macro

Kwenta A-3

Security Audit

October 19, 2022
Version 1.0.0

Prepared by Zellic

Table of Contents

- Introduction
- Overall Assessment
- Specification
- Source Code
- Issue Descriptions and Recommendations
- Security Levels Reference
- Disclaimer

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 Zellic's Kevin Kennedy and Macro's Jay Jonah from September 5, 2022 to September 23, 2022.

The purpose of this audit is to review the source code of certain Kwenta facility contracts, and provide feedback on the design, architecture, and quality of the contracts as well as any findings on validating the correctness and security of the software in its entirety.

Disclaimer: While Macro's Kwenta 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 prevent any exploits.

Overall Assessment

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

Severity	Count	Acknowledged	Won't Do	Addressed
High	1	-	-	1
Medium	1	-	-	1
Low	3	-	-	3
Code Quality	3	-	-	2
Gas Optimization	1	-	-	1

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.
- EIP-1
- EIP-19

Source Code

The following source code was reviewed during the audit:

- **Repository:** kwenta-managers
- **Commit Hash:** 95e7c75a66c76ec72dc4263cb4ff1ef3303e80fc7

Specifically, we audited the following contracts within this repository:

Contract	SHA256
contracts/AbstractConditionalOrder.sol	8f1a3d6e3f6a0ce2f2f7a4f823f85d579ac8d1b8f9f0823867e2f9a8ef9fe2fe1d
contracts/AbstractGelato.sol	c76c1fc6d2cc7f5f2279f0f7f0f0377d75c5fa8e8b8e8e8e8e8e8e8e7c7c7c7c7c
contracts/AbstractGelatoGreetings.sol	19ccb2f077beab7aeeb6aaf72f1c38f2e0245f4375d38e2aaeef5fce8b8e7bd7e5
contracts/interfaces/IAddressResolver.sol	38870d3b8d7f2d5750d2f2ce7d8c7bd7f4d2d2e2e2e2e8d8e8e8f8f8f8f8f8f8
contracts/interfaces/IExchanger.sol	e4c2284e9f7f7f9df9f9b28d9fcd8d1228fef5d7f3e8e8d8f8f8e5d75d8e8d5
contracts/interfaces/IFuturesMarket.sol	ef7a9e6d2d5c8f3f3f3f3e8e8e8f78fe0f3f7fd3787f7f7f7f7e8e8e5e7fd8d8e8e75
contracts/interfaces/IFuturesMarketBaseTypes.sol	3d8ba5a8eaeaeaeaeaeaeaeaeaeaeaeaeaeaeaeaeaeaeaeaeaeaeaeaeaeaeaeae
contracts/interfaces/IFuturesMarketManager.sol	c6ea6d3c5c3b3df7a2ca8d3d3f8f8c5cd88b0f8fe6ef7fd8e75d7d8f8f8d8d7d8
contracts/interfaces/IGelatoBox.sol	3c12e0e2b3df3bf7f8e2a3df3c2c3df3f3f3f3e8e8e8e8e8e8e8e8e8e8e8e8e8
contracts/interfaces/IGelatoBoxTypes.sol	3a8e7f4a2a2c3748f8d7f3f3f3f3f3f3f3f3f3f3f3f3f3f3f3f3f3f3f3f3f3f3
contracts/interfaces/IOps.sol	73a04c1ff1f2e0f2a0f8f1cf3f8e3f3f8f3f3f3f3f3f3f3f3f3f3f3f3f3f3f3f
contracts/interfaces/IPyth.sol	9d8cca8bc5d9f8e2e2f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8f8
contracts/interfaces/IPythEvents.sol	2c7f1d8b1a1c1e7b8c2d3e4f5a6b7c8d9e0f1a2b3c4d5e6f7a8b9c0d1e2f3a4b
contracts/utils/AbstractPausable.sol	2c0a2f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f
contracts/utils/AbstractPausable.sol	9e8b758631c1f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7
contracts/utils/ConfirmedOwner.sol	98a6758d5f2d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3

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.

- [H] Any account holder can withdraw their committed sUSD margin using `executeWithdraw()`
- [M] Order cannot be executed if the fee is equal to the maximum amount that the order account is willing to pay for that order
- [L] `tradeFee`, `limitOrderFee`, and `stopLossFee` cannot be set to `MAX_BPS`
- [L] `ERC20.transfer` return value is not checked
- [L] `TokenLockedEnough`, `TradeFeeChanged`, `LimitOrderFeeChanged`, and `StopLossFeeChanged` emit even if the new value is the same as the old one
- [Q] `marginAsset` is set twice in the `initialize()` function
- [Q] `marginAsset` is immutable
- [Q] `marginAssetSettings` address can be immutable
- [G] `onlyOwner` modifier not needed on `depositAndExecuteSwap()` function

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 (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-s] Critical	We recommend the client **must** fix the issue, no matter what, because not fixing would mean **significant funds/assets WILL be lost.**
[H-s] 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-s] 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, but not yet an existential manner.
[L-s] 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-s] 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-s] Informational	Warnings and things to keep in mind when operating the protocol. No immediate action required.
[G-s] Gas Optimizations	The presented optimization suggestion would save an amount of gas significant enough that it should be worth the development cost of implementing it.

