IDEX A-1

Security Audit

July 21st, 2023
Version 1.0.0

Presented by 0xMacro

Table of Contents

Introduction

This document includes the results of the security audit for IDEX's smart contract code as found in the section titled ‘Source Code’. The security audit was performed by the Macro security team from March 28, 2023 to April 26, 2023.

The purpose of this audit is to review the source code of certain IDEX Solidity contracts, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While Macro’s review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Overall Assessment

The following is an aggregation of issues found by the Macro Audit team:

Severity	Count	Acknowledged	Won't Do	Addressed
Medium	8	-	2	6
Low	4	-	-	4
Code Quality	5	2	1	2
Informational	2	1	-	1

IDEX was quick to respond to these issues.

Specification

Our understanding of the specification was based on the following sources:

- Discussions on Telegram/Slack with the IDEX team.
- Available documentation in the repository.

Trust Model, Assumptions, and Accepted Risks (TMAAR)

IDEX's v4 protocol is built upon a hybrid architecture, where the trading engine operates off-chain for allowing low-latency trades while providing on-chain order settlement and custody. Due to this hybrid design choice, a lot of functions are restricted and can only be accessed by privileged roles:

- **owner**: The owner can change the admin without any delay.
- **admin:** The admin can change various exchange settings with immediate effect while other more sensitive ones are subject to governance delay. A detailed description about admin's privileges can be found in the Governance doc.
- **dispatcher:** Trade settlements as well as liquidating and deleveraging operations can only be initiated from an authorized dispatcher wallet. The admin can change the dispatcher wallet with no delay.

While the high number of restricted functions impose a high-level of centralization, users need to sign trades, transfers, and withdrawals so that no funds can be moved without the users permission (except for liquidating and depositing operations). In addition, users can always claim there funds in a permission-less way via the "wallet exit" mechanism. Note that users will usually receive less funds when closing the wallet via "wallet exit" as opposed to closing open positions via normal trades and then withdrawing the funds via IDEX's exchange. This is needed to ensure the solvency of the protocol.

Source Code

The following source code was reviewed during the audit:

- **Repository:** idex-contracts-ikon
- **Commit Hash:** `94726de9d487e1e34fff62c2436bf23ebc9fbad8`

Specifically, we audited the following contracts within this repository:

Contract	SHA256
contracts/Custodian.sol	`789716647791632c8b8faa55813320ae58dbd322a15d49d8b08f8d6a00168051`
contracts/Exchange.sol	`646a2d95a074eccfa0b88ff1388c4b9c520fcdba447b2622f1643ba673568b69`
contracts/Governance.sol	`2ed058275ab1df04d1dafcbf76a5f2f88bd0f5e33486e29c6788c165786f35a7`
contracts/Owned.sol	`4e4fb4ab981d4c8571fb8843c4fd098f8b464bef9a4dc24438490fb8855e204c`
contracts/libraries/AssetUnitConversions.sol	`fcd0417d6b4c3df05633231659c44f0b5e797d02b5574e6daff457a14ddef1ba`
contracts/libraries/BalanceTracking.sol	`90da9148a327d4f72edaaca09435c1587659dcd63caf848c6ed400aa9abc8d3b`
contracts/libraries/ClosureDeleveraging.sol	`f6584ec655f35e1fa666a4e4647782c381731548e549adc05c9ffaadd16a6b30`
	`b852eb25205374d81f98ac5c8cc57e3`

