MAY 2023

SECURITY AUDIT

Key Finance

Audited By :
HollaDieWaldfee

Scope

The audit has been conducted in the “key-for-gmx” repository which is private:

The commit hash at the start of the audit was:
ca1fd296812fec7048ac8b573c92a1180c328446

During the duration of the audit, the client opened a public repository (https://github.com/KeyFinanceTeam/key-finance-contracts) and new commits have been made such that the final commit was: *60cb18161e53dce1de692881cb8ae974855ecc42*

Files included in the audit:

/contracts/Staker.sol
/contracts/Rewards.sol
/contracts/TransferReceiver.sol
/contracts/Converter.sol
/contracts/FeeCalculator.sol

Severity Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	HIGH	HIGH	MEDIUM
Likelihood: Medium	HIGH	MEDIUM	LOW
Likelihood: Low	MEDIUM	LOW	LOW

Impact - the technical, economic and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability is discovered and exploited

IMPROVEMENT: Findings in this category are recommended changes that are not related to security but can improve structure, usability and overall effectiveness of the protocol.

Summary

Severity	Total	Fixed	Acknowledged	Disputed	Reported
HIGH	0	0	0	0	0
MEDIUM	1	0	1	0	0
LOW	4	4	0	0	0
IMPROVEMENT	1	1	0	0	0

#	Title	Severity	Status
1	Centralization Risk	MEDIUM	ACKNOWLEDGED
2	Missing zero address check in Rewards.sol constructor	LOW	FIXED
3	Wrong accounting of accumulated rewards if rewards from TransferReceiver are not updated in each period	LOW	FIXED
4	Use ReentrancyGuardUpgradeable	LOW	FIXED
5	Parent contracts of TransferReceiver don't have storage gaps which can lead to storage collision	LOW	FIXED
6	Use same TransferReceiver implementation for creating multiple proxies	IMPROVEMENT	FIXED

Findings

Medium Risk Findings (1)

1. Centralization Risk MEDIUM ACKNOWLEDGED

The protocol makes use of trusted roles (admin + operator) that are able to set fee percentages, pause reward updating and reward claiming.

Also the TransferReceiver is upgradeable with a time delay of at least 2 days. It means that the admin is fully trusted to handle user funds. However, users will also be able to reclaim their accounts through a buyback feature that will be developed in the future. The client acknowledged this is necessary because changes will need to be adapted as GMX upgrades occur in the future.

Low Risk Findings (4)

2. Missing zero address check in Rewards.sol constructor LOW FIXED

Description: All parameters in the `Rewards.sol` constructor are checked to not be equal to `address(0)` except for `_GMXkey`.

contracts/Rewards.sol#L65-L89

```
constructor(address _admin, IRewardRouter _rewardRouter, address _GMXkey,
address _esGMXkey, address _MPkey, address _staker, address _treasury)
Pausable(_admin) {
    require(address(_rewardRouter) != address(0), "Rewards: rewardRouter must
not be zero address");
    require(_esGMXkey != address(0), "Rewards: esGMXkey must not be zero
address");
    require(_MPkey != address(0), "Rewards: MPkey must not be zero address");
    require(_staker != address(0), "Rewards: staker must not be zero
address");
    require(_treasury != address(0), "Rewards: treasury must not be zero
address");
    stakedGmxTracker = _rewardRouter.stakedGmxTracker();
    require(stakedGmxTracker != address(0), "Rewards: stakedGmxTracker must
not be zero address");
    feeGmxTracker = _rewardRouter.feeGmxTracker();
    require(feeGmxTracker != address(0), "Rewards: feeGmxTracker must not be
zero address");
    gmx = _rewardRouter.gmx();
    require(gmx != address(0), "Rewards: gmx must not be zero address");
    esGmx = _rewardRouter.esGmx();
    require(esGmx != address(0), "Rewards: esGmx must not be zero address");
    bnGmx = _rewardRouter.bnGmx();
    require(bnGmx != address(0), "Rewards: bnGmx must not be zero address");
    GMXkey = _GMXkey;
    esGMXkey = _esGMXkey;
    MPkey = _MPkey;
    weth = _rewardRouter.weth();
    require(weth != address(0), "Rewards: weth must not be zero address");
    staker = _staker;
    treasury = _treasury;
    maxPeriodsToUpdateRewards = 4;
}
```

Recommendation: Add a zero address check for `_GMXkey`.

Fix: The zero address check has been added during the audit in the following commit: *28dcee08877a7d8334aa1f97d4ebccf8e803df7a*.

3. Wrong accounting of accumulated rewards if rewards from TransferReceiver are not updated in each period LOW FIXED

Description: When the `Rewards.updateAllRewardsForTransferReceiverAndTransferFee` function is called for a `receiver` and a new period is entered, the `_initializePeriod` function is called which applies the cumulated rewards which is performed by the `_applyCumulatedReward` function:

contracts/Rewards.sol#L464-L467

```
function _applyCumulatedReward(address stakingToken, address rewardToken,
uint256 totalShareForPrevPeriod) internal {
    rewardPerUnit[stakingToken][rewardToken][currentPeriodIndex] +=
Math.mulDiv(cumulatedReward[stakingToken][rewardToken], PRECISION,
totalShareForPrevPeriod);
    cumulatedReward[stakingToken][rewardToken] = 0;
}
```

`totalShareForPrevPeriod` is the total shares for the previous period for a given staked token (`GMXkey`, `esGMXkey`, `MPkey`).

The rewards should thus be added to the previous period. However the previous period is not necessarily equal to `currentPeriodIndex` if updating rewards for the `receiver` is skipped for one or more periods.

Thus the accumulated rewards would be added to the wrong period.

