Three Sigma

Code Audit

tradable

Tradable Non-custodial perpetual exchange

Disclaimer

Code Audit

Tradable Non-custodial perpetual exchange

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

Tradable Non-custodial perpetual exchange

Table of Contents

Summary

Code Audit

Tradable Non-custodial perpetual exchange

Summary

Three Sigma audited Tradable in a 2 person week engagement. The audit was conducted from 29-05-2023 to 02-06-2023.

General Code Appreciation

After conducting the audit, the ThreeSigma auditing team deems this codebase **unready for production**. The number of critical vulnerabilities found showcases a lack of review and testing which compromises the integrity of the project.

Since fixing all the issues raised would require critical changes to the codebase we believe a second audit should be conducted after all the issues have been addressed (and tests have been added) to make sure the project is ready for deployment.

Protocol Description

Tradable is a non-custodial perpetual exchange that is developing an accessible and robust onboarding system for the internet while focusing on building a creator ecosystem for traders.

With access to assets from any blockchain, Tradable allows users to trade without the complexities of bridging from one chain to the other, trade on-demand low-cap coins, and access over $35Bn+ of stablecoin liquidity across all major blockchains. With deep un-fragmented liquidity, Tradable creates the on-chain liquidity base for traders.

Tradable also focuses on shared trade experiences where traders could share trades privately among peers, co-trade with each other, and access other reimagined features.

Scope

Code Audit

Tradable Non-custodial perpetual exchange

Scope

```
Java
── contracts
    ├── Multisig.sol
    ├── SideMarginFundingAccount.sol
    ├── StableToken.sol
    ├── TradableMarginHandler.sol
    ├── TradableMarginVault.sol
    ├── TradableSettings.sol
    ├── TradableSettingsMessageAdapter.sol
    ├── TradableSideVault.sol
    └── TradableStaking.sol
```

The review was conducted on the code present in the Tradable private repository, which contains the main contracts, interfaces, contract tests and deployment scripts. The code was frozen for review at commit 4cad9a04f44513c124629db48012ecb7f279aae6 .

Assumptions

The scope of the audit was carefully defined to include the contracts at the lowest level of the inheritance hierarchy, as these are the ones that will be deployed to the mainnet. The protocol extensively uses LayerZero and OpenZeppelin tools - these contracts were not audited since they are known resources that have already been battle-tested by the community.

Methodology

Code Audit

Tradable Non-custodial perpetual exchange

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

Tradable Non-custodial perpetual exchange

Project Dashboard

Application Summary

Name	Tradable
Commit	4cad9a04f44513c124629db48012ecb7f279aae6
Language	Solidity
Platform	Multiple chains

Engagement Summary

Timeline	29 May to 02 June, 2023
№ of Auditors	2
Review Time	2 person weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	4	0	4
High	1	0	1
Medium	4	0	4
Low	6	0	6

None	18	0	18

Category Breakdown

Suggestion	6
Documentation	1
Bug	10
Optimization	9
Good Code Practices	9

Code Maturity Evaluation

Code Audit

Tradable Non-custodial perpetual exchange

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. The codebase has a strong access control mechanism.
Arithmetic	Satisfactory. The codebase uses Solidity version >0.8.0 implementing safe arithmetic
Centralization	Moderate. The owner (a multisig) has privileges over the protocol (white-list tokens, black-list tokens, etc.) as well as control over the LayerZero authorized senders.
Code Stability	Unknown. At the time of writing this report we do not have direct access to the repo so we are not aware if the code was stable during the audit.
Upgradeability	Moderate. Even though the contracts themselves are not upgradable, there are a lot of parameters which the admin can modify, especially in the TradableSettings contract
Function Composition	Moderate. Some functions could be smaller and more efficient with a different code logic implementation. [3S-TRADE-L06]
Front-Running	Satisfactory. There are no known front-running opportunities.
Monitoring	Satisfactory. Events are generally emitted when there are state changing occurrences.
Specification	Satisfactory. Functions are generally commented and the protocol has public documentation.
Testing and Verification	Weak. The codebase severely lacks proper testing, a lot of issues raised could have been easily spotted by the developers if proper tests were included. [3S-TRADE-C01, C02, C03, C04, H01, etc.]

