macro

Kwenta A-4

Security Audit

May 4, 2023
Version 1.0.0

Presented by 0xMacro

Table of Contents

Introduction

This document includes the results of the security audit for Kwenta's smart contract code as found in the section titled ‘Source Code’. The security audit was performed by the Macro security team from April 4, 2023 to April 21, 2023.

The purpose of this audit is to review the source code of certain Kwenta 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	3	-	2	1
Low	3	-	-	3
Code Quality	8	-	1	7
Gas Optimization	3	-	-	3

Kwenta was quick to respond to these issues.

Specification

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

- Discussions on Discord with the Kwenta team.
- KIP-18
- KIP-19
- KIP-55

Source Code

The following source code was reviewed during the audit:

- **Repository:** smart-margin
- **Commit Hash:** `bb6f99632ef8eda46f1eae4e67819b854955c13a`

Specifically, we audited the following contracts within this repository:

Contract	SHA256
src/Account.sol	`e71ccaa1d4a1feebd135103921449b69fe12b305ccac21482d73077d5e7558f4`
src/AccountProxy.sol	`bca54d6d95501325240281b19e47094a83ab3fa9505796f220bd83eb2aae9174`
src/Events.sol	`baac416359900d7c9233ecb6214236d70c013cfcf0f2a05f82d0efe913eb9737`
src/Factory.sol	`f9473f9beecc69681a021be796f95d2e8c726688a6bd0744164eb8fbbb33c30f`
src/Settings.sol	`1cd03cf82dd372c992cff4e84719fedded6576a21534803130e7c43b6d45068d`
src/utils/Auth.sol	`616a63517dbc3254825d4565e08a0068cd73b56ebcea75bff4022e9b695eabd6`
src/utils/OpsReady.sol	`8addf1216923862b85039a194540ee68b5f9c660120711f2f90147af14636df6`

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~~ Delegate can send an account's funds to the treasury

TOPIC	STATUS	IMPACT	LIKELIHOOD
Griefing	Fixed	High	Low

In Account.sol, a set delegate is able to execute multiple commands using `execute()` and `_dispatch()`. In these calls, parameters are passed in to make the proper execution.

However, the `market` contract address that the call is made on is also passed in as a parameter, and no checks are made to ensure this is a trusted contract.

Although there is not much a malicious contract can do to directly affect the account, it can be setup to return an assetPrice that sets the execution fee to be the amount of sUSD held in the account, draining it and sending to the set treasury address.

In most cases, this would be a griefing attack as they don't have control over the treasury wallet, but anyone that does could steal all funds held in accounts that have been delegated to them.

Remediations To Consider

Instead of having the `market` address as a parameter, use a marketKey, and query the address from the addressResolver using `_getPerpsV2Market()`. This will ensure calls are made to an actual market contract preventing malicious contract calls.

RESPONSE BY KWENTA

> Regarding the suggestion to use a market key instead of the market address, it was decided to leave it as is.

M-2 Delegates not reset when Account ownership is transferred

TOPIC	STATUS	IMPACT	LIKELIHOOD
Trust Model	Wont Do	High	Low

In Account.sol, an owner can add one or more delegates who are authorized to perform sensitive operations related to that account.

When Account ownership is transferred to a new owner, the new owner may not be aware that previously configured delegates are still set and can execute actions for the account.

This may lead to situations where the previous account owner has backdoor access to the new owner's account.

Remediations To Consider

- Remove all configured delegates when Account ownership is transferred, and
- Provide additional functionality for enumerating configured delegates.

RESPONSE BY KWENTA

> If ownership is canceled, the user should remove delegates if necessary. in most cases, I'd imagine ownership transfer is to another controlled EOA that would want the same functionality the delegates are already providing.

M-3 Account may get locked due to error-prone ownership transfer

TOPIC	STATUS	IMPACT	LIKELIHOOD
Input Validation	Wont Do	High	Low

In Account.sol, the account owner may transfer ownership to the new address by calling the `transferOwnership()`. However, there is no validation that the new address provided is valid or that it can be used to interact with the account. In case it is invalid, privileged actions accessible only by the

account owner will not be executable. In addition, all MARGIN_ASSET withdrawals will be directed toward an invalid owner address.

Remediations To Consider

- Implement 2 step ownership transfer process, where the new owner needs to claim the account for it to be considered transferred.

