SALUS SECURITY

MAY 2024

CODE SECURITY ASSESSMENT

KRAV

Overview

Project Summary

- Name: Krav
- Platform: EVM-compatible chains
- Language: Solidity
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Krav
Version	v1
Type	Solidity
Dates	May 29 2024
Logs	May 29 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	3
Total Low-Severity issues	2
Total informational issues	2
Total	7

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Chainlink's feed might return stale or incorrect results	Medium	Business Logic	Pending
2	User will have to wait for the cold period again if the user misses the time to withdraw	Medium	Business Logic	Pending
3	CoolPeriod might be bypassed	Medium	Business Logic	Pending
4	Missing duplicate value check	Low	Data Validation	Pending
5	Improper LINK fee for NFT order execution	Low	Business Logic	Pending
6	Redundant code	Informational	Redundancy	Pending
7	Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()	Informational	Risky External Calls	Pending

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Chainlink's feed might return stale or incorrect results

Severity: Medium	Category: Business Logic

Target:
- contracts/PriceAggregator.sol

Description

The ChainlinkAdapter calls out to a Chainlink oracle receiving the latestRoundData(). If there is a problem with Chainlink starting a new round and finding consensus on the new value for the oracle (e.g. Chainlink nodes abandon the oracle, chain congestion, vulnerability/attacks on the chainlink system) consumers of this contract may continue using outdated stale or incorrect data (if oracles are unable to submit no new round is started).

contracts/PriceAggregator.sol:L151-L213

```
function fulfill(bytes32 _requestId, uint _price) external
recordChainlinkFulfillment(_requestId){
        ...
        uint feedPrice = _price;

        IPairsStorage.Feed memory f = pairsStorage.pairFeed(r.pairIndex);
        if (f.feed1 != address(0)) {
                (, int feedPrice1, , , ) = IChainlinkFeed(f.feed1).latestRoundData();

            if(f.feedCalculation == IPairsStorage.FeedCalculation.DEFAULT){
                feedPrice = uint(feedPrice1*int(PRECISION)/1e8);
            }else if(f.feedCalculation == IPairsStorage.FeedCalculation.INVERT){
                feedPrice = uint(int(PRECISION)*1e8/feedPrice1);
            }else{
                (, int feedPrice2, , , ) = IChainlinkFeed(f.feed2).latestRoundData();
                feedPrice = uint(feedPrice1*int(PRECISION)/feedPrice2);
            }
        }

        uint priceDiff = _price >= feedPrice ? (_price - feedPrice) : (feedPrice -
_price);
        if(priceDiff * PRECISION * 100 / feedPrice <= f.maxDeviationP)
```

Recommendation

```
(uint80 roundId, int feedPrice1, , uint updateTime, uint80 answeredInRound ) =
AggregatorV3Interface(XXXXX).latestRoundData();
require(updateTime != 0, "Incomplete round");
require(answeredInRound >= roundId, "Stale price");
```

Consider adding checks of other returned variables from chainlink. This will help you understand the state of the chainlink and decide whether to accept the resulting price value or not.

2. User will have to wait for the cold period again if the user misses the time to withdraw

Severity: Medium	Category: Business Logic
Target: - contracts/KToken.sol	

Description

When a user creates a request for withdrawal of funds, the contract records the epoch (with cooldown period) in which the user can receive his funds. An epoch is 20 minutes. The problem is that if the user misses 20 minutes (or the attacker clogs the block with other transactions so that the user's transaction is not included in the block), then in the next epoch the user will no longer be able to receive his funds, but is forced to make a request again and wait cooldown period again.

contracts/KToken.sol:L155-L164

```
function makeWithdrawRequest(uint assets) public checks(assets) {
    require(assets <= maxWithdraw(_msgSender()), "ERC4626: withdraw more than max");

    (uint256 currentEpoch,) = this.updateEpoch();
    uint shares = previewWithdraw(assets);
    reqWithdrawals[_msgSender()][currentEpoch+coolPeriod] = shares;

    emit WithdrawRequested(_msgSender(), assets);
}
```

contract/KToken.sol:L175-L183

```
function withdraw(uint assets, address receiver, address _owner) public override
checks(assets) returns (uint) {
    uint shares = previewWithdraw(assets);
    (uint256 currentEpoch,) = this.updateEpoch();
    require(reqWithdrawals[_msgSender()][currentEpoch] == shares, "request first");
    reqWithdrawals[_msgSender()][currentEpoch] = 0;

    _withdraw(_msgSender(), receiver, _owner, assets, shares);
    return shares;
}
```

Recommendation

Consider allowing the user to receive their funds at any time, after passing the cold period.

