Key Finance Security Review

Version 1.0

July 24, 2023

Conducted by:
Georgi Georgiev (Gogo), Independent Security Researcher

Table of Contents

1 About Gogo

Georgi Georgiev, known as Gogo, is an independent security researcher specializing in Solidity smart contract auditing and bug hunting. Having conducted numerous solo and team smart contract security reviews, he always strives to deliver top-quality security auditing services. For security consulting, you can contact him on Twitter, Telegram, or Discord - *@gogotheauditor*.

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	Key Finance
Repository	https://github.com/cryptohiveteam/key-for-gmx
Commit hash	66223023c8b0cd70299a9921caf476413fe49a40
Resolution repository	https://github.com/KeyFinanceTeam/key-finance-contracts
Resolution commit	8c13a85c99371889331d29b2be424f51099aafef
Documentation	https://docs.gmxkey.com
Methods	Manual review

Scope

contracts/common/BaseMarketV2.sol
contracts/MarketV2.sol
contracts/RewardsV2.sol
contracts/StakerV2.sol
contracts/TransferReceiverV3.sol
contracts/UsdcMarketV2.sol

Issues Found

Critical risk	0
High risk	1
Medium risk	0
Low risk	1
Informational	4

5 Findings

5.1 High severity

5.1.1 An adversary can create the best bid order and DoS the MarketV2

Severity: *High Risk*

Context: MarketV2.sol

Description: One of the new features implemented in MarketV2 is that when a user creates a bid and deposits the corresponding `currency` token (i.e. GMX) into the contract, the order starts accruing rewards in WETH until it gets filled.

However, the rewards are converted to ETH and sent via a low-level call when the order gets filled, using the following methods:

```
function _settleBidReward(Order memory _order) private {
    uint256 _rewardAmount = _rewardAmountForOrder(_order);
    lastRewardsPerBidBalance[_order.id] = rewardsPerBidBalance;
    _transferAsETH(_order.maker, _rewardAmount);
}

function _transferAsETH(address to, uint256 amount) private {
    if (amount > 0) {
        IWETH(address(weth)).withdraw(amount);
        (bool success,) = to.call{value : amount}("");
        require(success, "Transfer failed");
    }
}
```

A malicious bidder can create a bid order with the best price so that every time a user wants to create an ask and `_matchOrders` is executed, the attacker's bid will be the first on the queue. Then, when `_takeOrder` is executed, which internally calls `_settleBidReward`, the `_order.maker` can simply revert in their `fallback()`/`receive()` function to revert the whole transaction, effectively blocking the `ask` functionality.

Recommendation:
To mitigate this attack vector, consider implementing one of the following solutions:
- Implement a pull-over-push method to allow bidders to withdraw their accumulated rewards in a separate transaction.
- Transfer the rewards to the order maker in WETH instead of ETH.
- Perform the low-level call in assembly to ignore the success status as well as the returned bytes data.

Status: Fixed. `_transferAsETH` makes the low-level call in inline-assembly now and ignores both the success status and returned data.

5.2 Low severity

5.2.1 claimableBidReward returns an outdated value

Severity: *Low Risk*

Context: MarketV2.sol#L125-L129

Description: The `claimableBidReward` function was added to provide information regarding what amount of WETH tokens are currently claimable (accumulated) from a bid order.

It calls the `_rewardAmountForOrder` function which uses the lastly calculated value of `rewardsPerBidBalance` This value is updated when an order is executed or canceled.

However, any change in `rewardsPerBidBalance` since the last update are not considered. As a result, the `claimableBidReward` function will return lower than the actual reward amount at the given timestamp.

Recommendation: Consider calculating and using the new value of `rewardsPerBidBalance` in `claimableBidReward` to provide the most up-to-date value.

Status: Fixed.

5.3 Informational

5.3.1 Storage variables can be marked immutable to save gas

Severity: *Informational*

Context: MarketV2.sol#L26-L33

Description: The following state variables in MarketV2 can be marked as immutable: `rewardRouter`, `stakedGmxTracker`, `weth`, `staker`.

Recommendation: Consider making the aforementioned variables immutable.

Status: Fixed.

5.3.2 Redundant modification of an existing method

Severity: *Informational*

Context: MarketV2.sol#L289

Description: `_transferTokensMaker` was unnecessarily modified to accept the `taker` address as an input parameter, while it is still always the same as `msg.sender`.

Recommendation: Consider reverting the aforementioned change.

Status: Fixed.

5.3.3 Code duplication

Severity: *Informational*

Context: MarketV2.sol#L376

Description: On line 376 in MarketV2, the remaining amount to fill is calculated as `_order.amount - _order.filledAmount`. However, there is already a view function implemented for this purpose - `_getRemainingAmount`.

Recommendation: Consider reverting the aforementioned change.

Status: Fixed.

5.3.4 Missing re-entrancy guard

Severity: *Informational*

Context: StakerV2.sol#L158

Description: The `nonReentrant` modifier is applied to all main functions in StakerV2, except for `stakeAndLock`.

Recommendation: Consider maintaining consistency by either removing the other re-entrancy guards or adding one to `stakeAndLock`.

Status: Fixed.