RESPONSE BY KWENTA

> This is an acceptable risk, and not one contracts should safeguard against.

~~L-1~~ Fees can be changed and exceed 100% of a trades value

TOPIC	STATUS	IMPACT	LIKELIHOOD
Trust Model	Fixed	High	Low

In Settings.sol, the fees that an account pays for making a transaction can be set by the owner of the contract by calling `setTradeFee()`, `setLimitOrderFee()`, and `setStopOrderFee()`. The only restriction is that the new doesn't exceed the `MAX_BPS`, which caps the fee at 100% of the `sizeDelta` of an account transaction. When a conditional trade is made in Account.sol, the `tradeFee` and either the `stopOrderFee` or `limitOrderFee` add together to determine the total fee for the transaction, allowing a max fee of 200% of a trades value.

Since these trade values can be set at any time, the owner could increase the trade fee to be 100% and users funds would either be locked or taken by the set `treasury` address.

Remediations To Consider

Either

- Set a max trade fee that is more reasonable.
- Ensure the owner of the settings contract is a multisig with a sufficient timelock to allow users time to exit their position if malicious fees are set.

~~L-2~~ Ensure market is active for conditional trades

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Fixed	Low	Low

In Account.sol's `_validConditionalOrder()` is called by Gelato to check if a set conditional trade has met the conditions to trigger its execution. The execution occurs when Gelato calls `executeConditionalOrder()`, and is not supposed to revert.

However, there is the possibility for a market to be inactive or paused for multiple reasons, and the status of the market isn't validated in `_validConditionalOrder()`, potentially causing the execution to revert if interacting with an inactive market.

Remediations To Consider

In `_validConditionalOrder()` check to ensure the market is active to prevent conditional trades from failing to execute.

~~L-3~~ Account is not upgrade safe

TOPIC	STATUS	IMPACT	LIKELIHOOD
Upgradability	Fixed	Medium	Low

Account.sol is intended to be deployed as an implementation contract, with proxies delegating to the current implementation which is retrieved from Factory.sol. Account inherits from multiple contracts which don't have a storage gap set. If Account needs an upgrade, this can potentially limit the options of what can be upgraded, as there is a potential for storage collisions to occur if not handled properly.

Remediations To Consider

Add storage gaps to Account.sol's inherited contracts to allow more freedom for future upgrades.

~~G-1~~ Shifting arrays is expensive

TOPIC	STATUS	GAS SAVINGS
Gas Optimization	Fixed	Medium

In Factory.sol, when the ownership of an account gets updated, the account calls `updateAccountOwnership()`, which updates an array of accounts owned by an owner, removing the account from the old owner account's array and adding it to the new owner's account array.

However, when removing the account from the old owner, it loops through the array until the address is found, then calls `_shiftArrayLeftFrom()`, which removes the account address at the passed-in index, then shifts every address after the removed one each a space to the left. If an address owns many accounts, this can be expensive, as each address shifted is a storage write.

Remediations To Consider

Instead of shifting the addresses, retrieve the address at the end of the array and set it to the index that is being removed, then `pop()` the array. This ensures there is only a single storage write, no matter the size of the address array.

~~G-2~~ Unnecessary repeated calls to _getPerpsV2Market()

TOPIC	STATUS	GAS SAVINGS
Gas Optimization	Fixed	Low

In Account.sol's `executeConditionalOrder()`, there are multiple calls to `_getPerpsV2Market()`, which is an external function call to retrieve the `market` contract from a `marketKey`, each call costs unnecessary additional gas.

Remediations To Consider

Call `_getPerpsV2Market()` and store the resulting address in memory, and reuse it where necessary.

~~G-3~~ `futuresMarketManager` can be immutable

TOPIC	STATUS	GAS SAVINGS
Gas Optimization	Fixed	Low

In Account.sol, `futuresMarketManager` is set in the initializer by querying the ADDRESS_RESOLVER for its contract address.

However, the futuresMarketManager is unlikely to change, and can be set in the constructor using the same method.

Remediations To Consider

Set the futuresMarketManager in the constructor rather than on each initialization. This will reduce gas costs of deployment and operation.

~~Q-1~~ Unnecessary account parameter in events

TOPIC	STATUS	QUALITY IMPACT
Code Quality	Fixed	Low

In Events.sol, each function ensures the caller is an account contract by querying the Factory.sol contract. It then emits the relevant events, taking in an `account` address as a parameter.

