



Kwenta A-6

Security Audit

July 21, 2023

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 July 17, 2023 to July 20, 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
Code Quality	2	-	-	2
Informational	1	1	-	-
Gas Optimization	2	-	-	2

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-80](#)
- [KIP-87](#)

Source Code

The following source code was reviewed during the audit:

- **Repository:** [smart-margin](#)
- **Commit Hash:** `5cc3ccc3817d41ad28e2b777450b308f460c9d4c`






Specifically, we audited the following contracts within this repository:

Contract	SHA256
src/Account.sol	<code>2f7e12de5990f91e637fbffe52687165974a565260dbfa390c7a5becf290634c</code>
src/Events.sol	<code>589e2b8ba86ce005e781484c4a60e560f520f907914b2c83fa05603af8baddf4</code>
src/Settings.sol	<code>899249ee3c5f8b6abf35b08787f3960639a2c8837078d5d05d29998ee6ef43b7</code>
src/utils/uniswap/BytesLib.sol	<code>7903c99885cbaf879e23620a6d622139b0b5b9f9bc90c0af0170645ede99ea42</code>
src/utils/uniswap/Constants.sol	<code>35fad5748bc13afa2f849e7840c6955bbbc249414b5b24dfc9380f0274b6a2d6</code>
src/utils/uniswap/SafeCast160.sol	<code>8b6ef0f56be154d60c3e3ddef5f940c1c100ed4b2a1cebf843c91ca59f96870a</code>
src/utils/uniswap/V3Path.sol	<code>c38051ae1c2ef6bc53413eb57146fffb377f6dc9a8d854572a5ff8605ab515393</code>

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.

-  Swap approvals can be removed
-  `nonReentrant` called excess times
-  Unused event
-  Unnecessary `payable` casting
-  Accounts cannot set `executorFee`

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	<p>The risk is small, unlikely, or may not be relevant to the project in a meaningful way.</p> <p>Whether or not the project wants to develop a fix is up to the goals and needs of the project.</p>
(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

⚡ Swap approvals can be removed

TOPIC

Gas Optimization

STATUS

Fixed [↗](#)

GAS SAVINGS

Medium

In Account.sol when the owner swaps tokens using the `UNISWAP_V3_SWAP` command, in `_uniswapV3Swap()` the account contract approves the `PERMIT2` address for the `tokenIn` and also calls approves the `UNISWAP_UNIVERSAL_ROUTER` address on `PERMIT2` on the same `tokenIn`.

```
// approve Permit2 to spend _amountIn of this contract's tokenIn
IERC20(tokenIn).approve(address(PERMIT2), _amountIn);

// approve tokens to be swapped via Universal Router
PERMIT2.approve({
    token: tokenIn,
    spender: address(UNISWAP_UNIVERSAL_ROUTER),
    amount: _amountIn.toUint160(),
    /// @dev timestamp will never overflow (i.e. maximum value of uint48 is year 8 mill:
    expiration: uint48(block.timestamp)
});
```

Reference: [Account.sol#L1045-L1055](#)

However, if the desired tokens to swap are directly sent to the `UNISWAP_UNIVERSAL_ROUTER` address before calling `_universalRouterExecute()` and the `payerIsUser` value set as data for the `UNISWAP_UNIVERSAL_ROUTER`'s `execute()` `input` is set to false, then these approvals can be removed.

Remediations to Consider

Remove the approvals and transfer the `tokenIn` to the `UNISWAP_UNIVERSAL_ROUTER` instead of the contract in the case where `MarginAsset == tokenIn`, otherwise transfer `assetIn` from the contract directly. Also set `payerIsUser` to be false. Doing so can reduce gas costs of executing swaps.

2 **nonReentrant** called excess times

TOPIC

Gas Optimization

STATUS

Fixed 

GAS SAVINGS

Low

In `Account.sol`'s `dispatch()` function has a `nonReentrant` modifier that prevents reentrancy. However, `dispatch()` is internal and only called by `execute()` for each command. This means that the `locked` value will be read and set twice per call to dispatch for each command execute, but it only needs to be set once in `execute()`.

Remediations to Consider

Move the `nonReentrant` modifier from `dispatch()` to the `execute()` function to prevent unnecessary calls to the `nonReentrant` modifier.

4 **Unused event**

TOPIC

Code Quality

STATUS

Fixed 

QUALITY IMPACT

Low

In `Events.sol`, the function `emitExecutorFeeSet()` is never called. It implies that an account can set their own executor fee, but that is currently not the case.

Remediations to Consider

Remove `emitExecutorFeeSet()` and its associated `ExecutorFeeSet` event as they are not used.

Q-2 Unnecessary payable casting

TOPIC

Code Quality

STATUS

Fixed 

QUALITY IMPACT

Low

In `Account.sol`'s `_payExecutorFee()`, when the executor is not Gelato, the ETH fee is sent to the executor via a low level call.

```
fee = SETTINGS.executorFee();
(bool success,) = payable(msg.sender).call{value: fee}("");
if (!success) revert CannotPayExecutorFee(fee, msg.sender);
```

Reference: (`Account.sol`#L911-L913)[<https://github.com/Kwenta/smart-margin/blob/5cc3ccc3817d41ad28e2b777450b308f460c9d4c/src/Account.sol#L911-L913>]

However, low level calls to transfer ETH do not require the address to be set to `payable`.

Remediations to Consider

Remove the casting of `msg.sender` to `payable`.

I-1 Accounts cannot set executorFee

TOPIC

Informational

STATUS

Acknowledged

IMPACT

Informational *

The `executorFee` used to pay for the execution of conditional orders, in the case where the executor is not gelato, is set by the owner of the `Settings.sol` contract, and cannot be adjusted by account owners. In times where gas prices cause conditional order execution to be more expensive than the currently set fee, then gelato will most likely be the only executor of the conditional order. Conversely, if the currently set fee is much lower than the gas cost to execute, it will most likely occur quickly but will may be executed for a larger fee than what Gelato would have charged if they executed the order.

RESPONSE BY KWENTA

Can be remedied via robust chain monitoring and quick calibration response

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.