SALUS SECURITY

JULY 2023

CODE SECURITY ASSESSMENT

KAIZEN

Overview

Project Summary

- Name: Kaizen - Kaizen Corp (KZN) Token
- Address: 0x3e9fbb8c168dF1f6cfAA372e5bF7cE5F162A7617
- Platform: Ethereum
- Language: Solidity
- Audit Scope: See Appendix - 1

Project Dashboard

Application Summary

Name	Kaizen - Kaizen Corp (KZN) Token
Version	v1
Type	Solidity
Dates	July 21 2023
Logs	July 21 2023

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	2
Total Low-Severity issues	4
Total informational issues	7
Total	13

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	DoS vulnerability when creating pair contract	Medium	Denial of Service	Pending
2	Centralization risk	Medium	Centralization	Pending
3	Inconsistent units	Low	Code Consistency	Pending
4	Incomplete permission transfer	Low	Business Logic	Pending
5	Check for ‘success’ is missing	Low	Data Validation	Pending
6	Replacing ‘transfer’ method with ‘safeTransfer’ method	Low	Business Logic	Pending
7	Malicious bot can lead to regular users being unable to make transfers	Informational	Business Logic	Pending
8	Incorrect variable name	Informational	Code quality	Pending
9	Missing events	Informational	Logging	Pending
10	Spelling error	Informational	Code quality	Pending
11	Can use local variables to save gas	Informational	Gas Optimization	Pending
12	Redundant code	Informational	Redundancy	Pending
13	Missing zero address check	Informational	Data Validation	Pending

2.2 Notable Findings

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

1. DoS vulnerability when creating pair contract

Severity: Medium	Category: Denial of Service
Target: - KZN.sol	

Description

The launch() function creates the pair through the factory contract. If the pair is already deployed, launch() will fail.

KZN.sol:L1217

```
function launch(address[] memory wallets, uint256[] memory amountsInTokens, uint256
blocksForPenalty) external onlyOwner {
    ...
    lpPair = IDexFactory(_dexRouter.factory()).createPair(address(this),
_dexRouter.WETH());
    ...
}
```

Recommendation

It is recommended to check whether the pair already exists before attempting to create it. One possible approach is to use the getPair() in factory contract to verify if the pair already exists. If the pair does not exist, then proceed with the createPair().

2. Centralization risk

Severity: Medium	Category: Centralization
Target: - KZN.sol	

Description

In the KZA token, there is a privileged Owner role. This role has the ability to:

- remove/enable limits.
- disable transfer delay.
- disable flagBots forever.
- add/remove early buyer.
- exclude/include a user from the maximum transaction limit.
- update the minimum tokens required before swapping.
- update the maximum transfer amount.
- set the automated market maker pair.
- exclude/include the address from receiving rewards.
- force swap back.
- exclude/include the address from transfer fee.
- set sell/buy fee
- set operations address
- set the swap and liquify feature as enabled or disabled.
- withdraw stuck ETH
- Launch KZA Token

And most importantly, the Owner can set the transaction fee to 100% to receive all tokens from the traders.

```
function setBuyAndSellFees(uint256 buyFee, uint256 sellFee)
    external
    onlyOwner
{
     require(highTaxModeEnabled, "High tax mode disabled for ever!");

    _buyTaxFee = 100;
    _buyLiquidityFee = 0;
    _buyOperationsFee = buyFee;

    _sellTaxFee = 100;
    _sellLiquidityFee = 0;
    _sellOperationsFee = sellFee;
}
```

Recommendation

Consider setting an upper limit for the fees configured in the setBuyAndSellFees() function.

For Example:

```
function setBuyAndSellFees(uint256 buyFee, uint256 sellFee)
    external
    onlyOwner
{
    ...
    _buyTaxFee = 100;
    _buyLiquidityFee = 0;
    _buyOperationsFee = buyFee;
    require(_buyTaxFee + _buyLiquidityFee + _buyOperationsFee <= 3000, "Must keep buy
taxes below 30%");

    _sellTaxFee = 100;
    _sellLiquidityFee = 0;
    _sellOperationsFee = sellFee;
    require(_sellTaxFee + _sellLiquidityFee + _sellOperationsFee <= 3000, "Must keep
sell taxes below 30%");
}
```

And we recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

3. Inconsistent units

Severity: Low	Category: Code Consistency
Target: - KZN.sol	

Description

The updateMinimumTokensBeforeSwap and updateMaxAmount functions pass the parameter newAmount and newNum in different units. This inconsistency may lead to confusion when interacting with the contract.