3. CoolPeriod might be bypassed

Severity: Medium	Category: Business Logic
Target: - contracts/KToken.sol	

Description

Users can request a withdrawal through the makeWithdrawRequest() or makeRedeemRequest() functions and settle the withdrawal after the cool period. However, shares that have been requested to withdraw will not be recorded or locked. Thus, users can make a request every epoch with the same shares.

contracts/KToken.sol:L155-L164

```
function makeWithdrawRequest(uint assets) public checks(assets) {
    require(assets <= maxWithdraw(_msgSender()), "ERC4626: withdraw more than max");

    (uint256 currentEpoch,) = this.updateEpoch();
    uint shares = previewWithdraw(assets);
    reqWithdrawals[_msgSender()][currentEpoch+coolPeriod] = shares;

    emit WithdrawRequested(_msgSender(), assets);
}
```

contract/KToken.sol:L175-L183

```
function withdraw(uint assets, address receiver, address _owner) public override
checks(assets) returns (uint) {
    uint shares = previewWithdraw(assets);
    (uint256 currentEpoch,) = this.updateEpoch();
    require(reqWithdrawals[_msgSender()][currentEpoch] == shares, "request first");
    reqWithdrawals[_msgSender()][currentEpoch] = 0;

    _withdraw(_msgSender(), receiver, _owner, assets, shares);
    return shares;
}
```

Here is a possible scenario (the cool period lasts for 2 epochs):

1. Alice owns 1 share and makes withdrawal requests for 1 share at epoch 1 and 2;
2. Alice settles the withdrawal for 1 share at epoch 3;
3. Bob also wants to withdraw 1 share and transfer his share to Alice. Alice withdraws Bob's share at epoch 4. Thus, Bob's cool period is bypassed.

Recommendation

Consider recording or locking shares that have been requested to withdraw.

4. Missing duplicate value check

Severity: Low	Category: Data Validation
Target: - contracts/PriceAggregator.sol	

Description

The addNode function will check if the node exists, But the replaceNode function does not check, which conflicts with the mechanism of the addNode function.

contracts/PriceAggregator.sol:L91-L107

```
function addNode(address _a) external onlyGov{
    require(_a != address(0), "VALUE_0");
    require(nodes.length < MAX_ORACLE_NODES, "MAX_ORACLE_NODES");
    for(uint i = 0; i < nodes.length; i++){ require(nodes[i] != _a, "ALREADY_LISTED"); }

    nodes.push(_a);

    emit NodeAdded(nodes.length-1, _a);
}
function replaceNode(uint _index, address _a) external onlyGov{
    require(_index < nodes.length, "WRONG_INDEX");
    require(_a != address(0), "VALUE_0");
    emit NodeReplaced(_index, nodes[_index], _a);

    nodes[_index] = _a;
}
```

Recommendation

Consider adding duplicate value detection.

5. Improper LINK fee for NFT order execution

Severity: Low	Category: Business Logic

Target:
- contracts/Trading.sol
- contracts/TradingStorage.sol

Description

Triggers need to pay LINK fees for every NFT order execution.

contracts/Trading.sol:L333-L406

```
function executeNftOrder(
    IStorage.LimitOrder _orderType,
    address _trader,
    uint _pairIndex,
    uint _index
) external notContract notDone{
    ...
    // link transfer
    storageT.transferLinkToAggregator(msg.sender);
    uint orderId = aggregator.getPrice(
        _pairIndex,
        _orderType == IStorage.LimitOrder.OPEN ?
            IAggregator.OrderType.LIMIT_OPEN :
            IAggregator.OrderType.LIMIT_CLOSE
    );
    ...
}
```

