
✓ Should be able to set settings provider (387ms)
✓ Should be able to change owner contract (320ms)
✓ Should allow deposits to margin account (535ms)
✓ Should allow withdrawals from margin account (1043ms)
✓ Should be able to get user token balance (54ms)
✓ Should be able to activate authorized caller (268ms)
✓ Should be able to deactivate authorized caller (271ms)
✓ Should be able to update withdrawable balance (560ms)
✓ Should be able to transfer withdrawable balance (669ms)

Tradable settings

- ✓ Should be able to deploy (67ms)
- ✓ Should be able to set protocol reward settings (273ms)
- ✓ Should be able to get liquidity provider vault (60ms)
- ✓ Should be able to set liquidity provider vault (119ms)
- ✓ Should be able to set protocol reward vault (89ms)
- ✓ Should be able to set liquidity provider vault (122ms)
- ✓ Should be able to add side vaults (269ms)
- ✓ Should be able to get side vault (287ms)
- ✓ Should be able to check if a side vault exists (182ms)
- ✓ Should be able to set side vault destination configuration (355ms)
- ✓ Should be able to enable side vaults (454ms)
- ✓ Should be able to disable side vaults (436ms)
- ✓ Should be able to set trading fee tier (411ms)
- ✓ Should be able to deactivate trading fee tier (375ms)
- ✓ Should be able to activate trading fee tier (463ms)
- ✓ Should be able to get trading fee tier (444ms)
- ✓ Should be able to get trading fee tier list (195ms)
- ✓ Should be able to set staking settings (568ms)
- ✓ Should be able to get staking settings (193ms)
- ✓ Should be able to set default decimal (499ms)
- ✓ Should be able to setvault default decimal (380ms)
- ✓ Should be able to get default decimal (231ms)
- ✓ Should be able to set user margin settings (440ms)
- ✓ Should be able to get user margin settings (240ms)
- ✓ Should be able to set vault user margin settings (386ms)
- ✓ Should be able to set market maker tier (472ms)
- ✓ Should be able to getMarketMakerTierList (196ms)
- ✓ Should be able to get market maker tier (538ms)
- ✓ Should be able to deactivate market maker tier (347ms)
- ✓ Should be able to activate market maker tier (523ms)
- ✓ Should be able to set reward provider (150ms)
- ✓ Should be able to update pending reward (303ms)
- ✓ Should be able to set and get message adapter (331ms)
- ✓ Should be able to add message adapter caller (371ms)
- ✓ Should be able to remove message adapter caller (332ms)
- ✓ should be able to add accepted token (599ms)
- ✓ Should be able to get token exists (356ms)
- ✓ should be able to get accepted tokens lidt (444ms)
- ✓ should be able to get accepted token info (420ms)
- ✓ should be able to delete accepted token (584ms)
- ✓ Should be able to activate accepted token (542ms)
- ✓ Should be able to deactivate accepted token (666ms)

```
    ✓ Should be able to set admin user (210ms)
    ✓ Should be able to get admin user (103ms)
    ✓ Should be able to set market maker margin settings (140ms)
    ✓ Should be able to set fee settings (314ms)
    ✓ Should be able to set vault staking settings (349ms)
  Tradable settings Message Adapter
    ✓ Should be able to deploy (167ms)
    ✓ Should be able to set settings provider (406ms)
    ✓ Should be able to add accepted caller (336ms)
    ✓ Should be able to remove accepted caller (553ms)
    ✓ hould be able to send message (467ms)
    ✓ Should be able to  (1718ms)

    233 passing (6m)
```

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

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

zokyo