Kim Tokens

Security Review

Cantina Managed review by:
Deadrosesxyz, Security Researcher

May 10, 2024

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

KIM is a decentralized exchange (DEX) protocol that uses a mathematical formula to price assets, facilitating trading without needing a traditional order book.

From May 6th to May 9th the Cantina team conducted a review of kim-tokens on commit hash 140fb7c7. The team identified a total of **8** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 4
- Gas Optimizations: 0
- Informational: 4

3 Findings

3.1 Low risk

3.1.1 Users can avoid deallocation fees by starting and cancelling a `redeem`

Severity: Medium risk

Context: XKimToken.sol#L557

Description: Within the `XKimToken` contract, admins can set a `deallocation` fee which users have to pay when deallocating.

```
balance.allocatedAmount = balance.allocatedAmount.sub(amount);
_transfer(
    address(this),
    userAddress,
    amount.sub(deallocationFeeAmount)
);
// burn corresponding Kim and XKim
KimToken.burn(deallocationFeeAmount);
_burn(address(this), deallocationFeeAmount);
```

However, this fee is not charged, when users `deallocate` within the `cancelRedeem` function.

This allows users to bypass the deallocation fee by simply starting a `redeem` and later cancelling it, instead of using the `allocate` and `deallocate` functions.

Recommendation: Charge deallocation fees upon cancelled redeems.

3.1.2 Protocol should keep `address(0)` in the transfer whitelist at all times in order to allow `redeems`

Severity: Low risk

Context: XKimToken.sol#L826

Description: When users finalize their `redeem`, it burns their `XKim` tokens. The `_burn` method invokes `_beforeTokenTransfer(account, address(0), amount);`

Since the user will not be whitelisted, the only way the transaction does not revert is if `address(0)` is kept permanently on the transfer whitelist..

Recommendation:

```
  function _beforeTokenTransfer(
      address from,
      address to,
      uint256 /*amount*/
  ) internal view override {
      require(
          from == address(0) ||
+          to == address(0) ||
             _transferWhitelist.contains(from) ||
             _transferWhitelist.contains(to),
          "transfer: not allowed"
      );
  }
```

3.1.3 Protocol whitelist mechanism allows for some slight bypassing.

Severity: Low risk

Context: XKimToken.sol#L826

Description: The way how `whitelist` mechanism works within `XKim` is that it requires at least either the sender or the recipient is whitelisted (meaning that a non-whitelisted sender can transfer to a whitelisted receiver and vice-versa).

Meaning that if user A wants to send user B tokens and both of them are not whitelisted, they can bypass this restriction as long as they find a whitelisted user X and do the following 2 transfers:

1. A transfers tokens to X.
2. X transfers the tokens to B.

Recommendation: Code would need major refactoring in order to fix this issue.

3.1.4 Protocol owner might cause accidental `redeem` DoS

Severity: Low risk

Context: XKimToken.sol#L298

Description: If we look at the code of `redeem`, we'll see that as long as `dividendsAllocation` has a non-zero value, a call to `dividendsAddress` will be made.

The problem is, that `dividendsAddress` could be set to `address(0)` (in case owner decides to disable dividends allocation on redeems).

Although, this would usually reset `redeemDividendsAdjustment` to 0, it would not later prevent owner from changing its value back to a non-zero value, allowing them to brick all future redeems

Recommendation: Add the following check to `updateRedeemSettings`

```
if (dividendsAddress != address(0)) redeemDividendsAdjustment = redeemDividendsAdjustment_;
```

3.2 Informational

3.2.1 Centralization risks

Severity: Informational

Context: Global scope

Description: Currently, the protocol admins have a lot of privileged rights, allowing them to change crucial contract parameters at any time. This includes:

- Changing `minRedeemRatio` and `maxRedeemRatio`.
- Changing `minRedeemDuration_` and `maxRedeemDuration_`.
- Changing `dividendsAddress` to any address they'd like, allowing them to block `redeems`.
- Chaning `Kim` token's `emissionRate`, `maxSupply` and `treasury/masterShare` allocation ratio.

It is expected that protocol admins will behave in best interest of the protocol. However, users must be aware that all parameters mentioned above could be changed by an admin at any time.

3.2.2 Unnecessary usage of `safeTransferFrom` as the used token is known to have no weird behaviour

Severity: Informational

Context: XKimToken.sol#L706, XKimToken.sol#L723

Description: When `Kim` tokens are converted to `XKim`, the contract pulls them with a `safeTransferFrom`. This is unnecessary, as the `Kim` token is known to be a regular ERC20.

Recommendation: Use regular `transfer` methods.

3.2.3 Typo in the function comments

Severity: Informational

Context: KimToken.sol#L386

Description: If we look at the comment regarding the used function selectors, we'll see the following:

```
bytes4(keccak256(bytes('withdraw(uint256, address, uint256)')));
```

However, the used selector actually (correctly) responds to:

```
bytes4(keccak256(bytes('withdraw(uint256,address,uint256)')));
```

Recommendation: Fix the comment to:

```
// bytes4(keccak256(bytes('withdraw(uint256,address,uint256)')));
```

3.2.4 Unnecessary variable used

Severity: Informational

Context: KimToken.sol#L269

Description: Within `KimToken#updateAllocations`, unnecessary `totalAllocationsSet` variable is used. Protocol can simply remove it, to make the code cleaner and increase its readability:

```
function updateAllocations(
    uint256 farmingAllocation_
) external onlyOwner {
    // apply emissions before changes
    emitAllocations();

    // total sum of allocations can't be > 100%
    uint256 totalAllocationsSet = farmingAllocation_;
    require(
        totalAllocationsSet <= 100,
        "updateAllocations: total allocation is too high"
    );

    // set new allocations
    farmingAllocation = farmingAllocation_;

    emit UpdateAllocations(
        farmingAllocation_,
        treasuryAllocation()
    );
}
```

Recommendation: Change the function's code to the following

```
function updateAllocations(
    uint256 farmingAllocation_
) external onlyOwner {
    // apply emissions before changes
    emitAllocations();

    require(
        farmingAllocation_<= 100,
        "updateAllocations: total allocation is too high"
    );

    // set new allocations
    farmingAllocation = farmingAllocation_;

    emit UpdateAllocations(
        farmingAllocation_,
        treasuryAllocation()
    );
}
```