macro

Kwenta A-16

Security Audit

August 12, 2024
Version 1.0.0

Presented by 0xMacro

Table of Contents

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 22 to July 25, 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
Medium	2	-	-	2
Low	1	-	1	-
Code Quality	6	-	2	4
Informational	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.

Source Code

The following source code was reviewed during the audit:

- **Repository:** paymaster
- **Commit Hash:** `ce5b0b11a84568269f410d4072106f03325df463`

Specifically, we audited the following contracts within this repository:

Contract	SHA256
./src/MarginPaymaster.sol	`45643499f3f79b7ee8046a7ea76b069d7cd382644de84e05c9f793e0b0e8fe92`
./src/interfaces/external/IEngine.sol	`9e2caac280c27aeb61c09a21f70eecf943b13505a4bec07e5d73fdf4d45302c5`
./src/interfaces/external/INftModule.sol	`50a282c824895ca4d7fcf7c232814210837c0822f96f081ed872eb6e68f45b81`
./src/interfaces/external/IPerpsMarketProxy.sol	`cf499ca86ad3304da0da9e68a0af0824f1c3b9f9c986fe4af1f896c5f7a3fc9b`
./src/interfaces/external/IUniswapV3Pool.sol	`4913b1537ce723f927224fca1d80d1ce34a11ddcb37d81346f8d58692f507024`
./src/interfaces/external/IV3SwapRouter.sol	`a93f9bf59f77910150c498217b8a8d997e41363ef72ce4fcef88780019bce5ba`
./src/interfaces/external/IWETH9.sol	`aa1368a1534f0da91f88540ca2d911f3339e4903df32ad6106dca7c1f519b226`

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.

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

~~M-1~~ `postOp` will revert for any `accountId` with zero withdrawable margin

TOPIC	STATUS	IMPACT	LIKELIHOOD
Integration	Fixed	Spec Breaking	Medium

When executing the `postOp()` logic, the `MarginPaymaster` contract will attempt to pull assets from the user's withdrawal margin if he does not have available `USDC` tokens. To handle all potential revert scenarios, the `withdrawFromMargin()` function checks that the proper `accountId` is used and that the paymaster contract has the `*PERPS_MODIFY_COLLATERAL_PERMISSION`* permission. After access control checks, the available withdrawal margin is checked and function execution returns if negative.

```
int256 withdrawableMargin = perpsMarketSNXV3.getWithdrawableMargin(
    accountId
);

if (withdrawableMargin < 0) return 0;
```

Reference: MarginPaymaster.sol#L392-396

However, if the account's available margin is zero, the function will not return, and it will attempt to execute a `modifyCollateral()` call with a zero amount:

```
uint256 withdrawableMarginUint = uint256(withdrawableMargin);

uint256 amountToPullFromMargin = min(
    sUSDToWithdrawFromMargin,
    withdrawableMarginUint
```

```
);

// pull sUSD from margin
perpsMarketSNXV3.modifyCollateral(
    accountId,
    sUSDId,
    -int256(amountToPullFromMargin)
);
```

Reference: MarginPaymaster.sol#L397-409

This will cause the `perpsMarketSNXV3.modifyCollateral()` to revert with an `InvalidAmountDelta` error, causing the `postOp()` to revert.

Remediations to Consider:

Consider returning `0` if the available withdrawal margin is ≤ `0` when verifying the `withdrawableMargin` value.

~~M-2~~ Transactions with a wrong `accountId` ownership will be incorrectly sponsored

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec breaking	Fixed	Medium	Medium

In the MarginPaymaster contract, the function `withdrawFromMargin()` is responsible for recouping the transaction cost from the associated account's collateral. When retrieving the owner of the corresponding account fails, it falls back to functionality that tries to withdraw assets from the first account associated with a particular user.

```
if (accountId != 0) {
    // check if the account Id is valid
    try snxV3AccountsModule.ownerOf(accountId) returns (address owner) {
        // only allow the owners accounts to subsidise gas
        if (owner != sender) return 0;
```

```
    } catch {
        // set accountId to zero, and then check if the sender has an account c
        accountId = 0;
    }
}
```

Reference: MarginPaymaster.sol#L366-375

However, when retrieving the owner of the provided account does not fail, but instead returns a different owner, execution returns immediately instead and does not try to withdraw assets from the first account associated with a particular user.

As a result, if the caller provides the account of a different user, they will never be charged gas transaction costs. Instead, MarginPaymaster will be incorrectly sponsoring these transactions.

