KHOOD Security Review

Version 1.0

April 9, 2024

Conducted by:
George Hunter, Independent Security Researcher

Table of Contents

1 About George

George Hunter, is an independent security researcher experienced in Solidity smart contract auditing and bug hunting. Having conducted over 50 solo and team smart contract security reviews, he consistently aims to provide top-quality security auditing services. He serves as a smart contract auditor at Paladin Blockchain Security, where he has been involved in security audits for notable clients such as LayerZero, TraderJoe, SmarDex, and other leading protocols.

For security consulting, you can contact him on Twitter, Telegram, or Discord - *@GeorgeHNTR*.

2 Disclaimer

Audits are a time, resource and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to find as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

3 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

3.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

3.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

Overview

Project Name	Kreatorhood, Airdrop contracts
Repository	https://github.com/Kreatorhood-HD/kthd-erc20
Commit hash	4f340bf6e6ef045c8038d7e2ce5c97b451e9067f
Resolution	1993e6039b4a6006d2917d7bc4bbcdedc66a06c3
Methods	Manual review

Scope

contracts/*

Issues Found

Severity	Total	Resolved	Acknowledged	Failed
Critical risk	1	1	0	0
High risk	0	0	0	0
Medium risk	0	0	0	0
Low risk	1	1	0	0
Informational	1	1	0	0
Total	3	3	0	0

5 Findings

5.1 Critical risk

5.1.1 A fee recipient can drain the KreatorhoodFeesDistributor via re-entrancy

Severity: *Critical risk*

Context: KreatorhoodFeesDistributor.sol#L83-L89

Description: The KreatorhoodFeesDistributor distributes funds to fee recipients based on a whitelist determined by a merkle tree root. The problem within its *claim* function is that it first executes the transfer of native or ERC20 tokens to the fee recipients and just then stores the replay protection storage variable which is located in the *claimed* mapping:

```
if (_claim.token == address(0)) {
    (bool success, ) = payable(_msgSender()).call{ value: _claim.amount }("");
    if (!success) revert LibMerkleDistributor.NativeTransfer(_merkleTree.root,
        _msgSender());
} else {
    IERC20(_claim.token).safeTransfer(_msgSender(), _claim.amount);
}
claimed[_merkleTree.root][_msgSender()] = 1;
```

A malicious caller may reenter the function in case native tokens or ERC777 tokens are used for the payment and that way drain the whole balance of the KreatorhoodFeesDistributor.

Recommendation: Consider following the Checks-Effects-Interactions pattern as well as applying re-entrancy guards in the distributor contracts.

Resolution: Resolved.

5.2 Low risk

5.2.1 Improper replay protection

Severity: *Low risk*

Context: KreatorhoodTokenDistributor.sol#L70-L89

Description: The *startVesting* function of KreatorhoodTokenDistributor implements the following checks to ensure it is not called more than once:

```
if (khoodToken.balanceOf(address(this)) < (AIRDROP_AMOUNT + TRADING_REWARDS_AMOUNT)
    * WEI_MULTIPLIER) {
    revert InvalidTokenAmount();
}
if (khoodToken.balanceOf(_airdropDistributor) != 0) revert AlreadyTransferred();
```

The goal is to prevent distributing *(AIRDROP_AMOUNT + TRADING_REWARDS_AMOUNT) * WEI_MULTIPLIER* tokens more than once since the rest of the contract's balance is dedicated to other KHOOD recipients such as the LP pool, airdrop distributor, rewards distributor.

There are 3 potential problems with how the replay protection is implemented:

1. An adversary can theoretically front-run the function and donate tokens to the *_airdropDistributor* effectively blocking the function.
2. The *_airdropDistributor* may spend all of its balance once the function has distributed tokens to it, and it will become callable again.
3. The contract's owner may change the address of the *_airdropDistributor* once vesting has started and call the function again.

Recommendation: Consider implementing a replay protection mechanism similar to the one in *withdrawForLP* where the sent amount is tracked in storage instead of being retrieved via *.balanceOf*.

Resolution: Resolved.

5.3 Informational

5.3.1 Typographical mistakes, code-style suggestions and non-critical issues

Severity: *Informational*

Description: The contracts contain one or more typographical issues, code-style suggestions and non-critical issues. In an effort to keep the report size reasonable, we enumerate these below:

1. KreatorhoodTreasuryWithFeeSharing can accept native tokens, but cannot withdraw them.
2. If KreatorhoodToken's *totalSupply* reaches *TOTAL_SUPPLY_CAP* and then tokens get burned, new tokens could be minted again up to the TOTAL_SUPPLY_CAP. Consider whether this is the intended behavior or the constraint should be that no more tokens than *TOTAL_SUPPLY_CAP* can *ever* be minted.
3. Consider combining *setLockAddresses* and *setVestingAddresses*.
4. *setLockAddresses*, *setVestingAddresses* and *startPhaseNewRound* can be called more than once but should not
5. Consider using OpenZeppelin's Ownable2Step instead of Ownable.
6. There is no need to apply for re-entrancy guards in KreatorhoodTreasuryWithFeeSharing as no unsafe external calls are made.
7. Consider removing the NativeMetaTransaction from RewardToken as its functionality is never used.
8. The EIP712Domain struct definition contains incorrect parameters.

Recommendation: Consider fixing the above typographical issues, code-style suggestions and non-critical issues.

Resolution: Resolved.