macro

Kwenta A-11

Security Audit

January 17, 2024
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 Jan 10, 2023 to Jan 12, 2024.

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
Code Quality	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.

Trust Model, Assumptions, and Accepted Risks (TMAAR)

Entities

- Users: Owners of a SynthetixV3 PerpsMarket account NFT, which give ownership and control of positions and collateral, as well as ability set specific permissions to other addresses for the account.
- Delegates: Addresses that are given permission by an account owner to execute orders or sign conditional orders on behalf of the account owner.
- Executors: Execute signed conditional orders once conditions are met and receive a conditional order fee. There are two types of executor
- Trusted Executor: This is an executor that is trusted to only execute a conditional order if conditions are met. Conditions are likely verified off-chain, so there is no guarantee all conditions are met when an order is attempting to be executed.
- Normal Executor: Can be anyone, but the conditions set by the order are verified on-chain before execution of the order.
- pDAO: Multisig wallet that has the ability to upgrade the Engine.sol contract.

Trust Model

- Users that sign conditional orders trust that the order will be executed reasonably soon after the conditions are met, this may not be the case as there may not be executors taking orders at any given time, or gas costs could theoretically exceed max conditional order fee set.
- If a trusted executor is used, there is trust that they only execute the order when conditions are met.
- Any delegates given permission to execute orders are trusted to act in the interest of the account owner.
- Engine.sol now has the possibility to be upgraded, this is intended to allow for the engine contract to adapt to changes that synthetix may make, since SynthetixV3 is upgradeable itself. Upgrades are authorized by pDAO, if set, and it is trusted that only upgrades that are beneficial to users will be made.

Assumptions

- It is assumed that when buying/selling, sUSDC and sUSD can be exchanged at a 1:1 rate. Functions using Zap will revert if there isn't a 1:1 exchange, which can occur if fees are added to the sUSD spot market or the price of sUSDC deviates from sUSD.

Accepted Risks

- Synthetix could be exploited, potentially causing users funds to be lost.

Source Code

The following source code was reviewed during the audit:

- **Repository:** smart-margin-v3
- **Commit Hash (initial):** `78b14a18440e9d56a9538ee1ff2e8624e5f6d4e1`
- **Commit Hash (final):** `866479d7c6a0957e085831605ac078afc8aac420`

Specifically, we audited the following contracts within this repository:

Contract	SHA256
src/Engine.sol	`24383912f66cf3b66b3702b756813ba63bf31da98c52fccc3a65244b68c41476`
src/interfaces/IEngine.sol	`ff3f950ea0752668c902fc0f85ca2b5df081cd75e2f125cbcf7cc7c534e9c363`

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.

Q-1 Storage gaps can be added

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

Q-1 Storage gaps can be added

TOPIC	STATUS	QUALITY IMPACT
Code quality	Fixed	Low

`Engine.sol` is now upgradeable to allow it to adapt to changes in SynthetixV3, and potentially add functionality, like is done by inheriting the `Zap.sol` contract. In future upgrades, if storage slots are required for additional functionality it may require restructuring Engine.sol to be a base contract rather than the child-most, since storage slots are directly used.

If `Engine.sol` is desired to inherit all new functionality, it may be wise to add a storage gap before the current storage variables to allow more flexibility to inherit functionality that requires storage slots without requiring `Engine.sol` to be restructured. `Zap.sol` could also require storage slots in the future if additional functionality is required, and could benefit from adding a storage gap as well.

Remediations to Consider

Add storage gaps to either `Zap.sol` or `Engine.sol` to allow more flexibility in developing future upgrades.

Disclaimer