Remediations to Consider

- Make the following change in `if` branch

```
- if (owner != sender) return 0;
+ if (owner != sender) accountId = 0;
```

  Reference: MarginPaymaster.sol#L370

L-1 Paymaster may drain user assets with an unexpected value of `percentageMarkup` configuration setting

TOPIC	STATUS	IMPACT	LIKELIHOOD
Input validation	Wont Do	High	Low

In the MarginPaymaster contract, the owner may set and update the percentageMarkup value using `setPercentageMarkup()`. Currently, there is no input

value restriction, and high values for `percentageMarkup` will result in charging the user more than expected.

Therefore, if a high `percentageMarkup` value is set accidentally or intentionally by a compromised contract owner, it may drain the smart account's balance and free-to-withdraw margin from an associated account on the SNXV3 perp market.

Remediations to Consider

- Add restrictions and corresponding checks on `percentageMarkup` value initialization and updates.

~~Q-1~~ Missing event emission for important state updates

TOPIC	STATUS	QUALITY IMPACT
Events	Fixed	Low

Several important state-changing functions do not emit events.

- `setPercentageMarkup()`
- `setAuthorizer()`

Remediations to Consider

- Add corresponding events for easier off-chain tracking and monitoring.

~~Q-2~~ Redundant hashing in `getHash()` function

TOPIC	STATUS	QUALITY IMPACT

Best practices	Fixed	Low

The `getHash()` function uses both `paymasterAddress` and `address(this)` as hash inputs. However, these two variables should have the same value, making the encoding of both redundant.

```
    return
        keccak256(
            abi.encode(
                ...
                uint256(bytes32(paymasterAddress)),
                ...
                address(this)
            )
        );
```

Reference: MarginPaymaster.sol#L120-L136

Remediations to Consider

Consider validating that parsed `paymasterAddress` is equal to `address(this)` and removing one of those two from the input for hashing.

~~Q-3~~ Set `depositToEntryPoint()` to be `payable`

TOPIC	STATUS	QUALITY IMPACT
Best practices	Fixed	Low

Currently, to deposit funds to the `EntryPoint` contract, it is necessary first to send specific amount of native token directly to the `MarginPaymaster` contract, and then call `MarginPaymaster.depositToEntryPoint()` function.

However, by adding `payable` modifier to the `depositToEntryPoint()` function, funds can be deposited in one transaction.

Consider updating `depositToEntryPoint()` function signature in the following way:

```
-    function depositToEntryPoint(uint256 amount) external onlyOwner {
+    function depositToEntryPoint(uint256 amount) payable external onlyOwner {
```

Reference: MarginPaymaster.sol#L258

~~Q-4~~ Making the amount of USDC swapped configurable in `swapUSDCToETH()` function for more flexible swapping

TOPIC	STATUS	QUALITY IMPACT
Best practices	Addressed	Low

Currently, the `swapUSDCToETH()` function swaps **all** of the USDC in the `MarginPaymaster` contract to ETH.

Consider adding `usdcAmount` as an argument in the `swapUSDCToETH()` function for more flexible swapping and having less risk, making the Uniswap pool volatile from swapping with the huge amount

Reference: MarginPaymaster.sol#L250

RESPONSE BY KWENTA

> No logic change, added natspec

Q-5 Not using the latest Entrypoint version

TOPIC	STATUS	QUALITY IMPACT
Integration	Wont Do	Medium

The current `MarginPaymaster` contract is designed to use the Entrypoint version 0.6.0, released in April 2023. Some major optimizations and changes were deployed with the latest 0.7.0 version, as explicitly described in the release notes, while some were optimizing Paymaster integrations and solving potential issues.

Consider upgrading the Entrypoint version to 0.7.0.

Q-6 `_zapIn()` is not used, so it can be removed

TOPIC	STATUS	QUALITY IMPACT
Unnecessary code	Wont Do	Low

The `MarginPaymaster` contract inherits the `Zap` contract, which contains 2 internal functions:

- `_zapIn()` , and
- `_zapOut()`

However, `MarginPaymaster` only uses `_zapOut()`.

Consider removing the `_zapIn()` function as it is not used.

I-1

External integrations may cause MarginPaymaster to stop functioning

TOPIC	IMPACT
Interoperability	Informational ✳

The MarginPaymaster relies on USDC and Synthetix v3 perpetual market module to calculate and pull assets on the operation sender's behalf. Both external parties can prevent this contract from working:

- `USDC` blacklisting the MarginPaymaster address or the sender.
- Synthetix governance disabling the feature flag for the perps market module.

This will essentially cause all `UserOp` signed with this paymaster as a target to not be executed

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.