Issue Details

[H] Any account holder can withdraw their committed sUSD margin using `executeWithdraw()`

Topic	Status	Impact	Likelihood
Protocol Design	Fixed ✓	Medium	High

Kwenta is planning to use `sUSD` as a margin asset.

`sUSD` is a token that could be accessed by two different addresses, an implementation address, and a target address.

Both of these addresses share the same state as per as the token state is concerned.

Ideally, you are supposed to interact with the implementation address. However, there is a way someone can exploit the present Kwenta contract by using the target address.

Kwenta's `MarginBase.sol` has a feature that allows account owners to withdraw `withdrawable` assets from the contract i.e. all ERC20s are treated the same.

However, as someone can access the same `tokenState` of implementation address (`sUSD`) via target address, they can rescue `sUSD` and hence the committed margin.

```
/// @notice added to support recovering trapped sUSD tokens
/// @dev callable by owner only, emits event when called
/// @param tokenAmount amount of token to be recovered
function recoversUSD(address tokenAddress, uint256 tokenAmount)
    external
    onlyOwner
{
    if (tokenAddress == address(sUsdProxy)){revert("Cannot recoverSUSD");}
    emit TokenRecovered(msg.sender, tokenAddress, tokenAmount);
}
```

Consider restricting rescuing the target address as well. One can fetch this target address on chain from the implementation proxy's address.

RESPONSE BY KWENTA

After careful consideration, we determined that the functionality provided via recoverSUSD() was unnecessary. This conclusion was not a result of the audit's findings, but rather that the function is solely unneeded in the system and has required no fix it to work properly.

[M] Order cannot be executed if the fee is equal to the maximum amount that the owner of the margin account is willing to pay for that order

Topic	Status	Impact	Likelihood
Use Cases	Fixed ✓	Medium	Medium

If the owner of the margin account places an order and sets the `fee` to be not considered valid and cannot be executed if the cap they specified is equal to the fee.

`if (feeInUnits.gt(_desiredFee)) revert LimitOrderFeeTooHigh(_desiredFee);`

This means the owner of the margin account can lose out on an order that they wanted to place.

Consider updating `lte` to `lt` to make the fee cap inclusive.

[L] `tradeFee`, `limitOrderFee`, and `stopLossFee` cannot be set to `MAX_BPS`

Topic	Status	Impact	Likelihood
Use Cases	Fixed ✓	Low	Low

The maximum fee is a constant 10,000 bps. However, `tradeFee`, `limitOrderFee`, and `stopLossFee` can never be set to 10,000.

Consider updating `lte` to `lt`.

[L] `ERC20.transfer` return value is not checked

Topic	Status	Impact	Likelihood
Events	Fixed ✓	Low	Medium

There is no guarantee that the ERC20 token being rescued will inherit US's ERC20 which reverts if the call fails. The transfer could be unsuccessful and the function will still emit the `Rescued` event and still exit as intended. This could cause confusion for the owner of the margin account.

Consider checking the return value of `ERC20.transfer` and only emitting the `Rescued` event if the return value is true.

RESPONSE BY KWENTA

After careful consideration, we determined that the functionality provided via recoverERC20() was unnecessary. This conclusion was not a result of the audit's finding, but because it's function is solely unneeded in the system and has required no fix it to work properly.

[L] `TokenLockedEnough`, `TradeFeeChanged`, `LimitOrderFeeChanged`, and `StopLossFeeChanged` emit even if the new value is the same as the old one

Topic	Status	Impact	Likelihood
Events	Fixed ✓	Low	Low

`MarginBaseSettings`'s `setTreasury()`, `setTradeFee()`, `setLimitOrderFee()`, and `setStopLossFee()` do not check if the new value is different from the old one when emitting their respective events. This creates the potential for duplicate events to be emitted, which could cause confusion for the client app, such as the client app showing a more recent 'updated' trade fee when in fact it's the same.

[Q] `marginAsset` is set twice in the `initialize()` function

Topic	Status	Gas Optimized
Code Quality	Fixed ✓	Low

In `MarginBase.sol`, the state variable `marginAsset` is unnecessarily set two times in the `initialize()` function on lines 203 and 211.

Consider only setting `marginAsset` one time.

[Q] `marginAssetSettings` address can be immutable

Topic	Status	Gas Optimized
Code Quality	Fixed ✓	Low

In `MarginAccountFactory.sol`, `marginAssetSettings` address is only being assigned inside the constructor but it is not declared as immutable.

Consider changing the variable declaration of `marginAssetSettings` to immutable.

[G] `onlyOwner` modifier not needed on `depositAndExecuteSwap()` function

Topic	Status	Gas Optimized
Gas Optimization	Fixed ✓	Medium

The `onlyOwner` modifier is not needed on the `depositAndExecuteSwap()` function since the function calls `deposit()` which has its own `onlyOwner` modifier. This means the `onlyOwner` modifier would be called 2 times unnecessarily.

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 claims 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 an 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. Macro does not comment on the merits of the project or whether it will function, or whether any third party websites, Macro accounts or repositories will be secure from malicious hacking, loss of or damage to anything uploaded to or accessed via the Services, or completeness of any outcome governed by smart contracts.