The actual LINK fee spent in a single price request is linkFee() * nodes.length.

contracts/PriceAggregator.sol:L118-L148

```
function getPrice(
        uint _pairIndex,
        OrderType _orderType
) external onlyTrading returns(uint){
    ...
    uint linkFeePerNode = linkFee();
    ...
    for(uint i = 0; i < nodes.length; i ++){
        orderIdByRequest[sendChainlinkRequestTo(nodes[i], linkRequest, linkFeePerNode)] =
orderId;
    }
```

However, triggers only pay linkFee() for one node.

contracts/TradingStorage.sol:L391-L393

```
function transferLinkToAggregator(address _from) external onlyTrading{
        linkErc677.transferFrom(_from, address(priceAggregator),
priceAggregator.linkFee());
}
```

Recommendation

Consider updating the logic in the transferLinkToAggregator() function so that triggers pay for enough LINK.

2.3 Informational Findings

6. Redundant code

Severity: Informational	Category: Redundancy

Target:
- contracts/PairInfos.sol
- contracts/PriceAggregator.sol
- contracts/Trading.sol
- contracts/TradingCallbacks.sol
- contracts/TriggerInfo.sol

Description

The following events are defined but not used.

contracts/PriceAggregator.sol:L42

```
event AddressUpdated(string name, address a);
```

contracts/TradingCallbacks.sol:L66-L67

```
event AddressUpdated(string name, address a);
event NumberUpdated(string name, uint value);
```

contracts/TriggerInfo.sol:L36

```
event TokensClaimed(address bot, uint tokens);
```

contracts/Trading.sol:L34, L39,

```
event AddressUpdated(string name, address a);
event TriggerOrderSameBlock(address nftHolder, address trader, uint pairIndex);
```

Since the storeOpenLimitOrder() function in the TradingStorage contract will also call the firstEmptyOpenLimitIndex() function, the index query in the Trading contract can be safely removed.

contracts/Trading.sol:L147

```
function openTrade(
    IStorage.Trade memory t,
    ITriggerInfo.OpenLimitOrderType _type,
    uint _spreadReductionId,
    uint _slippageP          // for market orders
) external notDone {
    ...
    if(_type != ITriggerInfo.OpenLimitOrderType.LEGACY) {
        uint index = storageT.firstEmptyOpenLimitIndex(msg.sender, t.pairIndex);
        storageT.storeOpenLimitOrder(
            IStorage.OpenLimitOrder(
                msg.sender,
                t.pairIndex,
                index,
                t.positionSizeDai,
                ...
```

```
        )
    ...
}
```

Unused file.
contracts/PairInfos.sol:L6

```
import "hardhat/console.sol";
```

Recommendation

Consider removing the redundant code.

7. Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()

Severity: Low	Category: Risky External Calls
Target: - contracts/TradingStorage.sol	

Description

Tokens not compliant with the ERC20 specification could return false from the transfer function call to indicate the transfer fails, while the calling contract would not notice the failure if the return value is not checked. Checking the return value is a requirement, as written in the EIP-20 specification:

```
Callers MUST handle false from returns (bool success). Callers MUST NOT assume that
false is never returned!
```

Recommendation

Consider using the SafeERC20 library implementation from OpenZeppelin and call safeTransfer or safeTransferFrom when transferring ERC20 tokens.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
contracts/KToken.sol	b76f303c5e72e101fa941fc743a67a8b0319a69a
contracts/PairInfos.sol	e820e2c88ed8ab4cbc9f08b5482fa6b8dfa1b41e
contracts/PairStorage.sol	517a949cdc77b636131a3b11dc5d2cc97888ab3f
contracts/PriceAggregator.sol	464ff09475689e533b98df7fea5d7016f937d5fb
contracts/Trading.sol	86c33d2d0967123aa3d121170268eec0c57cea15
contracts/TradingCallbacks.sol	6b73de2aa638d33c183661cd38af24ec65f17bfc
contracts/TradingStorage.sol	224c26adf40a3f834ad5562a20f9cf34808c85dd
contracts/TriggerInfo.sol	22e906f56da80d4c4cb9dc69bb5c98114a0222d7