KZN.sol:L594-L603

```
function updateMinimumTokensBeforeSwap(uint256 newAmount) external onlyOwner{
    require(newAmount >= _tTotal * 1 / 100000, "Swap amount cannot be lower than 0.001%
total supply.");
    require(newAmount <= _tTotal * 5 / 1000, "Swap amount cannot be higher than 0.5%
total supply.");
    minimumTokensBeforeSwap = newAmount;
}

function updateMaxAmount(uint256 newNum) external onlyOwner {
    require(newNum >= (_tTotal * 2 / 1000)/1e18, "Cannot set maxTransactionAmount lower
than 0.2%");
    maxTransactionAmount = newNum * (1e18);
}
```

Recommendation

It is recommended to use the same token amount unit(e.g., wei) for both functions to ensure clarity and consistency.

4. Incomplete permission transfer

Severity: Low	Category: Business Logic
Target: - KZN.sol	

Description

In the constructor, both the owner and operationsAddress are assigned specific permissions, and allocating a significant portion of the tokens to the owner.

However, when transferring ownership using the transferOwnership() function inherited from the Ownable contract, only the owner address is changed. Thus the new owner does not inherit the permissions granted during contract deployment, and the original owner's permissions remain active even after the ownership transfer.

For operationsAddress, it is set to exclude from MaxTransaction in the constructor. However, excludeFromMaxTransaction() is not called in the setOperationsAddress().

KZN.sol:L238-L245

```
function transferOwnership(address newOwner) external virtual onlyOwner {
    require(
        newOwner != address(0),
        "Ownable: new owner is the zero address"
    );
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
}
```

KZN.sol:L1150-L1156

```
function setOperationsAddress(address _operationsAddress) external onlyOwner {
    require(_operationsAddress != address(0), "_operationsAddress address cannot be 0");
    _isExcludedFromFee[operationsAddress] = false;
    operationsAddress = payable(_operationsAddress);
    _isExcludedFromFee[operationsAddress] = true;
    emit UpdatedOperationsAddress(_operationsAddress);
}
```

Recommendation

When updating ownership and operationsAddress, it is recommended to transfer all corresponding permissions as well.

5. Check for ‘success’ is missing

Severity: Low	Category: Data Validation
Target: - KZN.sol	

Description

The swapBack() and withdrawStuckETH() functions could be success even if transferring ETH is failed.

KZN.sol L825

```
(success,) = address(operationsAddress).call{value: address(this).balance}("");
```

KZN.sol L1187

```
(success,) = address(msg.sender).call{value: address(this).balance}("");
```

Recommendation

Consider adding a check for ‘success’.

6. Replacing ‘transfer’ method with ‘safeTransfer’ method

Severity: Low	Category: Business Logic
Target: - KZN.sol	

Description

In this contract, the `transferForeignToken` function uses the `transfer` method to transfer ERC20 tokens, but there may be potential vulnerabilities during the transfer process, resulting in asset transfer failure or asset loss.

The `safeTransfer` method is a safe transfer function, which adds a check on the result of the transfer process and will revert when the transfer fails to avoid possible loss of funds. So it is better to use the `safeTransfer` method.

KZN.sol:L1174

```
function transferForeignToken(address _token, address _to)
        external
        onlyOwner
        returns (bool _sent)
    {
        ...
        _sent = IERC20(_token).transfer(_to, _contractBalance);
        ...
}
```

Recommendation

Consider utilizing the secure `safeTransfer` method instead of the conventional `transfer` to seamlessly execute the transfer of ERC20 tokens.

2.3 Informational Findings

7. Malicious bot can lead to regular users being unable to make transfers

Severity: Informational	Category: Business Logic
Target: - KZN.sol	

Description

To prevent bots from entering and profiting during the early stages of the project, the project team designed a condition in the 'transfer' function to restrict bot transactions during normal trading periods

KZN.sol:L694

```
function _transfer(
    address from,
    address to,
    uint256 amount
) private {
    ...
    if(!earlyBuyPenaltyInEffect()){
        require(!boughtEarly[from] || to == owner() || to == address(0xdead), "Bots
cannot transfer tokens in or out except to owner or dead address.");
    }
}
```

However, since bots can still perform transfers during the early stage, they can call the transfer function to send tokens to regular user addresses, accidentally marking regular users as boughtEarly(bot). This will result in regular users being unable to make transfers during normal trading periods.