contracts/libraries/Constants.sol	1209e89baff27269925df447014ffdfe9
contracts/libraries/Depositing.sol	854c3004ca434c21566f4ffb0bbd8292a495e801289cbdf39b095355f6a65037
contracts/libraries/Enums.sol	cae7cb57aab1169cc6247560776f2b0e91f6cdcbcb26d5cbb19a48b0b62fb8f7
contracts/libraries/ExitFund.sol	be00b433309b838f55e131cbf37f71135a9cf94ed190ec8da39979a97611608a
contracts/libraries/Funding.sol	429a917d3919f5d5bcb32c7b633fdd6b8c654efacde2b3f050242964c3268c06
contracts/libraries/FundingMultiplierQuartetHelper.sol	4d6a3933fef07b7fb8307688530b1da0a4f333b30c070ab13ab98aa61ddddaf4
contracts/libraries/Hashing.sol	0a5184390f9fa6677303d4df1c4c1485a48b49eba3a7e0b3ed113ba469a857c7
contracts/libraries/IndexPriceMargin.sol	3b1a3b97d325120b8201c2dd3e9e71496d5461f0a014e01846b836cae2cd433d
contracts/libraries/Interfaces.sol	bff1692b2c72cf6ce937d5ae2acdd8fdd174264b88c284b8f329ff3f03de6b68
contracts/libraries/LiquidationValidations.sol	c4d774aaddcca4107330f49b41d1f77481835c57bcd8386278e06c6ccc9d8aa8
contracts/libraries/MarketAdmin.sol	2c72bcbaa7dec14f3a723c3a774586038a82ad4236716feb4218c210faff6c73
contracts/libraries/MarketHelper.sol	68a09f8d5a0b5e0ed4529b3b0219aa6678a162fb08c67fbdd35467ed33eab93b

contracts/libraries/Math.sol	41348de8c5f43536c5ff18921ffa691348763f87b65176c128ff6b8a8755e21b
contracts/libraries/NonceInvalidations.sol	c191ac34cda19f3407a659ad89002e1cf75adce47ce8b77203b698fcb5217e89
contracts/libraries/OraclePriceMargin.sol	81243e5d058fa6a850228d6493376c49a80e1fad5d418f8b31aeb6254b1745dc
contracts/libraries/PositionBelowMinimumLiquidation.sol	3437792652eb508ac3bf27bfc53be51ba2a6ac7a649d5776ad2c399013fd5b8e
contracts/libraries/PositionInDeactivatedMarketLiquidation.sol	72174221abf251e20187bec0af859483e23fcbbe75595e083c689f70546d5671
contracts/libraries/SortedStringSet.sol	f7215fb5d542142a853bb60566474e348fd79575a5bc410bb6800bfb1ab62c91
contracts/libraries/String.sol	b4d9d0589df933d3b60652e2911e39c5caef981437f1686c58b4c9ba8ce91fa4
contracts/libraries/Structs.sol	29b86950c0d9a7240b4372320082362ec1a8aa97b9ca9cfc5fe3e7e9874366c5
contracts/libraries/Time.sol	aa67ad4149430e42f61ab59139621d24c17e8a205e58cdddb724a756cd4e893d
contracts/libraries/TradeValidations.sol	d43d50ac738401a2ca301e44d1614168a23b34485f246aee2dfd3ae07e2a9d36
contracts/libraries/Trading.sol	fb36be336126ed7f836e128058258538db8351d9d4bab1d1fe7ad2d5c3901ba8
contracts/libraries/Transferring.sol	a9dea4472d05bba812b686e7e773d41bbc8de8b6f843b53c83b9aef0b937dd

	48
contracts/libraries/UUID.sol	4a4cd7d0c15b380be066f520f75f57c49a2b4d8094466dc6e407609dede0d999
contracts/libraries/Validations.sol	03b6e3d290cecdc50860d3dc7e8b88a91ba332d5648fdc0a093ee2e88986fd29
contracts/libraries/WalletExitAcquisitionDeleveraging.sol	e48e47a773089f04ab256e228c8c9f472823a3f1c88d5472df4875364004dd56
contracts/libraries/WalletExitLiquidation.sol	afda31e46fc89374ad0f91111ee8732c00877c80c5f0a60dd813df2a07cb9b35
contracts/libraries/WalletExits.sol	d87c3059e2cf144411da55fd52b28d0d1408fd08d36fd9fb385f94777d8ba32a
contracts/libraries/WalletInMaintenanceAcquisitionDeleveraging.sol	666eb21534b5d8c9df5d1c0bd32e1d9c9123164f799737033301a1f127a4dd1b
contracts/libraries/WalletInMaintenanceLiquidation.sol	061e524d17fd2b2a1024be458ab08fe277bb6796cbc28b3c9a35c81b067eeae2
contracts/libraries/Withdrawing.sol	4c7f74828681d53f67a1585a53221a3465ae4f386e0fffc7d45b1f559e2d2814
contracts/bridge-adapters/ExchangeStargateAdapter.sol	cd254a80b33acca329f676a873171df6db3abcb674e7e466171116695deb5048

Note: This document contains an audit solely of the Solidity contracts listed above. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Issue Descriptions and Recommendations

Click on an issue to jump to it, or scroll down to see them all.

Security Level Reference

We quantify issues in three parts:

1. The high/medium/low/spec-breaking **impact** of the issue:
    - How bad things can get (for a vulnerability)
    - The significance of an improvement (for a code quality issue)
    - The amount of gas saved (for a gas optimization)
2. The high/medium/low **likelihood** of the issue:
    - How likely is the issue to occur (for a vulnerability)
3. The overall critical/high/medium/low **severity** of the issue.

This third part – the severity level – is a summary of how much consideration the client should give to fixing the issue. We assign severity according to the table of guidelines below:

Severity	Description
(C-x) Critical	We recommend the client **must** fix the issue, no matter what, because not fixing would mean **significant funds/assets WILL be lost.**
(H-x) High	We recommend the client **must** address the issue, no matter what, because not fixing would be very bad, *or* some funds/assets will be lost, *or* the code's behavior is against the provided spec.
(M-x) Medium	We recommend the client to **seriously consider** fixing the issue, as the implications of not fixing the issue are severe enough to impact the project significantly, albiet not in an existential manner.
(L-x) Low	The risk is small, unlikely, or may not relevant to the project in a meaningful way. Whether or not the project wants to develop a fix is up to the goals and needs of the project.
(Q-x) Code Quality	The issue identified does not pose any obvious risk, but fixing could improve overall code quality, on-chain composability, developer ergonomics, or even certain aspects of protocol design.
(I-x) Informational	Warnings and things to keep in mind when operating the protocol. No immediate action required.
(G-x) Gas Optimizations	The presented optimization suggestion would save an amount of gas significant enough, in our opinion, to be worth the development cost of implementing it.

Issue Details

~~M-1~~ Fee-on-transfer tokens not supported for quote token

TOPIC	STATUS	IMPACT	LIKELIHOOD
Coding Standards	Fixed	High	Low

Description

The `deposit` function in *Exchange.sol* does not account for tokens applying transfer tax, which results in less value transferred to the custodian than requested. However, the balance stored in `BalanceTracking` is using the full amount without considering the fees.

In the current version of IDEX v4, only USDC is supported as quote token. The USDC token contract implements upgradable logic, thus there is no guarantee that USDC will not add fee-on-transfer logic in the future.

Remediations to Consider

Consider handling fee-on-transfer tokens correctly by calculating `balanceAfter – balanceBefore` when collateral is deposited to custodian.

M-2 Oracle price can be outdated

TOPIC	STATUS	IMPACT	LIKELIHOOD
Incentive Design	Wont Do	High	Low

Description

The call to Chainlink's price feed using `latestRoundData()` can return stale data in extreme situations such as highly volatile market conditions and flash crashes. In such situations, users could `withdrawExit` their wallet and benefit from outdated price data.

Remediations to Consider

Consider checking that the returned `updatedAt` timestamp - as referenced in the Chainlink docs - falls within a valid time period.

RESPONSE BY IDEX

> withdrawExit does not validate oracle price update timing by design.
>
> 1. The protocol is not sensitive to stale oracle pricing on exit withdrawals. While exiting wallets may receive out-of-date pricing when liquidating positions, the protocol is not at risk of insolvency due to exit pricing.
> 2. Preventing exit withdrawals due to oracle price update timing validation introduces a source of withdrawal censorship.

~~M-3~~ Delay calculations lead to shorter delays than specified

TOPIC	STATUS	IMPACT	LIKELIHOOD
Incentive Design	Fixed	High	Low

Description

The exchange uses hard-coded parameters listed in the GOVERNANCE readme.

Some of the parameters - defined in *Constants.sol* - are subject to block timing:

```
// 1 week at 3s/block
uint256 public constant EXIT_FUND_WITHDRAW_DELAY_IN_BLOCKS = (7 * 24 *

// 1 day at 3s/block
uint256 public constant FIELD_UPGRADE_DELAY_IN_BLOCKS = (1 * 24 * 60 *

// 1 week at 3s/block
uint256 public constant MAX_CHAIN_PROPAGATION_PERIOD_IN_BLOCKS = (7 * 2
```

The division by 3 is derived from the average block time on Polygon, which is slightly higher than 2 seconds. The intention was to calculate delays so that they are at least the specified amount or longer (depending on block time) rather than always falling short.

However, this is not the case with the current implementation - as they are always falling short. Let's consider the parameter `FIELD_UPGRADE_DELAY_IN_BLOCKS` that is used for delaying upgrades to critical configuration settings such as changing the insurance wallet or market settings:

- it is calculated as: `(1 * 24 * 60 * 60) / 3`, resulting in `28800` blocks that need to pass for finalizing the change.
- Using the average block time of 2 seconds (for Polygon), the change can be settled after `28000*2` seconds, resulting in `15.5 hours` (instead of the actual 24 hours).

This behavior of delays falling shorter than specified, can have quite severe implications for users, as they might e.g. want to close their positions before any change comes into effect.

Remediations to Consider

Consider calculating above block delays by dividing by a value that is at maximum the average block time, preferable a bit smaller.

RESPONSE BY IDEX

> The targeted deployment chain does not have stable block times. As a

> result, we switched any logic that uses block numbers to timestamps.

~~M-4~~ Traders have no control over incurring fees

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Addressed	High	Medium

Description

For `trades`, `withdrawals`, and `transfers`, traders need to pay fees of up to 20% of the amount being moved. These fees are charged by IDEX to cover the incurred gas costs.

Despite this limit of 20%, traders have no control over the amount of fees they are willing to pay.

Hence, when traders perform one of the above operations, it is not transparent to them how much fees they gonna pay, and they can end up loosing 20% of the amount being moved.

Especially for withdrawals and transfers, where on-chain settlement happens almost immediately, the amount of gas required for the transaction can be predicted quite reliable. For such cases, traders want to have the transparency and control over the expected fees.

Remediations to Consider

Consider one of the following remediations:

1. To give control over incurred fees back to traders, consider adding a field such as `predictedFee` amount to the signature message, so that traders can give their consent over paid fees when signing the message. The actual amount being paid on fees must then match the signed `predictedFee`

amount, considering a certain tolerance.

2. While the above remediation step adds quite some complexity, as a quicker solution it could be also considered to significantly lower the maximum allowed fee value.

RESPONSE BY IDEX

> Fees are considerably more complex in Ikon than in many dapp trading protocols, such as swap AMMs. For example, fees vary between makers and takers, can be negative for promotional reasons, include variable gas, and can change over the course of trade executions against a single placed order. As such, providing a reasonable predictedFee is not straightforward and is a source of complexity we would like to avoid. Notably, the value of the fee cap is largely driven by gas concerns. In prior releases, the fee cap was configured at 20% to allow for gas collection during price spikes while maintaining low trade minimums. Of course, it is in the interest of neither the user nor the exchange to lose 20% of trade value from gas. Recent developments in blockchain deployment target performance indicate that settlement gas may be negligible for this release. In that case, we would lower that value, likely to something in the 1% to 5% range.
>
> ### More context on the fix change:
>
> The fee cap is updated to 5% and may be lowered based on the gas characteristics of the production deployment chain.

~~M-5~~ Missing validation for index price

TOPIC	STATUS	IMPACT	LIKELIHOOD
Incentive Design	Addressed	High	Low

Description

In `MarketAdmin.publishIndexPrices_delegatecall` , there is no sanity check for

the passed index price. Propagating an incorrect index price can have severe consequences on all the other operations relying on the index price.

Note that the likelihood of providing an incorrect index price is considered low, as IDEX stated that they take several precautions in off-chain components to mitigate this case. However, the possibility of a compromised service wallet or a logic bug in off-chain components is not negligible.

Remediations to Consider

Consider checking the provided index price is valid by e.g. checking against oracle price if the index price is too far off from previous value.

RESPONSE BY IDEX

> We deeply considered potential validations for index pricing and ultimately decided against their inclusion. Index prices are a critical system input for a perpetuals exchange, and accuracy, timing, and update frequency are all important requirements.
>
> One potential validation scheme limits the rate of change of index prices over time. Crypto markets are volatile, of course, so picking limits that provide practical protection while allowing the necessary inputs during extraordinary events is a challenge. Further complicating the issue is Ikon's lazy index price publishing system. If no activity happens in a market, and thus no price updates are published for an extended period of time, what change caps should apply?
>
> Another potential scheme compares published index prices to another source, such as an on-chain oracle. Oracles are not perfect, however, and can themselves be wrong or delayed, likely under the very extraordinary market conditions when it's most critical to apply up-to-date index pricing. Oracle price comparisons are also complicated by Ikon's serialized dispatch model, both for out-of-date and up-to-date settlements. If dispatch is delayed, for example due to RPC issues, and settlement occurs an hour after trading, index prices from the time must be compared against matching historical oracle pricing. When dispatch is running normally, off-chain systems must accept propagation delays in on-chain pricing when deciding whether to accept a new index price update. Any conflict resulting from propagation time could interrupt operations.

> While it is possible to pick an on-chain validation scheme, in our analysis such schemes introduce operational tradeoffs that can themselves increase risk for traders. Our design combines baseline validations, for example replay protection, while focusing engineering efforts in making index price collection as reliable, performant and secure as possible off chain.

More context on the fix change:

> We modularized Ikon's index price validation systems in an effort to diversify index prices sources. As a result, Ikon supports independent index price services with protocol-level validations, starting with Pyth Network. First party index prices are still supported for redundancy, subject to governance.

~~M-6~~ Missing validation for funding rate

TOPIC	STATUS	IMPACT	LIKELIHOOD
Incentive Design	Fixed	High	Low

Description

In `Funding.publishFundingMultiplier_delegatecall` , there is no sanity check for the passed funding rate. An incorrect provided funding rate can have severe consequences on the funding payments, which can lead to wallets ending up with high negative balance or others that receive much more in payments than intended.

Remediations to Consider

Consider adding a minimum and maximum boundary to the funding rate.

RESPONSE BY IDEX

> Funding rates are clamped to 75% of the maintenance margin ratio of a

> market in our off-chain systems. Validating that funding rates are less than the maintenance margin ratio of a market also makes sense as a contract validation.

M-7 Missing reasonable limits for market fields

TOPIC	STATUS	IMPACT	LIKELIHOOD
Incentive Design	Wont Do	High	Low

Description

`Validations.validateOverridableMarketFields` lacks certain validation checks allowing to create and modify markets with undesirable behavior. For instance, markets could be created with `maintenanceMarginFraction` > 100%, making it possible to liquidate all positions.

Remediations to Consider

Consider adding the following validation checks:

- upper limit for `initialMarginFraction` and `maintenanceMarginFraction` (e.g. ≤ 100%)
- upper limit for `incrementalInitialMarginFraction`
- reasonable upper limit for `minimumPositionSize` (currently it is set to `uint64(type(int64).max - 1))`)

RESPONSE BY IDEX

> We considered adding more hard limits to market fields but chose the implemented governance protections instead. Putting aside legitimate scenarios where it may make sense to have 100% or even higher margin requirements, any change to these settings must be published on chain

> significantly before taking effect. Even if there were validation capping the maintenance margin fraction at say 50%, an attacker changing a market with a 5% mmr to a 50% mmr could have a significant adverse effect on many users. The upgrade governance mechanism, however, significantly reduces such risk.

~~M-8~~ Outstanding funding payments are ignored when calculating maintenance requirements for IF wallet

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Fixed	Medium	Low

Description

For deleveraging “In Maintenance Acquisition”, the documentation states the following:

> Validations confirm that the insurance fund cannot liquidate the wallet in maintenance via a standard Wallet In Maintenance liquidation.

However, when checking if the IF wallet can liquidate the wallet, the outstanding funding payments for the IF wallet are not taken into account. Thus, it becomes possible under specific circumstances, that a wallet is getting deleveraged via `deleverageInMaintenanceAcquisition` while standard liquidation via the IF wallet would have been possible.

The same issue is present with the `deleverageExitAcquisition` call.

While the likelihood of this is low (see response by IDEX), it breaks one of the protocols invariants and can lead to undesirable outcome.

Remediations to Consider

Consider taking outstanding funding payments into account when validating that the insurance fund cannot acquire a position.

~~L-1~~ Lack of event emission for critical, state-changing functions

TOPIC	STATUS	IMPACT	LIKELIHOOD
Best Practice	Fixed	Low	High

Description

The following state-changing functions in *Exchange.sol* don't emit events:

- `setCustodian`
- `setDepositIndex`
- `setDispatcher` , `removeDispatcher`
- `addMarket` , `activateMarket` , `deactivateMarket`
- `unsetMarketOverridesForWallet`
- `skim`

In addition, all the liquidation and deleverage functions don't emit any events, which we assume is intentional to save gas costs for expected exchange activities.

Remediations to Consider

Consider adding events for above functions.

~~L-2~~ Maximum allowed value for chain propagation period makes nonce invalidation ineffective.

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Fixed	Medium	Low

Description

The maximum value for the chain propagation period (`MAX_CHAIN_PROPAGATION_PERIOD_IN_BLOCKS`) is defined with `7 days` . Especially for nonce invalidation - which should protect against “canceled-order submission attacks” - using a high value for the `chainPropagationPeriodInBlocks` parameter makes nonce invalidation rather ineffective.

Remediations to Consider

Consider lowering the maximum allowed `chainPropagationPeriodInBlocks` to a much smaller value.

~~L-3~~ Missing `chainId` in signature

TOPIC	STATUS	IMPACT	LIKELIHOOD
Use Cases	Fixed	High	Low

Description

The signature generation doesn’t include `chainId` parameter. This makes the protocol susceptible to replay attacks when the protocol is deployed to multiple chains or when a hard fork of the chain happens.

Remediations to Consider

Consider including the `chainId` to the signature message.

RESPONSE BY IDEX

> Rather than using our own message format for signature creation, we decided to use **EIP712** signature scheme (which protects against above attack vector).

~~L-4~~ Possible to `withdrawExit` from the IF wallet

TOPIC	STATUS	IMPACT	LIKELIHOOD
Trust Model	Fixed	High	Low

Description

Insurance fund is an important instrument in perpetual swaps to keep the exchange solvent whilst trading with leverage. In doing so, the insurance fund accumulates value over time under normal market conditions.

The insurance fund has the ability to close open positions at the market price via order book trades. However, under no circumstances should the insurance fund be able to `withdrawExit` and thus close its positions against the `exitFund`. This is guaranteed by the following check in Withdrawing.sol#L59:

```
require(wallet != insuranceFundWallet, "Cannot exit IF");
```

Consider the following scenario where an admin could run the following steps making it possible to `withdrawExit` from an IF wallet:

1. Admin uses a new wallet address and calls `exitWallet()` setting this new wallet to exit status.

2. Admin can use the respective Governance functions to set the above wallet to become the new insurance fund wallet.

Following the above steps, an admin could now `withdrawExit` from insurance fund wallet.

Note that this attack vector is considered as low likelihood, however it needs to be taken into consideration that this can potentially damage the trustworthiness of the protocol.

Remediations to Consider

Consider checking that the new insurance fund wallet hasn't been exited before.

