



Kwenta A-3

Security Audit

October 19, 2022

Version 1.0.0

Presented by [0xMacro](#)

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 the Macro security team from September 5, 2022 to September 23, 2022.

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
High	1	-	-	1
Medium	1	-	-	1
Low	3	-	-	3
Code Quality	2	-	-	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.
- [KIP-18](#)
- [KIP-19](#)

Source Code

The following source code was reviewed during the audit:

- **Repository:** [smart-margin](#)
- **Commit Hash:** 85c473abbb70c6c710c4552c449f6e1824bdb3cf

Specifically, we audited the following contracts within this repository:









Contract	SHA256
contracts/MarginAccountFactory.sol	87ca2ddc89ad4cd429faef81105bd1f9ec0820f858138df9a279a0ee954c5a43
contracts/MarginBase.sol	c7dc15cd1c6cf93217955f35912f754f5cb5ebb88b6db328cd607cc17c55135c
contracts/MarginBaseSettings.sol	5644182973ee67ae403adf1f33c301a42d30527b258b24abe99c68054ef81eb2
contracts/interfaces/IAddressResolver.sol	1887b65b0871d372d1b2fcc64740e7ee9462616da3e485dda4d97dd4c386c3
contracts/interfaces/IExchanger.sol	44133b6bf82ff9e0d998e26869ca041118abf46f74e0e4d859bfbeaf7bdbae55
contracts/interfaces/IFuturesMarket.sol	efa26e61b2c8192819da56c8370ec43743e17d7929f4d64caa75455688dd0e7c
contracts/interfaces/IFuturesMarketBaseTypes.sol	585a61a8a4eb6ca84af80992f723aa14e9c0a0befa47161e7dd303c4cca5781c
contracts/interfaces/IFuturesMarketManager.sol	c5ead2442b348f2a1e8262d90242ce00d4c00404aa97b8ae756dbfe806b165d8
contracts/interfaces/IMarginBase.sol	b4110e3d4395eff706442e0f6ce240f523751d3d4f5085fb86dda9e002a13185
contracts/interfaces/IMarginBaseTypes.sol	ba0e9f6de2374b08e591a2f4535670f5304adc489

Contract	SHA256
	5db9eb65b5679830d2f5004
contracts/interfaces/IOps.sol	7daddcc8921ed2a884b89ce2eb8ef3985a6fff3b749c2b6d47df25dfe8aab058
contracts/interfaces/ISynth.sol	9d51cd8443d96ec59ab53a6af4582ec2c5fb33cc2c6503485ca787cef9dc4721
contracts/interfaces/IVirtualSynth.sol	2cf52408cec467b8cae38074cbd7566653401b1831b86463afd750995514505f
contracts/utis/MinimalProxyFactory.sol	1143e3ff6352f6d51a49e42229f3a453be4f4ffdc e84faaa3fef30dc25e4c577
contracts/utis/MinimalProxyable.sol	8a86778db51c99fd15d0495552229394fe1fc89ffaec9b82b34e4a2aa5971ef3
contracts/utis/OpsReady.sol	88ab73d58fbe68feb5de4f2ccdfa90e03e41c24940e0c98c0e07e4bd70dd88e1

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.

-  Any account holder can withdraw their committed `sUSD` margin using `rescueERC20()`
-  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
-  `tradeFee`, `limitOrderFee`, and `stopOrderFee` cannot be set to `MAX_BPS`
-  `IERC20.transfer` return value is not checked
-  `TreasuryAddressChanged`, `TradeFeeChanged`, `LimitOrderFeeChanged`, and `StopOrderFeeChanged` emit even if the new value is the same as the old one
-  `marginAsset` is set twice in the `initialize()` function
-  `marginBaseSettings` address can be immutable
-  `onlyOwner` modifier not needed on `depositAndDistribute()` 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 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, <i>or</i> some funds/assets will be lost, <i>or</i> 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, albeit not in an existential manner.
(L-x) Low	The risk is small, unlikely, or may not be 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

H-1

Any account holder can withdraw their committed **sUSD** margin using **rescueERC20()**

TOPIC

Protocol Design

STATUS

Fixed [↗](#)

IMPACT

Medium

LIKELIHOOD

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 redundant assets from the contract. i.e. all ERC20s except the margin asset.

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 erc20 tokens
/// @param tokenAddress: address of token to be recovered
/// @param tokenAmount: amount of token to be recovered
function rescueERC20(address tokenAddress, uint256 tokenAmount)
    external
    onlyOwner
{
    if (tokenAddress == address(marginAsset)) { //@audit high severity
        revert CannotRescueMarginAsset();
    }
    IERC20(tokenAddress).transfer(owner(), tokenAmount);
}
```

```
emit Rescued(tokenAddress, tokenAmount);
}
```

Consider restricting rescuing the target address as well. One can fetch this target address on-chain from the implementation contract address.

RESPONSE BY KWENTA

After careful consideration, we determined that the functionality provided via rescueERC20() was unnecessary. This conclusion was not a result of the audit's finding, but because it's function is solely unrelated to the system and not required for it to work properly.

M-1 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 with a fee cap, the order is not considered valid and cannot be executed if the cap they specified is equal to the fee.

```
if (tooVolatile || dynamicFee >= order.maxDynamicFee) { return (false, 0); }
```

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

Consider updating `>=` to `>` to make the fee cap inclusive.

⚡ tradeFee , limitOrderFee , and stopOrderFee 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 `stopOrderFee` can never be set to 10,000.

Consider updating `>=` to `>`.

⚡

`IERC20.transfer` return value is not checked

TOPIC

Events

STATUS

Fixed [↗](#)

IMPACT

Low

LIKELIHOOD

Medium

There is no guarantee that the ERC20 token being rescued will inherit OZ's ERC20 which `reverts` on all bad inputs. The transfer could be unsuccessful and return false, and the `Rescued` event will still be emitted. This could cause confusion for the owner of the margin account.

Consider checking the return value of `IERC20.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 `rescueERC20()` was unnecessary. This conclusion was not a result of the audit's finding, but because its function is solely unrelated to the system and not required for it to work properly.

⚡

`TreasuryAddressChanged`, `TradeFeeChanged`, `LimitOrderFeeChanged`, and `StopOrderFeeChanged` emit even if the new value is the same as the old one

TOPIC

Events

STATUS

Fixed [↗](#)

IMPACT

Low

LIKELIHOOD

Low

MarginBaseSettings's `setTreasury()`, `setTradeFee()`, `setLimitOrderFee()`, and `setStopOrderFee()` 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 may cause confusion to users on the client app, such as displaying a more recent “updated” trade fee when no data has actually changed.

 Q-1

`marginAsset` is set twice in the `initialize()` function

TOPIC

Code Quality

STATUS

Fixed 

QUALITY IMPACT

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-2

`marginBaseSettings` address can be immutable

TOPIC

Code Quality

STATUS

Fixed 

QUALITY IMPACT

Low

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

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



`onlyOwner` modifier not needed on `depositAndDistribute()` function

TOPIC

Gas Optimization

STATUS

Fixed [↗](#)

GAS SAVINGS

Medium

The `onlyOwner` modifier is not needed on the `depositAndDistribute()` 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 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.