MAY 2023

SECURITY AUDIT

Key Finance

Audited By :

HollaDieWaldfee

Audit Report - Key Finance

Audit Date	14/05/2023 - 15/05/2023
Auditor	HollaDieWaldfee (@HollaWaldfee100)
Version 1	16/05/2023 Initial Report

Contents

Disclaimer

The following smart contract audit report is based on the information and code provided by the client, and any findings or recommendations are made solely on the basis of this information. While the Auditor has exercised due care and skill in conducting the audit, it cannot be guaranteed that all issues have been identified and that there are no undiscovered errors or vulnerabilities in the code.

Furthermore, this report is not an endorsement or certification of the smart contract, and the Auditor does not assume any responsibility for any losses or damages that may result from the use of the smart contracts, either in their current form or in any modified version thereof.

About HollaDieWaldfee

HollaDieWaldfee is a top ranked Smart Contract Auditor doing audits on code4rena (www.code4rena.com) and Sherlock (www.sherlock.xyz), having ranked 1st in multiple contests. On Sherlock he uses the handle “rogueredddwarf” to compete in contests. He can also be booked for conducting Private Audits.

Contact:

Twitter: @HollaWaldfee100

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:
845efdf76d1e41abd34459f0fcebfde739a9d563

The final version of the contract has been pushed to the client’s public repository (https://github.com/KeyFinanceTeam/key-finance-contracts) at commit:
9f4bb6ebf737c01ba3886e94b0ca11addd962d3d

Files included in the audit:

/contracts/TransferSender.sol

/contracts/TransferReceiverV2.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	2	2	0	0	0
LOW	0	0	0	0	0
IMPROVEMENT	0	0	0	0	0

#	Title	Severity	Status
1	Attacker can front-run user when he locks a receiver (griefing attack)	MEDIUM	FIXED
2	Centralization risk due to admin privileges in the buy back flow	MEDIUM	FIXED

Findings

Medium Risk Findings (2)

1. Attacker can front-run user when he locks a receiver (griefing attack) MEDIUM FIXED

Description: When a user wants to perform a buy back of a GMX account, he needs to first call the TransferSender.lock function (*contracts/TransferSender.sol#L122-L153*). The receiver is then "locked" for 5 minutes in which the user needs to call the TransferSender.unwrap function (*contracts/TransferSender.sol#L155-L178*) to actually burn his Key tokens and follow through with the account transfer.

The purpose of the TransferSender.lock function is for the user to "lock" a price. This means the user has a final price before making the purchase decision.

The issue with the mechanism is that it can be abused by an attacker in a griefing attack.

The attacker can front-run the user's call to TransferSender.lock and call the function himself.

Thereby the user is not able to perform the buy back for 5 minutes or he needs to find another TransferReceiver contract to perform the buy back on.

Impact: Due to the fact that the protocol is deployed on L2, the cost of the griefing attack is minimal considering that the buy back mechanism is an important and time-critical functionality where a delay (possibly hours or days) is not acceptable.

Recommendation: An attacker should not be able to front-run a user that wants to perform a buy back unless the attacker needs to burn his own Key tokens.

Fix: The issue has been fixed in commit *5b80faf711af351a6f01bb7edb2ee17122c4d311* by implementing the TransferSender.unwrapImmediate function (in the final commit the function is called claimAndUnwrap):

```
function unwrapImmediate(address _receiver) external nonReentrant whenNotPaused {
   require(IConverter(converter).isValidReceiver(_receiver), "TransferSender: invalid receiver");
   require(ITransferReceiverV2(_receiver).version() >= 1, "TransferSender: invalid receiver
version");
   require(!isUnwrappedReceiver[_receiver], "TransferSender: already unwrapped");

   Price memory _price = _claimRewardAndGetPrice(_receiver);

   _settleFeeAndSignalTransfer(_receiver, _price);

   emit UnwrapCompleted(msg.sender, _receiver, _price);
}
```

This function can even be called when there is an active lock on the receiver. Therefore when a user calls this function an attacker can only prevent the buy back if he himself calls the function and burns his Key tokens.

This implements the recommended solution.

Note: The fix breaks the previous behavior that by calling TransferSender.lock, a user is guaranteed to be able to go through with the buy back. Also the new function does not allow the user to see the exact price of the purchase. But this is not an issue, it's just a change in behavior. The price of the purchase depends on the rewards generated by GMX which are generated slowly. So it does not pose a threat to the user if he cannot be certain of the price of the purchase. Also he can protect himself by only setting a certain allowance.

2. Centralization risk due to admin privileges in the buy back flow MEDIUM FIXED

Description: The intention of the protocol team is to build the protocol with as little trust in the admin needed as possible. This is why they implemented a 2 day delay for making critical changes to the protocol. It's therefore important that users can perform a buy back and thereby leave the protocol at any time.

The issue is that it's possible for the protocol admin to block the buy back flow immediately which makes it impossible for users to leave the protocol.

The first instance of this issue is that the admin can set the converter address without a delay:

contracts/TransferSender.sol#L97-L100

```
function setConverter(address _converter) external onlyAdmin {
  require(_converter != address(0), "TransferSender: converter is the zero address");
  converter = _converter;
}
```

The converter is used to determine which receiver is valid. Therefore the protocol admin can disable the buy back feature by setting a converter that determines all receiver contracts to be invalid.

The second instance of this issue is that the TransferSender.lock and TransferSender.unwrap functions have the whenNotPaused modifier which means that the protocol admin can arbitrarily pause the buy back feature:

contracts/TransferSender.sol#L97-L122

```
function lock(address _receiver) nonReentrant whenNotPaused external returns (Lock memory,
Price memory) {
```

contracts/TransferSender.sol#L155

```
function unwrap(address _receiver) external nonReentrant whenNotPaused {
```

Impact: The protocol admin can block the buy back flow which means that the protocol is not trustless.

Recommendation: Introduce a time delay for setting the converter and allow buy backs when the protocol is paused.

Fix: A time delay is now required to change the converter. This change has been implemented in commit *a2dc029e28ea1bb48feb6620ab38d79b839c5d34*.

In the same commit, the whenNotPaused modifiers have been removed from the affected functions in TransferSender.

In addition it is now ensured that the downstream functions in TransferReceiverV2 can also not be paused by the admin by removing the whenNotPaused modifier from the signalTransfer function and implementing a new claimAndUpdateRewardFromTransferSender function which can also not be paused. These changes have been completed in commit *071607666c809b4a8900dab321eeebec70dd99b2*.

Overall it's now not possible anymore for the protocol admin to block the buy back mechanism unless a 2 day delay passes which means that users of the protocol do not need to trust the admin role.