Recommendation: The accumulated rewards need to be added to the previous period which can be calculated by `block.timestamp / PERIOD - 1`.

Fix: The issue has been fixed during the audit in the following commit: *79f1f635ffb5c164019078b70f35b8719f44a432*.

The fix implements the recommendation and the accumulated rewards are now added to the previous period, not to the period when rewards were last updated.

4. Use ReentrancyGuardUpgradeable LOW FIXED

Description: The `TransferReceiver` contract inherits from `ReentrancyGuard`.

contracts/TransferReceiver.sol#L31

```
contract TransferReceiver is ITransferReceiver, Initializable,
UUPSUpgradeable, ConfigUserInitializable, ReentrancyGuard,
PausableInitializable {
```

The problem with this is that `TransferReceiver` is supposed to be an upgradeable contract and is therefore initialized by an initializer function.

However the initializer function does not initialize the `_status` variable of the `ReentrancyGuard`.

Therefore the `_status` variable starts with its default value which is `0`. This is an undefined state as states are only defined for the values `1` and `2`:

ReentrancyGuard.sol#L34-L35

```
    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;
```

Upon first use of the `ReentrancyGuard` though, the `_status` is then updated to a defined value and the issue has resolved itself:

ReentrancyGuard.sol#L56-L62

```
function _nonReentrantBefore() private {
    // On the first call to nonReentrant, _status will be _NOT_ENTERED
    require(_status != _ENTERED, "ReentrancyGuard: reentrant call");

    // Any calls to nonReentrant after this point will fail
    _status = _ENTERED;
}
```

An additional risk is the lack of a storage gap in the `ReentrancyGuard` contract which can lead to storage collisions (also see issue #5 regarding this).

Recommendation: It is recommended that `ReentrancyGuardUpgradeable` is used (https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/security/ReentrancyGuardUpgradeable.sol) which exposes the `__ReentrancyGuard_init` function. This function must then be called in `TransferReceiver.initialize`.

Fix: The issue has been fixed during the audit in the following commit: *3689490e39d31eff766bb0815ae621efec7c41d4*.

The fix implements the recommendation.

5. Parent contracts of TransferReceiver don’t have storage gaps which can lead to storage collision LOW FIXED

Description: `TransferReceiver` is an upgradeable contract and inherits from multiple parent contracts.

contracts/TransferReceiver.sol#L31

```
contract TransferReceiver is ITransferReceiver, Initializable,
UUPSUpgradeable, ConfigUserInitializable, ReentrancyGuardUpgradeable,
PausableInitializable {
```

Three of these contracts do not have a storage gap implemented.
These contracts are `ConfigUserInitializable`, `PausableInitializable` and `AdminableInitializable` (a parent contract of `PausableInitializable`).

This means that it’s not possible to add additional variables to these three contracts since it would lead to storage collisions.

Refer to the OZ documentation for more information about storage gaps:
https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#storage-gaps

Recommendation: I recommended to implement storage gaps in the three contracts mentioned above.

```
diff --git a/contracts/common/AdminableInitializable.sol
b/contracts/common/AdminableInitializable.sol
index c68d6bd..386349d 100644
--- a/contracts/common/AdminableInitializable.sol
+++ b/contracts/common/AdminableInitializable.sol
@@ -33,4 +33,6 @@ abstract contract AdminableInitializable {
         admin = candidate;
         candidate = address(0);
     }
+
+    uint256[64] private __gap;
 }
\ No newline at end of file
diff --git a/contracts/common/ConfigUserInitializable.sol
b/contracts/common/ConfigUserInitializable.sol
index 60a4037..533a4ce 100644
--- a/contracts/common/ConfigUserInitializable.sol
+++ b/contracts/common/ConfigUserInitializable.sol
@@ -10,4 +10,6 @@ contract ConfigUserInitializable {
         require(_config != address(0), "ConfigUserInitializable: config is
the zero address");
         config = _config;
     }
+
+    uint256[64] private __gap;
```

```
 }
\ No newline at end of file
diff --git a/contracts/common/PausableInitializable.sol
b/contracts/common/PausableInitializable.sol
index 1cae26a..8263604 100644
--- a/contracts/common/PausableInitializable.sol
+++ b/contracts/common/PausableInitializable.sol
@@ -29,4 +29,6 @@ abstract contract PausableInitializable is
AdminableInitializable {
         paused = false;
         emit Resumed();
     }
+
+    uint256[64] private __gap;
 }
```

Fix: The client fixed the issue during the audit in the following commit: *7d4fd9945c4840e17e7307c8ac1847966bfd9225*.

Improvement Findings (1)

6. Use same TransferReceiver implementation for creating multiple proxies IMPROVEMENT FIXED

Description: The `Converter` contract creates a new proxy for each `TransferReceiver` that is created.

It's unnecessary that there's also a new implementation contract created for each proxy:

contracts/Converter.sol#L157-L167

```
TransferReceiver transferReceiver = new TransferReceiver();
ERC1967Proxy proxy = new ERC1967Proxy(
    address(transferReceiver),

abi.encodeWithSelector(ITransferReceiver(transferReceiver).initialize.selecto
r,
    operator,
    config,
    address(this),
    rewardRouter,
    stakedGlp,
    rewards)
);
```

All the proxies can use the same implementation contract.

Recommendation: Set the `TransferReceiver` implementation contract in the constructor of the `Converter` and use it when creating new proxies.

Fix: The sponsor implemented the recommendation during the audit in the following commit: *e28a2ce779320cd37001a2d6e48416dec0e09f42*.

In addition to the recommendation the `TransferReceiver` implementation contract can also be updated by the admin via the new `setTransferReceiver` function such that new proxies get created with the new implementation.