macro

Kwenta A-12

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
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.

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:** zap
- **Commit Hash (initial):** `dbc1978719b2859edd3597efbae555629b61d021`
- **Commit Hash (final):** `43116650b4d7958b243dcb8aadfb921f9c2383a3`

Specifically, we audited the following contracts within this repository:

Contract	SHA256
src/Zap.sol	2cdcb9f51564b5e7a1ec108dc09c7d7820dfc663246164ffd0302927dfb800d4
src/ZapErrors.sol	5914f8be71220e4b5e00259f369edb4e3d46285f1692361909dde265f9fa6be8
src/ZapEvents.sol	566028f6adc384f5af4af7b5e64989b0bba3f97dada52efa6251e9e3429dee4e

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~~ Follow Checks Effects and Interactions pattern

~~G-1~~ Unnecessary referrer for zap buys or sales

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~~ Follow Checks Effects and Interactions pattern

TOPIC	STATUS	QUALITY IMPACT
Code Quality	Fixed	Low

In `Zap.sol`'s `_zapOut()` function, there is a check to ensure the `_amount` is greater than the `_DECIMALS_FACTOR` to prevent all of _amount to be lost to precision when converting from `sUSDC`'s 18 decimals to `USDC`'s 6 decimals. However, this check is made after external calls are made and state has been changed, without mutating `_amount` in the process.

```
/// @notice prior to unwrapping, ensure that there
/// is enough $sUSDC to unwrap
/// @custom:example if $USDC has 6 decimals, and
/// $sUSDC has greater than 6 decimals,
/// then it is possible that the amount of
/// $sUSDC to unwrap is less than 1 $USDC;
/// this contract will prevent such cases
/// @dev if $USDC has 6 decimals, and $sUSDC has 18 decimals,
/// precision may be lost
if (_amount < _DECIMALS_FACTOR) {
    revert InsufficientAmount(_amount);
}
```

Reference: Zap.sol#L212-L223

Remediations to Consider

Move this check to the top of the function to more closely follow the Checks, Effects, and Interactions pattern, and refund users more gas if this check fails during execution.

~~G-1~~ Unnecessary referrer for zap buys or sales

TOPIC	STATUS	GAS SAVINGS
Gas Optimization	Fixed	Low

In `Zap.sol`'s `_zapIn()` and `_zapOut()`, a referrer is passed in as a parameter and used in when either buying or selling `sUSDC` for `sUSD`. The intention of the referrer is to receive a portion of the fees of the spot market sale. However, in this case the assumption is that there will be no fees for this market, since the `minimumAmountReceived` is equal to the amount put in, meaning the sale will not execute unless there is a 1:1 exchange of `sUSD` and s `USDC`, and no fees are taken.

```
/// @notice buy $sUSDC with $sUSD
/// @dev call will result in $sUSDC minted/transferred
/// to the Zap contract
_SPOT_MARKET_PROXY.buy({
    marketId: _SUSDC_SPOT_MARKET_ID,
    usdAmount: _amount,
    minAmountReceived: _amount,
    referrer: _referrer
});
```

Reference: Zap.sol#L192-L200

When executing a SynthetixV3 spot market buy/sell, additional logic is executed if the passed in referrer is non-zero:

```
function _collectReferrerFees(
    Data storage self,
    uint128 marketId,
    OrderFees.Data memory fees,
    address referrer,
    SpotMarketFactory.Data storage factory,
    Transaction.Type transactionType
) private returns (uint256 referrerFeesCollected) {
    if (referrer == address(0)) {
        return 0;
    }
```

```
        uint256 referrerPercentage = self.referrerShare[referrer];
        referrerFeesCollected = fees.fixedFees.mulDecimal(referrerPercentage);

        if (referrerFeesCollected > 0) {
            if (Transaction.isSell(transactionType)) {
                factory.synthetix.withdrawMarketUsd(marketId, referrer, referrer
            } else {
                factory.usdToken.transfer(referrer, referrerFeesCollected);
            }
        }
    }
```

Reference: MarketConfiguration.sol#L520-L542

Since the fees will be zero, passing in `address(0)` as the referrer saves an unnecessary `SLOAD`, and some execution totalling to roughly 2400 gas.

Remediations to Consider

Do not accept a referrer, or use `address(0)` in `_zapIn()` and `_zapOut()` to save users a bit of gas. It is important to note that the referrer is also logged in the `SynthBought` event, which may be desired for referrers despite the additional gas cost.

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 Kwenta 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.