Findings

Code Audit

Tradable Non-custodial perpetual exchange

Findings

3S-TRADE-C01

TradableStaking: rewards accounting is corrupted

Id	3S-TRADE-C01
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Relevant Links	

Description

The TradableStaking contract uses the **`previousRewardPerToken`** mapping to keep track of the rewards-per-token, the last time each user updated their rewards. The problem is that this mapping isn't initialized whenever a staking position is created. This means that the first time a user claims rewards, the value of **`previousRewardPerToken[user]`** will be zero, so they will receive rewards as if they were the first user to stake their tokens in the contract.

This results in users getting way more rewards than they are entitled to, and the whole system to keep track of rewards will not work.

Recommendation

Initialize the **`previousRewardPerToken`** mapping to the current **`cumulativeRewardPerTokenStored`** whenever a staking position is created (this also requires updating **`cumulativeRewardPerTokenStored`**). The mapping **`previousRewardPerToken`** can also be removed and this field could be added to the Stake struct, as suggested in issue L-05. The logic for deposits, withdrawals and rewards should also be heavily tested.

Status

Acknowledged.

3S-TRADE-C02

TradableStaking: shares accounting is corrupted

Id	3S-TRADE-C02
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Relevant Links	

Description

In the current implementation of the TradableStaking contract, the staking function gives any user staking in the protocol the total amount of outstanding shares instead of the ones entitled by the user. This is due to a typo on line 174. Here, the user gets **`shares: uint64(shares)`** instead of **`shares: uint64(userShares)`**, where **`shares`** is the global variable representing the total number of outstanding shares, which gets corrupted by the typo.

This allows any staking user to withdraw and receive rewards as if they owned all the shares of the vault.

For clarification, take a look at the following code (a simplification of the logic used in the TradableStaking contract).

```
uint96 public balance;
uint64 public staked;
uint64 public shares; //total shares of staking pool at the moment
function _stake(address user, uint256 amount) internal{
    uint256 userShares = amount * shares / balance;
    balance += uint96(amount);
    staked += uint64(amount);
    shares += uint64(userShares);
    stakes[user] = Stake({
        owner: user,
        amount: uint64(amount),
        shares: uint64(shares),
        timestamp: uint32(block.timestamp)
        });
```

```
        emit StakeValidated(user, amount, shares);
    }
```

The event **`StakeValidated()`** is also emitting the wrong variable, it should be **`userShares`** instead of **`shares`**

Recommendation

On line 174, replace **`shares: uint64(shares)`** with **`shares: uint64(userShares)`**, same goes for the event on line 186. The logic for deposits, withdrawals and rewards should also be heavily tested.

Status

Acknowledged.

3S-TRADE-C03

TradableMarginHandler: can't perform liquidations

Id	3S-TRADE-C03
Classification	Critical
Severity	Critical
Likelihood	Medium
Category	Bug
Relevant Links	

Description

The **`attemptLiquidation()`** function of the TradableMarginHandler contract reverts if the position is liquidatable, leading to positions impossible to liquidate. This happens since there is an accounting underflow error on line 231 if the following condition is verified : **`marginBalance + (collateralAmount - protocolPercentOfCollateral) <= tradingFeesAccumulated`**. The issue is that this is the main condition to check if a position is liquidatable, i.e. the majority of the time, a position is liquidatable exactly because this condition is true, but this conditions also prevents liquidations since the call to **`attemptLiquidation()`** will revert.

For clarification, take a look at the following code (a simplification of the logic in the TradableMarginHandler):

```
contract test_liquidation {
    uint256 marginBalance = 100;
    uint256 collateralAmount = 200;
    uint256 protocolPercentOfCollateral = 20;
    uint256 tradingFeesAccumulated = 300;
    uint256 user_reserveBalance = 1000;


    function isPositionLiquidatable() view public returns(bool) { // returns true
        return marginBalance + (collateralAmount - protocolPercentOfCollateral) <=
tradingFeesAccumulated;
    }


    function attemptLiquidation() public { // calls revert
        if(!isPositionLiquidatable()) return;
        uint256 totalAmountOwed = tradingFeesAccumulated +
```

```
protocolPercentOfCollateral;
        user_reserveBalance += marginBalance + collateralAmount - totalAmountOwed;
//underflows
    }
}
```

Under these conditions, all calls to attemptLiquidation() will revert.

Recommendation

Use ints instead of uints, allowing for negative values to be incremented. In alternative, check if the increment to **`user_reserveBalance`** is positive or negative and perform addition or subtraction accordingly. A liquidation scenario similar to this one should also be included in the tests.

Status

Acknowledged.

3S-TRADE-C04

TradableMarginHandler: margin account withdraw message will revert on the sideVault

Id	3S-TRADE-C04
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Relevant Links	

Description

Users are unable to withdraw their tokens on a sideVault.

On the TradableMarginHandler contract, function withdrawalValidation(), line 263 states: **`sendMessage(dstVault, abi.encodePacked("mw", user, selectedToken.token, amount));`**

Here, since **`abi.encodePacked`** is used, instead of **`abi.encode`**, this message cannot be decoded by the side vault. Since the side vault is on a different chain, the call to **`withdrawalValidation()`** will be successful, but the call to **`_nonblockingLzReceive()`** on the side Vault will always revert.

Take a look at the following proof of concept:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract test_encodePacked {
    function test() external {
        address user = address(10);
        uint256 amount = 100;
        bytes memory payload = abi.encodePacked("mw", user, amount);
        (string memory messageType) = abi.decode(payload, (string)); // This
line always reverts
    }
}
```

Recommendation

use **`abi.encode`**

Status

Acknowledged.

3S-TRADE-H01

TradableSideVault: accepted tokens cannot be activated, deactivated or deleted

Id	3S-TRADE-H01
Classification	High
Severity	High
Likelihood	High
Category	Bug
Relevant Links	

Description

The current implementation of the TradableSideVault, doesn't allow the following operations:

- aat+ - activate accepted token

- aat- - deactivate accepted token

- dat - delete accepted token

This is due to a bug in the decoding of the message payload on the sideVault. In function **`_nonblockingLzReceive()`**, the payload (which contains the **`messageType`** string and the token address) is only being decoded to an address, which will be set to the first 20 bytes of the payload, a value that has nothing to do with the token address. Since the functions to activate, deactivate and delete tokens require that the token exists, these calls will revert when called through layerZero.

Take a look at the following code:

```
    function test() external {
        address token = address(this); // 0x9d83e140330758a8fF...
        bytes memory payload = abi.encode("aat", token);
        (string messageType) = abi.decode(payload, (string)); // returns "aat"
        (address token_) = abi.decode(payload, (address)); // returns
0x0000000000000000000000000000000000000060
    }
```

Here, the value for the decoded token address has nothing to do with the value that was encoded.

Recommendation

Replace `(address token_) = abi.decode(payload, (address))` with `(, address token_) = abi.decode(payload, (string, address))` and test these functions.

Status

Acknowledged.

3S-TRADE-M01

TradableStaking: Users cannot unstake all their shares

Id	3S-TRADE-M01
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Relevant Links	

Description

The **`_unstake`** function of the TradableStaking contract has the following code:

```
require(userShares < _stakes.shares, "!insufficient-shares");
bool isFullRedeem = userShares >= uint256(_stakes.shares);
if (isFullRedeem) {
    userShares = uint256(_stakes.shares);
}
```

Here, there are conflicting inequalities. Since **`userShares`** must be smaller than **`_stakes.shares`**, it can never be bigger or equal to **`_stakes.shares`**. The requirement also stops users from unstaking all their shares.

Recommendation

It should probably be:

```
require(userShares <= _stakes.shares, "!insufficient-shares");
bool isFullRedeem = userShares == uint256(_stakes.shares);
```

Note that the **`if`** is not necessary.

Status

Acknowledged.

3S-TRADE-M02

TradableSideVault.sol: users can grief sideVaults by spending their funds on LZ messages

Id	3S-TRADE-M02
Classification	Medium
Severity	High
Likelihood	Low
Category	Suggestion
Relevant Links	

Description

Any user can spam function **`marginAccountDeposit()`** on a **`sideVault`** with an amount of 0 tokens and drain the contract of ether (they would also receive the LZ gas refunded, if any). They could call this function several times within a single transaction, which would result in a cheap call to grief the vault, but a big cost to the protocol.

This would also prevent other users in the same chain from interacting with the **`sideVault`**, not allowing them to make deposits, withdrawals, etc.

Note: Before spamming this function, the griefer would need to run **`createUserFundingAccount()`** once, but at a small gas cost.

Recommendation

Set a minimum deposit amount on side vaults or make the depositor pay for the cost of sending the LayerZero message.

Status

Acknowledged.

3S-TRADE-M03

SideMarginFundingAccount: Unreachable functions changeAdminUser() and allocateFundingAccount()

Id	3S-TRADE-M03
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Relevant Links	

Description

`SideMarginFundingAccount` constructor sets the 'admin' and 'vault' to the address of the sideVault. Since the sideVault can never call **`changeAdminUser()`** or **`allocateFundingAccount()`** these functions are out of reach.

The current implementation of the SideMarginFundingAccount constructor states:

```
constructor (address _user, address _vault) {
    admin = msg.sender;
    user = _user;
    vault = _vault;
}
```

Since the sideVault deploys SideMarginFundingAccounts using the following line (line 321 in the code):

```
SideMarginFundingAccount marginAccount = new SideMarginFundingAccount(user, address(this))
```

Both the 'admin' and the 'vault' will be set to the same address, i.e. the address of the sideVault.

Recommendation

If the sideVault admin is the one supposed to call these functions, set that address as the SideMarginFundingAccount admin. If the user is the one supposed to call these functions, remove the variable 'admin' and give the user permission to call these functions. Also consider adding tests for these functions

Status

Acknowledged.

3S-TRADE-M04

TradableSettings: tokens can only be added in one chain

Id	3S-TRADE-M04
Classification	High
Severity	Medium
Likelihood	Low
Category	Bug
Relevant Links	

Description

Tokens can only be added onto a single **`sideVault`** in a different chain using the **`TradableSettings`** contract . This is due to the fact that calling function **`addAcceptedToken()`** will add the token on both the **`sideVault`** and **`TradableSettings`** contract, preventing future calls to the same function on the **`TradableSettings`** contract, since there is a requirement (**`require(!tokenExists(_token)`**) stopping multiple calls with the same token address.

For clarification, take a look at the TradableSettings implementation:

```
    function addAcceptedToken(address _token, uint8 _decimals, bool _isActive,
address vault) (...) external  {
        require(messageAdapter != address(0x0), "!invalid-message-adapter");
        require(vaultExists(vault), "!invalid-vault-information");
        require(!tokenExists(_token), "!token-already-exists");
        sendMessage(vault, abi.encode("aat", _token, _decimals, _isActive));
        acceptedTokenMap[_token] = AcceptedToken(_token, _decimals, false);
        acceptedTokenList.push(_token);
    }
    function tokenExists(address token) view public returns(bool) {
        (...)
        for(uint i = 0; i < acceptedTokenList.length; i++){
            if(acceptedTokenList[i] == token){
                return true;
            }
        }
        return false;
    }
```

Note: This is only an issue if the token has the same address in more than one chain.

Note 2: This issue is also found in the **`deleteAcceptedToken()`** function.

Recommendation

Remove the requirement: **`require(!tokenExists(_token), "!token-already-exists");`**. Also considering adding tests where multiple sideVaults on multiple side chains are used

---

Status

Acknowledged.

3S-TRADE-L01

TradableMarginHandler: rename variable to avoid confusion and to improve code readability

Id	3S-TRADE-L01
Classification	Low
Severity	Medium
Likelihood	Low
Category	Suggestion
Relevant Links	

Description

In lines 134 and 137 two low level calls are made to a "vault" contract. Since there are 3 types of vaults in the protocol (StakingVault, SideVault and MarginVault) it is confusing which vault it is referring to (adding to the fact that this is a low level call that does not specify the type of vault). The fact that the StakingVault is an instance of the MarginVault contract only further complicates this issue.

If the vault is mistakenly initialized to either a SideVault or MarginVault this function would revert since neither of them have both functions (only the TradableStaking contract). However it doesn't make much sense for the 'vault' variable in the TradableMarginHandler contract to refer to an instance of the TradableStaking contract.

Recommendation

Rename variable to avoid confusion. The variable renaming should also be done in the **`SideMarginFundingAccount`** by changing the name of the variable 'vault' address in line 20 to the respective type of vault expected. Using object calls, as suggested in issue N-18, is also heavily advised since casting the address to an Object would clarify which object (therefore contract) is being called.

Also consider renaming the MarginVault contract just Vault (since it can be staking or margin), and calling the Staking contract StakingHandler, adding some uniformity between the staking and margin parts of the protocol.

Status

Acknowledged.

3S-TRADE-L02

TradableSideVault: description of message types keywords use 'dc', however the implementation uses 'cd'

Id	3S-TRADE-L02
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Relevant Links	

Description

On the TradableSideVault, comments for the “Funds Reallocation Keywords”, line 207, mentions the following keyword “dc - set destination configuration”, this keyword is consistent with the Tradable convention of using the action initials as the messageType keyword.

The problem is that the current code implementation uses 'cd' instead of 'dc'. This can lead to a mistake in the messaging protocol in a future change.

Recommendation

Decide whether to use 'cd' or 'dc' and convert all the code (including comments) to that implementation

Status

Acknowledged.

3S-TRADE-L03

TradableMarginHandler: set values as constants

Id	3S-TRADE-L03
Classification	Low
Severity	Low
Likelihood	Medium
Category	Good Code Practices, Suggestion
Relevant Links	

Description

There are some instances where values should be set as constants.

Recommendation

In line 154 of **`TradableMarginHandler`** the value 10^4 should be set as a constant defined at the start of the contract for readability and to prevent future problems (if this ever gets changed). Same in line 213, 227.

Status

Acknowledged.

3S-TRADE-L04

SideMarginFundingAccount: event is not being emitted correctly

Id	3S-TRADE-L04
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Relevant Links	

Description

In line 75 of **`SideMarginFundingAccount`**, function **`verifyPayment()`**: event '**`PaymentVerified`**' is emitted after the "**`user = address(0x0)`**;" so the address emitted will always be the zero address

Recommendation

Change order of lines so the event is emitted correctly

Status

Acknowledged.

3S-TRADE-L05

Throughout repo: optimize structs to save gas

Id	3S-TRADE-L05
Classification	Low
Severity	Low
Likelihood	
Category	Optimization
Relevant Links	

Description

There are many instances in the code, where structs could be packed into smaller variable types, saving gas in the processes of both reading and writing them to storage.

Recommendation

TradableMarginHandler.sol:

- struct **`MarketMakerPosition`**: boolean variables could be packed with the **`user`** address or with the timestamp (converted to a uint40, for example). The address **`user`** could also just be removed from the struct since to access the struct you already need to know the user. **`MarketMakerPositions`** are supposed to be created regularly, so packing this struct seems very important. Also consider just storing a reference to the tier, and not the entire **`MarketMakingTier`** struct itself.

TradableStaking.sol

- struct **`Stake`** takes up 40 bytes, and not 32 as suggested by the comment, this means the struct will take up 2 storage spaces. Knowing this, timestamps should be stored as uint40s (since timestamps stored as uin32 will overflow in 2106). Mapping **`previousRewardPerToken`** can also be removed and a variable to hold the previous reward per token could be placed in this struct and packed into 2 words.

TradableSettings.sol

- struct **`UserMarginSettings`** only has one field, so it can just be stored as a variable

- struct **`MarketMakingTier`**: bool can be packed with uint16s

Status

Acknowledged.

3S-TRADE-L06

TradableSettings.sol: use linked lists instead of arrays

Id	3S-TRADE-L06
Classification	Low
Severity	Low
Likelihood	
Category	Optimization
Relevant Links	

Description

Solidity mappings allow for the implementation of linked lists, which are more efficient than dynamic arrays, especially if the array needs to be ordered.

For instance, to create a list with the ordered elements [1, 3, 6, 9], one can: create a **`mapping(uint => uint) private list`**, and assign the next value to the current element in the mapping, i.e. (**`list[1] = 3`**, **`list[3] = 6`** and **`list[6] = 9`**). This implementation still allows for iterating over the list, but has 3 big advantages.