However, each call to these functions by the Account.sol contract passes in its own address as the `account` parameter, meaning `account` will always equal `msg.sender`.

Remediations To Consider

Remove the account parameter from each function in Event.sol, and replace with msg.sender.

~~Q-2~~ Unnecessary parameter and check

TOPIC	STATUS	QUALITY IMPACT
Code Quality	Fixed	Low

In Factory.sol's `updateAccountOwnership()`, an `_account` address is passed in and checked to see if it's a valid address, and if msg.sender equals _account:

```
// ensure account is registered by factory
if (!accounts[_account]) revert AccountDoesNotExist();

// ensure function caller is the account
if (msg.sender != _account) revert OnlyAccount();
```

However, the same thing can be accomplished by checking `msg.sender` on the `accounts` mapping, removing the need for an `_account` parameter and the second check that throws `OnlyAccount`

Remediations To Consider

Remove the `_account` parameter and replace all instances of it with msg.sender. Also remove the check if `msg.sender` ≠ `_account`.

~~Q-3~~ FreeMargin check can be made into internal function

TOPIC	STATUS	QUALITY IMPACT
Code Quality	Fixed	Low

In Account.sol, there are multiple checks to ensure that if there is sufficient `freeMargin()` to allow a trade to occur. These checks are found in functions, `_modifyAccountMargin`, `_perpsV2ModifyMargin`, `_placeConditionalOrder`, and `_imposeFee`, each are very similar or identical to each other and can be replaced with an internal function.

Remediations To Consider

Replace freeMargin checks with an internal function to prevent duplicate code, and clean up the codebase.

~~Q-4~~ Duplicate calls to `transferMargin()`

TOPIC	STATUS	QUALITY IMPACT
Code Quality	Fixed	Low

In Account.sol's `_perpsV2ModifyMargin()` there is a branch if `_amount` is greater than zero:

```
if (_amount > 0) {
    if (uint256(_amount) > freeMargin()) {
        revert InsufficientFreeMargin(freeMargin(), uint256(_amount));
    } else {
        IPerpsV2MarketConsolidated(_market).transferMargin(_amount);
    }
} else {
    IPerpsV2MarketConsolidated(_market).transferMargin(_amount);
}
```

Each branch calls the same function `transferMargin()` on the market with the same `_amount` parameter.

Remediations To Consider

Remove the else branch, and check if there is enough free margin, then make the call to `transferMargin()` like this:

```
if (_amount > 0 && uint256(_amount) > freeMargin()) {
    revert InsufficientFreeMargin(freeMargin(), uint256(_amount));
}
IPerpsV2MarketConsolidated(_market).transferMargin(_amount);
```

~~Q-5~~ `MIN_ETH` never used

TOPIC	STATUS	QUALITY IMPACT
Code Quality	Fixed	Low

In Account.sol, the constant MIN_ETH is initialized but never used.

Remediations To Consider

Remove `MIN_ETH` as it is never used.

Q-6 Missing event information for filled conditional order

TOPIC	STATUS	QUALITY IMPACT
Code Quality	Wont Do	Low

In Account.sol, when `executeConditionalOrder()` is performed, the amount of `sizeDelta` filled for a particular conditional order may differ from the `sizeDelta` defined initially for that order. This edge case can happen when the `sizeDelta` of conditional order is larger than the position's `currentSize`.

Consider adding an attribute to the `emitConditionalOrderFilled` for `sizeDelta` processed.

~~Q-7~~ Missing indexed attribute for conditional order events

TOPIC	STATUS	QUALITY IMPACT
Code Quality	Fixed	Low

For events, ConditionalOrderPlaced, ConditionalOrderCancelled, and ConditionalOrderFilled consider making the conditionalOrderId parameter indexed to facilitate off-chain monitoring and tracking.

~~Q-8~~ Documentation improvements

TOPIC	STATUS	QUALITY IMPACT
Code Quality	Fixed	Low

- In IFactory.sol, natspec is missing for the version param of the `NewAccount` event.
- In Account.sol, natspec for the return value of `_getPerpsV2Market()` indicates IPerpsV2Market type. However, the return value is of type `IPerpsV2MarketConsolidated`.

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 Emergent 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.