Key Finance Security Review

May 5th, 2023

Report Prepared By:
Georgi Georgiev (Gogo), Independent Security Researcher

Table of contents

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.

Risk classification

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

Impact

- High - leads to a significant material 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 core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behaviour, but there are no funds at risk.

Likelihood

- High - there is a direct attack vector and the cost of the attack is relatively low to the amount of funds that can be stolen or lost.
- Medium - only conditionally possible incentivized attack vector, but still relatively likely.
- Low - depends on too many or too unlikely assumptions or requires a huge stake by an attacker with little or no incentive.

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.

Executive summary

Summary

Protocol name	Key Finance
Repository	https://github.com/cryptohiveteam/key-for-gmx
Commit hash	baf04e22aa5f28b3a1baf40cfaebaddfb1f9b37d
Documentation	https://docs.gmxkey.com

Scope

• contracts/Rewards.sol

Findings

Low severity

1. Fee collection in native token does not implement the withdrawal pattern

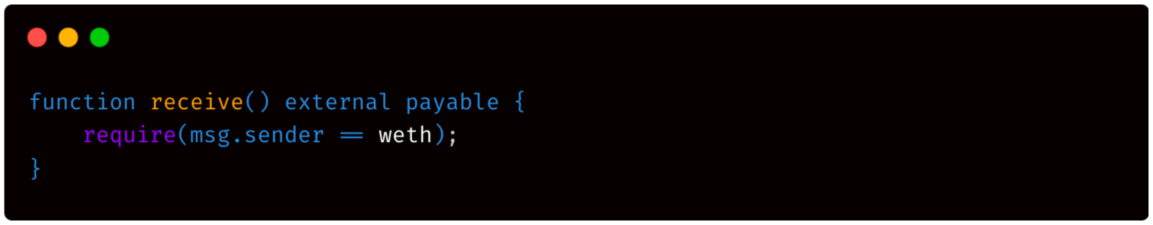
When staking rewards in WETH are claimed or updated, a certain percentage is sent to the treasury contract as a fee. This is generally considered a bad practice as the treasury contract can be changed at any time by the contract admin to a malicious one that always reverts the transaction on received native tokens. Therefore, this could result in a denial-of-service for claiming and updating WETH rewards.

Consider following the withdrawal pattern and letting the treasury address withdraw the accumulated fees instead of sending them on each claim/update.

2. Accidentally sent ether to contract will become stuck

The Rewards contract implements an empty receive() payable function to accept native tokens sent by the WETH contract when weth.withdraw is called in _transferAsETH(). As this is the only purpose of the contract for using native tokens, any excess native tokens sent to it will be accepted and stuck.

Consider adding the following check in the receive() function:



```
function receive() external payable {
    require(msg.sender == weth);
}
```

3. Privileged roles risks

The protocol uses privileged accounts to change state parameters such as feeCalculator, converter, and pause. Since the contract is responsible for managing the distribution of staking rewards, it is worth noting that the two core functionalities, claiming and updating rewards, can be paused by the admin at any time. This can be done by either calling the intended pause() function or by replacing the converter/treasury contracts with malicious ones.

The feeCalculator is an address of an external contract responsible for returning correct data for staking fees, but the returned value is never validated. Therefore, it could potentially be greater than 100%, leading to a denial-of-service for rewards claiming.

Moreover, at the time when this audit is being conducted, the admin is set to an EOA, which opens up the risk of compromised private key or malicious acts by the signer. It is recommended to consider switching to a multi-signature wallet and to ensure that users are aware of the risks of privileged roles.