RESPONSE BY IDEX

> Preventing the insurance fund wallet from exiting is primarily an operational concern. An exited insurance fund wallet cannot participate in trade settlements to close positions, and the protocol assumes that the insurance fund wallet itself acquires the positions of an exited wallet during normal operations. While the outlined scenario is possible, it does not provide the attacker any profit angle due to exit pricing. That said, agreed that validating that the new insurance fund wallet is not exited in Governance.initiateInsuranceFundWalletUpgrade and Governance.finalizeInsuranceFundWalletUpgrade provides additional operational assurances.

Q-1 Use custom errors

TOPIC	STATUS	QUALITY IMPACT
Best Practice	Acknowledged	Low

It is advised to revert with custom errors over using require statements. Using

them saves bytecode on deployment, a little gas on execution, and allows for more detailed error messages with the ability to pass in parameters.

Consider replacing require statements with custom errors to give more context about errors and save a bit of gas.

~~Q-2~~ Use `safeTransfer` / `safeTransferFrom`

TOPIC	STATUS	QUALITY IMPACT
Best Practice	Addressed Addressed	Low

For deposits and withdrawals, `transfer` and `transferFrom` are used. It is advised to check return values for `transfer` and `transferFrom` or to use OpenZeppelin's `safeTransfer` / `safeTransferFrom` .

RESPONSE BY IDEX

> Return value checks are not necessary due to the balance checks implemented.

Q-3 Missing zero-check in `withdrawNativeAsset`

TOPIC	STATUS	QUALITY IMPACT
Best Practice	Acknowledged	Low

In `ExchangeStargateAdapter.withdrawNativeAsset` , there is no check if the provided destination wallet is address(0).

Consider adding a check `destinationWallet ≠ address(0)` to ensure no funds are lost by sending it to address(0).

Q-4 Newly added markets use current timestamp rather than oracle timestamp

TOPIC	STATUS	QUALITY IMPACT
Protocol Design	Wont Do	Low

When a new market is added in `MarketAdmin.addMarket_delegatecall` , the current timestamp is used for setting `lastIndexPriceTimestampinMs` . While this doesn't impose any immediate security risks, it would be more accurate to use the `updatedAt` timestamp returned from Chainlink's `latestRoundData` call (see here).

RESPONSE BY IDEX

> Market.lastIndexPriceTimestampinMs prevents the publishing of old index prices. The current timestamp provides a consistent up-to-date value, unlike oracle providers which may be out of date.

~~Q-5~~ Create signatures according to EIP-712 standard

TOPIC	STATUS	QUALITY IMPACT
Protocol Design	Fixed	Medium

Moving funds via trades, withdrawals, and transfers, requires a valid signature from the trader. Currently, the protocol uses its own message format for

signature creation.

Consider supporting signature creation according to the **EIP712** standard for better security and usability.

~~I-1~~ `sgReceive` can be called by anyone

TOPIC	STATUS	IMPACT
Protocol Design	Fixed	Informational ✳

The `sgReceive` function in `ExchangeStargateAdapter.sol` is not restricted and can be called by anyone. If USDC tokens are left in the contract, anyone can call `sgReceive` to deposit the leftover tokens to their own wallet. However, this can be a desired behavior and doesn't impose any security risks. Additionally, all the recommendations are met that are expected from a permissionless `sgReceive` function.

I-2 Only USDC is supported as collateral

TOPIC	STATUS	IMPACT
Protocol Design	Acknowledged	Informational ✳

As stated in the documentation, the current version supports USDC as collateral only. Supporting any other collateral assets than USDC would require an upgrade of the Exchange contract.
USDC is supposed to be pegged to USD and usually variations in price between USDC and USD are small, but - as recent events have shown - it is never

guaranteed that USDC doesn’t get depegged from USD and drops in value (see here).

RESPONSE BY IDEX

> Ikon supports any ERC-20 token as a collateral asset, but the collateral asset cannot be changed after deployment without an upgrade.

Disclaimer

Macro makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Macro specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Macro will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand by any other party. In no event will Macro be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Macro has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the IDEX team and only the source code Macro notes as being within the scope of Macro's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Macro. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Macro is not responsible for the content or operation of such websites, and that Macro shall have no liability to your or any other person or entity for the use of third party websites. Macro assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.