1. Elements can be added or removed from the middle of the list with minimal modifications (e.g. to add the element 2, it is only required to change **`list[1] = 2`** and set **`list[2] = 3`**).

2. Accessing mapping values is cheaper than accessing dynamic array values, since for dynamic arrays there is the requirement that the index we are trying to access needs to be smaller than the array length, and this leads to 2 storage loads for each position accessed (unlike what happens in mappings).

3. To verify if an element is in the array, one can just do **`require(list[element] != 0)`** instead of iterating through the array. This requires adding a terminator to the list, e.g. **`list[9] = TERMINATOR`** in the example above, with TERMINATOR being a predefined uint used only for this role.

Note: It might also be useful to set: **`list[TERMINATOR] = 1`**, to mark the beginning of the list. In this case TERMINATOR should probably be renamed to SEPARATOR.

Recommendation

This sort of implementation should be used, for instance, on the **`TradableSettings`** contract, where **`marketMakingTierList`** should be a linked list, leading to significant optimization improvements to function **`setMarketMakerTier()`**. The same happens for the **`tradingFeeTierList`**.

Status

Acknowledged.

3S-TRADE-N01

TradableStaking.sol: readability improvement

Id	3S-TRADE-N01
Classification	None
Severity	None
Likelihood	
Category	Suggestion
Relevant Links	

Description

In the TradableStaking.sol contract, code can be re-factored so it is easier to read and cleaner.

Recommendation

To improve readability, line 318 of the TradableStaking contract: **`uint256 _pendingVaultReward = pendingRewards.pendingLiquidityProviderRewards;`** should be placed in line 304. With the **`_pendingVaultReward`** variable defined at the start of the function, it can be used in lines 307, 307 and 319, improving code readability, instead of constantly calling **`pendingRewards.pendingLiquidityProviderRewards`**

Status

Acknowledged.

3S-TRADE-N02

Multisig: Use an already established multisig implementation or state the source of the multisig used

Id	3S-TRADE-N02
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices
Relevant Links	

Description

The **`Multisig`** contract does not seem to be written by the Tradable team (the way the code is written is slightly different) but no source is given.

Recommendation

The original file where the multisig was sourced from should be stated in code for better transparency. If this code was in fact written by the Tradable team we would advise using an already established multisig contract to avoid possible issues.

Status

Acknowledged.

3S-TRADE-N03

Throughout repo: Comment fixes

Id	3S-TRADE-N03
Classification	None
Severity	None
Likelihood	
Category	Documentation
Relevant Links	

Description

Throughout the repo there are some comments with grammatical errors or that don't accurately describe the code at hand.

Recommendation

Below is a list of the issues of this type found:

TradableStaking

- line 25: The details (…) is used to (…) → The details (…) **are** used to (…)

- line 54: total size of stake struct adds to 40 not 32

- line 149: This the internal function (...) when deposit to made (...) → This **is** the internal function (...) when **a** deposit **is made** (...)

- line 150: (...) being performed on behave of → (...) being performed on **behalf** of

- line 27: (...) unstaking it just hold and verifies (...) → (...) unstaking it just **holds** and verifies (...)

- line 124: in **`depositValidation`**, the parameters "user" and "amount", have the wrong descriptions, they were just copied from the **`withdrawalValidation`** parameter descriptions

- line 301: "balanaces" → "balances"

TradableSideVault

- line 29, 243, 249, 262: "recieves" → "receives"

- line 32: "depsoit" → "deposit"

- line 156: "this message sends messages (...)" → "this **function** sends messages (...)"

- line 158, 187: "recieving" → "receiving"

- line 175: "(msg.sender will be this contract)". the msg.sender will be the address calling this contract (same happens in line 107 of TradableSettingsMessageAdapter.sol)

- line 343: "intiates" → "initiates"

TradableSettingsMessageAdapter

- line 16: "registerred" → "registered"

TradableMarginVault

- line 69: "reciever" → "receiver"

Status

Acknowledged.

3S-TRADE-N04

TradableSettingsMessageAdapter: redundant to have both receive and fallback functions in this situation

Id	3S-TRADE-N04
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices
Relevant Links	

Description

In line 55, 56 both **`receive`** and **`fallback`** functions are declared. Since both are empty there is no need to have them both.

Recommendation

`receive() external payable {}` does the job (unless a message signature is sent). Remove **`fallback`**

Status

Acknowledged.

3S-TRADE-N05

Throughout repo: error message too long and breaks convention

Id	3S-TRADE-N05
Classification	None
Severity	None
Likelihood	
Category	Optimization
Relevant Links	

Description

In line 100, the error message is too long (occupies more than one word and thus is more gas expensive) and it also breaks the already established convention (does not have the initial exclamation point).

This also happens in **`Multisig`**, line 59 and in **`TradableSideVault`**, function **`sendMessage()`**.

Recommendation

Error messages should be kept less than 32 characters.

Status

Acknowledged.

3S-TRADE-N06

SideMarginFundingAccount: libraries imported but not used

Id	3S-TRADE-N06
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices
Relevant Links	

Description

In function **`SideMarginFundingAccount`** in lines 5,7,8 the **`lzApp`**, **`String`** and **`SafeMath`** libraries are imported but never used.

Recommendation

These libraries can be safely removed.

Status

Acknowledged.

3S-TRADE-N07

TradableSettings.sol: variables are being written to storage multiple times

Id	3S-TRADE-N07
Classification	None
Severity	None
Likelihood	
Category	Optimization
Relevant Links	

Description

In the TradableSettings contract, functions **`activateMarketMakerTier()`** and **`deactivateMarketMakerTier()`**: lines 306 and 315 are not necessary as the struct has already been changed in storage.

Recommendation

Remove the redundant code. This issue of writing the same content to storage twice happens in other functions, like **`activateTradingFeeTier()`** and **`deactivateTradingFeeTier()`** (lines 406 and 415) and leads to considerable gas waste.

Status

Acknowledged.

3S-TRADE-N08

Throughout repo: array lengths should be cached in memory to save gas

Id	3S-TRADE-N08
Classification	None
Severity	None
Likelihood	
Category	Optimization
Relevant Links	

Description

When iterating over an array, the array length should be cached in memory and then used in the loop. This prevents multiple storage reads of the same variable, saving gas, since memory reads are a lot cheaper than storage reads.

Recommendation

Example:

