#HashEx
BLOCKCHAIN SECURITY

Kwil

smart contracts final audit report

February 2025

hashex.org

contact@hashex.org

Contents

1. Disclaimer 3

2. Overview 4

3. Found issues 6

4. Contracts 7

5. Conclusion 11

Appendix A. Issues' severity classification 12

Appendix B. Issue status description 13

Appendix C. List of examined issue types 14

Appendix D. Centralization risks classification 15

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the Kwil team to perform an audit of their smart contract. The audit was conducted between 27/01/2025 and 29/01/2025.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available at @kwilteam/rewards_contracts GitHub repository and was audited after the commit c974e75.

Update. The Kwil team has responded to this report. The updated code is available in the same repository after the commit 679853c.

2.1 Summary

Project name	Kwil
URL	https://kwil.com
Platform	Ethereum, Polygon Network, Base
Language	Solidity
Centralization level	Low
Centralization risk	Low

2.2 Contracts

Name	Address
RewardDistributor	

3. Found issues

High	1 (17%)
Medium	1 (17%)
Low	1 (17%)
Info	3 (49%)

C8d. RewardDistributor

ID	Severity	Title	Status
C8dIc4	High	Possible reward shortage	Resolved
C8dIc8	Medium	Invalid event parameter	Resolved
C8dIc5	Low	Gas optimizations	Resolved
C8dIc7	Info	Rewarding transaction signer	Acknowledged
C8dIc9	Info	Inconsistent documentation	Resolved
C8dIc6	Info	Unsupported tokens	Resolved

4. Contracts

C8d. RewardDistributor

Overview

The RewardDistributor is a reward distributor contract allowing whitelisted users to claim their rewards in form of immutable ERC20 token.

Issues

C8dIc4 Possible reward shortage

High Resolved

The rewards are posted by the contract owner in form of a root of Merkle tree and the total reward amount is ensured to be present on the contract's balance. However, the total sum of Merkle leaves is not checked in any kind, meaning that any particular reward root can consume an arbitrary amount of reward tokens.

```
    /// @dev Since root is unique, it can also prevent tx replay.
    /// @dev We can also use 'nonce', but seems not necessary since rewardRoot is unique.
    /// @param root The merkle tree root of an epoch reward.
    /// @param amount The total value of this reward.
    function postReward(bytes32 root, uint256 amount) external {
        require(msg.sender == safe, "Not allowed");
        require(amount > 0, "Total amount zero");
        require(rewardPoster[root] == address(0), "Already posted");
        require(rewardToken.balanceOf(address(this)) >= postedRewards + amount,
"Insufficient contract reward balance");

        rewardPoster[root] = tx.origin; // whoever initiate this TX through gnosis-safe
        postedRewards += amount;

        emit RewardPosted(root, amount, msg.sender);
    }
```

Recommendation

Consider storing the `amount` parameter of the `postReward()` function and track total amount of claimed rewards of the root to ensure it is always not greater than stored value.

C8dIc8 Invalid event parameter

Medium Resolved

`RewardPosted` event emits `msg.sender` instead of `tx.origin` as a reward poster.

```
    /// @notice rewardPoster maps a reward hash(merkle tree root) to the wallet that posts
the reward on chain.
    mapping(bytes32 => address) public rewardPoster;
    event RewardPosted(bytes32 root, uint256 amount, address poster);

    function postReward(bytes32 root, uint256 amount) external {
        . . .
        rewardPoster[root] = tx.origin; // whoever initiate this TX through gnosis-safe
        emit RewardPosted(root, amount, msg.sender);
    }
```

Recommendation

Fix the error or add documentation.

C8dIc5 Gas optimizations

Low Resolved

1. The `safe` variable should be declared as immutable.

2. Multiple reads from storage in the `updatePosterFee()` function: `nonce` variable.

3. Multiple reads from storage in the `claimReward()` function: `posterFee` variable.

4. Unused functions `receive()` and `fallback()` can be safely removed from the bytecode.

5. Unchecked math could be used in the `claimReward()` function for excess calculation.

C8dIc7 Rewarding transaction signer

Info Acknowledged

Gnosis Safe account allows creating an operation to be performed by an arbitrary address. In that case the `postReward()` function is susceptible to front-running in order to become fee receiver.

Additionally, if the Safe account is configured to be used as an ERC4337 account or similar, i.e., using user operations in bundles instead of atomic transactions, then the RewardDistributor's poster fee may be forwarded to a third-party instance that has already been paid via paymaster or entry point contract.

```
function postReward(bytes32 root, uint256 amount) external {
    . . .
    rewardPoster[root] = tx.origin; // whoever initiate this TX through gnosis-safe
    . . .
}
```

Team response

1. There is not enough information in the contract to determine when front running the poster will be profitable. The total fee paid to the poster is the `posterFee` * the amount of recipients in the merkle tree. Because the contract only stores the merkle hash, a prospective front runner cannot see how many recipients will claim from each hash. Therefore, bots cannot determine when front running is profitable.Furthermore, in future versions, we will design a more sophisticated mechanism to compensate posters that cannot be front-run. In the initial deployments of the Reward Contract, the poster will be operated by parties that accept the front running risk.

2. In our factory contract, we make sure that ERC4337 is not used. We will also make it clear in the documentation that the SAFE Wallet should not be in ERC4337 mode.

C8dIc9 Inconsistent documentation

Info Resolved

The README.md states that the amount in the `updatePosterFee()` function is set in Gwei, but in the contract it is set in wei.

C8dIc6 Unsupported tokens

Info Resolved

Non-standard reward tokens are not supported. ERC20 transfer of the reward tokens is not performed with the SafeERC20 or similar library. Using a rebasing token as reward tokens may result in locked tokens or reward shortage. We recommend adding explicitly to the documentation that rebase tokens are not supported.

Update

The SafeERC20 library was included and used for reward transfers. Information that that rebase token should not be used was added to the documentation.

Team response

Although we don't have programmatic control over which token will be used in the contract, we will make it clear in the documentation that non-standard reward tokens, such as rebasing tokens, should not be used with this contract.

5. Conclusion

1 high, 1 medium, 1 low severity issues were found during the audit. 1 high, 1 medium, 1 low issues were resolved in the update.

This audit includes recommendations on code improvement and the prevention of potential attacks.

Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. Issue status description

Resolved. The issue has been completely fixed.

Partially fixed. Parts of the issue have been fixed but the issue is not completely resolved.

Acknowledged. The team has been notified of the issue, no action has been taken.

Open. The issue remains unresolved.

Appendix C. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

Appendix D. Centralization risks classification

Centralization level

- **High.** The project owners can manipulate user's funds, lock user's funds on their will (reversible or irreversible), or maliciously update contracts parameters or bytecode.
- **Medium.** The project owners can modify contract's parameters to break some functions of the project contract or contracts, but user's funds remain withdrawable.
- **Low.** The contract is trustless or its governance functions are safe against a malicious owner.

Centralization risk

- **High.** Lost ownership over the project contract or contracts may result in user's losses. Contract's ownership belongs to EOA or EOAs, and their security model is unknown or out of scope.
- **Medium.** Contract's ownership is transferred to a contract with not industry-accepted parameters, or to a contract without an audit. Also includes EOA with a documented security model, which is out of scope.
- **Low.** Contract's ownership is transferred to a well-known or audited contract with industry-accepted parameters.

 contact@hashex.org

 @hashex_manager

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY