macro

Kwenta A-12

Security Audit

July 26, 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 25th to July 26th, 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

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:** token
- **Commit Hash:** `38114f26b5cc19e59ab5ff78578855df7bc41f85`

Specifically, we audited the following contracts within this repository:

Contract	SHA256
contracts/StakingRewardsNotifier.sol	`fad6b6939a78ec7c2f60380d89b4b2d5f fc5996f87055456b1a715d572a639e8`
contracts/StakingRewardsV2.sol	`fc70e979490e2ede50d042b2385281f95 d3c4672e7cf83d3e6ae70e2dc11f17c`

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~~ USDC tokens intended as rewards can be removed by the owner

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Fixed	High	Low

In `StakingRewardsV2.sol`, the `recoverERC20()` ****function allows the owner to retrieve `ERC20` tokens that may have accidentally been sent into the contract. Since `kwenta` tokens are held in the contract via staking or to distribute rewards, recovering `kwenta` tokens is prevented in `recoverERC20()`, so that these tokens cannot be stolen from stakers:

```
/// @inheritdoc IStakingRewardsV2
function recoverERC20(address _tokenAddress, uint256 _tokenAmount) external
    if (_tokenAddress == address(kwenta)) revert CannotRecoverStakingToken()
    emit Recovered(_tokenAddress, _tokenAmount);
    IERC20(_tokenAddress).transfer(owner(), _tokenAmount);
}
```

Reference: StakingRewardsV2.sol#L709-L714

With the addition of `USDC` rewards, now `USDC` will be held in the contract and distributed as rewards to stakers, which means in `recoverERC20()` is called with the `USDC` address, these tokens owed to stakers can be stolen. If all the `USDC` were to be taken from the contract, then calls to `_getReward()` would start to fail, as the contract would no longer have the balance to cover owed `USDC` rewards.

Remediations to Consider

Prevent `USDC` from being recovered via `recoverERC20()`

~~M-2~~ Rounding errors can lead to more significant loss with USDC

TOPIC	STATUS	IMPACT	LIKELIHOOD
Precision	Fixed	Medium	High

Currently precision loss occurs when setting the rewardRate in `notifyRewardAmount()`, since the set amount is divided by a `rewardsDuration`, where the remainder is lost:

```
/// @inheritdoc IStakingRewardsV2
function notifyRewardAmount(uint256 _reward, uint256 _rewardUsdc)
    external
    onlyRewardsNotifier
    updateReward(address(0))
{
    if (block.timestamp >= periodFinish) {
        rewardRate = _reward / rewardsDuration;
        rewardRateUSDC = _rewardUsdc / rewardsDuration;
    } else {
        uint256 remaining = periodFinish - block.timestamp;

        uint256 leftover = remaining * rewardRate;
        rewardRate = (_reward + leftover) / rewardsDuration;

        uint256 leftoverUsdc = remaining * rewardRateUSDC;
        rewardRateUSDC = (_rewardUsdc + leftoverUsdc) / rewardsDuration;
    }

    lastUpdateTime = block.timestamp;
    periodFinish = block.timestamp + rewardsDuration;
    emit RewardAdded(_reward, _rewardUsdc);
}
```

Reference: StakingRewardsV2.sol#L644-L666

The precision loss varies, based on reward and rewardsDuration, but is typically fairly negligible for tokens with 18 decimals. However, USDC has 6 decimals so this precision loss can lead to users losing out more substantial rewards, and having the excess dust locked in the contract.

Remediations to Consider

Multiply the rewards amount by some large number before dividing by the rewards duration, preserving the decimals, then divide out this number when used when calculating the `rewardPerToken()` in order to maintain a higher precision. Note: this may require a reinitalizer function to update the current kwenta rewards rate to have the updated rate multiple.

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.