```
for(uint i = 0; i < marketMakingTierList.length; i++) {
    (...)
}
```

Should be changed to :

```
uint256 arraylength = marketMakingTierList.length;
for(uint i = 0; i < arraylength; i++) {
    (...)
}
```

This issue is very common in the code, for example, in the TradableSettings.sol, it happens in lines: 248, 274, 281, 328, 344, 373, 381, 432, 436, 508 and 521.

Note: This assumes the array length is not changed in the loop.

Status

Acknowledged.

3S-TRADE-N09

Throughout repo: remove unused functions, events, errors and modifiers

Id	3S-TRADE-N09
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices, Optimization
Relevant Links	

Description

In several files, there are still functions, events, errors and modifiers written in the code that are not used anywhere.

Some examples:

TradableSettingsMessageAdapter.sol

- Line 58: function not used anywhere

TradableSettings.sol

- Line 134: this event is never used

TradableSideVault.sol

- Line 67, 73: modifier is never used

- Line 92: error is never used

- Line 181: function not used anywhere

TradableMarginVault.sol

- Line 35, 51: modifier is never used

Recommendation

These can be safely removed which will lower the bytecode size of the contract.

Status

Acknowledged.

3S-TRADE-N10

TradableStaking: move position of require to save on gas

Id	3S-TRADE-N10
Classification	None
Severity	None
Likelihood	
Category	Optimization
Relevant Links	

Description

In **`TradableStaking`**, on line 160, a require is made to ensure that the user is allowed to stake. This require can be positioned right after line 155 so the math in line 158 is only run in case a user can stake, and thus saving gas in situations where the user cannot stake.

Recommendation

Change the order of the mentioned lines

Status

Acknowledged.

3S-TRADE-N11

TradableStaking: remove unused input parameter

Id	3S-TRADE-N11
Classification	None
Severity	None
Likelihood	
Category	Optimization
Relevant Links	

Description

There are input parameters in some functions that are never used within that function

- In line 124, function **`depositValidation`** the argument **`caller`** is not used within the function.

- In line 197, function **`withdrawlValidation`** the argument **`dstVault`** is never used.

Recommendation

These variables can be safely removed.

Status

Acknowledged.

3S-TRADE-N12

Throughout repo: no need to initialize variables to zero

Id	3S-TRADE-N12
Classification	None
Severity	None
Likelihood	
Category	Optimization
Relevant Links	

Description

By default a value of a variable is set to 0 for uint, false for bool, address(0) for address. . . Explicitly initializing/setting it with its default value wastes gas.

Recommendation

In the constructor of **`TradableStaking`**, lines 93-95, three variables are initialized to zero. Line 247 of **`TradableSettings`** a variable is also initialized to false. There is no need to initialize these variables.

Status

Acknowledged.

3S-TRADE-N13

TradableStaking.sol and TradableSideVault.sol: OpenZeppelin SafeMath is no longer required

Id	3S-TRADE-N13
Classification	None
Severity	None
Likelihood	
Category	Suggestion
Relevant Links	math/SafeMath.sol math/SafeMath.sol#L13-L14

Description

Since solidity version 0.8 the compiler already adds the check for overflow and underflow, so the OpenZeppelin's SafeMath implementation of subtraction, multiplication and division is just '-', '*' and '/'.

Recommendation

Since regular mathematical operations are already safe, readability can be improved by removing the openZeppelin import and just using the regular characters for mathematical operations. This would also lead to a small gas saving.

Note: OpenZeppelin's SafeMath comments even state this on lines 13 and 14.

Note2: Removing this library should not have an impact on the Math.sol library since they are independent and used in different circumstances.

Status

Acknowledged.

3S-TRADE-N14

Throughout repo: overall code clean-up

Id	3S-TRADE-N14
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices
Relevant Links	

Description

Throughout the repo there are several signs of unfinished code which will negatively affect the image of the protocol.

Recommendation

The following issues should be addressed:

- Remove comments of unused code (TradableStaking: line 46, 157, 242, line 269-282; TradableSettings: line 32, 33, 159, line 173-175, etc.)

- Open TODOs (TradableStaking; line 305, 312, etc.) - address them and remove the comment

- Comment mentioning missing features to implement (TradableStaking: lines 13-19, TradableSettings: line 6-12, etc.) - either address the features missing or remove the comments and save the future features somewhere else for a future update

- Remove empty contracts (**`flat_TradableStaking.sol`**)

- Move **`StableToken.sol`** contract to test folder since it is only used in tests

- Remove commented import (TradableMarginHandler.sol: line 10)

The examples between parenthesis are just some examples of the issues mentioned. All the codebase should be reviewed by the Tradable team to address instances similar to the ones mentioned.

Status

Acknowledged.

3S-TRADE-N15

Throughout repo: Floating pragma used in several non-library contracts and use of pragma abicoder v2

Id	3S-TRADE-N15
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices
Relevant Links	Abicoder v2 info

Description

While floating pragmas make sense for libraries to allow them to be included with multiple different versions of applications, contracts should be deployed using a fixed pragma version. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

This happens in all the files.

Recommendation

Lock all contracts to **`pragma solidity 0.8.19;`**, since it's the most recent stable version.

Additionally abicoder v2 is set by default from solidity version 0.8.0 up, as stated here by the Solidity Team, so there is no need to include the line **`pragma abicoder v2;`** in several contracts as it is happening now.

Status

Acknowledged.

3S-TRADE-N16

Throughout repo: use custom errors or amend string error messages

Id	3S-TRADE-N16
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices
Relevant Links	Solidity team blogpost on custom errors

Description

Solidity allows for the implementation of native custom errors that are more gas-efficient than using a string within the "require" function to explain to the user where the call reverted.

Recommendation

For instance, this **`require`**:

```
require(userShares <= uint256(shares), "!staked");
```

could be replaced with the custom error:

```
error ErrorNotStaked();
(...)
if(userShares >  uint256(shares)){
     revert ErrorNotStaked();
}
```

Source: solidity team blogpost on custom errors

Note: This change would require reworking all errors (so that all are under the same convention), which would take a lot of work and could create bugs, as the conditions need to be negated under the new convention. Thus, we understand if you don't want to make

this change, but we still suggest changing the error convention used since it can cause confusion. For example in line 43 of **`TradableSettingsMessageAdapter`**:

```
require(!acceptedCallersMap[caller], "!caller-already-approved");
```

The exclamation point is used to indicate both a negation and just the start of an error message, which might lead to confusion since a user could read the error as "Not caller already approved".

Suggested mitigation: remove the exclamation point from the error messages.

Status

Acknowledged.

3S-TRADE-N17

Throughout repo: uniformize naming convention

Id	3S-TRADE-N17
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices
Relevant Links	Solidity style guide

Description

For better code-readability and ease to code, a naming convention should be used throughout the code-base (and maintained). There is already an attempt to use a naming convention in some code snippets, however it is not maintained in all the code. For example, in lines 34-36 of **`TradableMarginHandler.sol`** 2 of the private mappings start with an underscore but the third one does not.

Recommendation

We suggest the following convention, but others can also be used as long as it is consistent.

Variables

- use underscore as prefix for private variables (eg: _variable)

Example

```
contract foobar {
uint256 private _privateNumber;
function updatePrivateNumber(uint256 privateNumber_) external {
        _privateNumber = privateNumber_
    }
}
```

- use underscore as suffix for non-storage variables (eg: variable_)

Example

```
contract foobar {
uint256 poolShares;
function updatePoolShares(uint256 poolShares_) external {
        poolShares = poolShares_  //  State mutating.
    }
}
```

Functions

- use underscore as prefix for internal functions (eg: function _example())

We also advise following the style guide provided by Solidity, for example in the same file organize the functions by external, external view, external pure, public, internal, private (we understand that in some contracts like TradableSettings it is better to divide by "theme" and not "type").

Status

Acknowledged.

3S-TRADE-N18

Throughout repo: Object calls vs low-level calls

Id	3S-TRADE-N18
Classification	None
Severity	None
Likelihood	
Category	Suggestion
Relevant Links	

Description

Both object-calls and low-level calls achieve the same goal, however, object-calls are easier to implement (since they don't require the explicit function signature), safer (since they already require that the call succeeds) and less error prone (since they raise an error, if, for instance, the number or type of parameters doesn't match the implementation).

Recommendation

Use object calls whenever possible.

Status

Acknowledged.