KZN.sol:L748-L760

```
function _transfer(
    address from,
    address to,
    uint256 amount
) private {
    ...
    if(earlyBuyPenaltyInEffect() && automatedMarketMakerPairs[from] &&
!automatedMarketMakerPairs[to]){

        if(!boughtEarly[to]){
            boughtEarly[to] = true;
            botsCaught += 1;
            emit CaughtEarlyBuyer(to);
        }
    }
    ...
}
```

Recommendation

Consider prohibiting bots from trading at any time.

8. Incorrect variable name

Severity: Informational	Category: Code quality
Target: - KZN.sol	

Description

KZN.sol:L309

```
mapping(address => uint256) private _holderLastTransferTimestamp;
```

This variable is used to store the timestamp of the holder's last transfer during launch, but in the contract, use the content of this variable to compare with blocknumber.

Recommendation

Consider using appropriate variable names.

9. Missing events

Severity: Informational	Category: Logging
Target: - KZN.sol	

Description

Important parameter or configuration changes should trigger an event to enable tracking off-chain, but functions mentioned below changing the important parameters do not emit events.

KZN.sol:L571-L583

```
function addBoughtEarly(address wallet) external onlyOwner {
        ...
        boughtEarly[wallet] = true;
    }

function removeBoughtEarly(address wallet) external onlyOwner {
        ...
        boughtEarly[wallet] = false;
}
```

KZN.sol:L594-L609

```
function updateMinimumTokensBeforeSwap(uint256 newAmount) external onlyOwner{
           ...
           minimumTokensBeforeSwap = newAmount;
       }

function updateMaxAmount(uint256 newNum) external onlyOwner {
        ...
        maxTransactionAmount = newNum * (1e18);
}

function setAutomatedMarketMakerPair(address pair, bool value) public onlyOwner {
        ...

        _setAutomatedMarketMakerPair(pair, value);
}
```

KZN.sol:L646-L667

```
function excludeFromReward(address account) public onlyOwner {
        ...
        _isExcluded[account] = true;
        _excluded.push(account);
    }

function includeInReward(address account) public onlyOwner {
        require(_isExcluded[account], "Account is not excluded");
```

```
        for (uint256 i = 0; i < _excluded.length; i++) {
            if (_excluded[i] == account) {
                ...
                _isExcluded[account] = false;
                _excluded.pop();
                break;
            }
        }
}
```

KZN.sol:L1135-L1148

```
function setBuyAndSellFees(uint256 buyFee, uint256 sellFee)
        external
        onlyOwner
    {
         require(highTaxModeEnabled, "High tax mode disabled for ever!");

        _buyTaxFee = 100;
        _buyLiquidityFee = 0;
        _buyOperationsFee = buyFee;

        _sellTaxFee = 100;
        _sellLiquidityFee = 0;
        _sellOperationsFee = sellFee;
}
```

Recommendation

Design proper events and add them to the aforementioned functions.

10. Spelling error

Severity: Informational	Category: Code quality
Target: - KZN.sol	

Description

KZN.sol:L361

```
// store addresses that a automatic market maker pairs. Any transfer *to* these
addresses
```

"a automatic" should be "an automatic".

Recommendation

Consider correcting the spelling error.

11. Can use local variables to save gas

Severity: Informational	Category: Gas Optimization
Target: - KZN.sol	

Description

1. _tTotal is read twice from storage. Could have a local variable.

   KZN.sol L595 - L596

   ```
   require(newAmount >= _tTotal * 1 / 100000, "Swap amount cannot be lower than
   0.001% total supply.");
   require(newAmount <= _tTotal * 5 / 1000, "Swap amount cannot be higher than 0.5%
   total supply.");
   ```

2. _excluded.length is read every time during iteration. Could have a local variable for the length.

   KZN.sol L658- L666

   ```
   for (uint256 i = 0; i < _excluded.length; i++) {
           if (_excluded[i] == account) {
               _excluded[i] = _excluded[_excluded.length - 1];
               _tOwned[account] = 0;
               _isExcluded[account] = false;
               _excluded.pop();
               break;
           }
   }
   ```

3. operationsAddress could be replaced with _operationsAddress.

   KZN.sol L1154

   ```
   _isExcludedFromFee[operationsAddress] = true;
   ```

Recommendation

Consider using local variables instead of these state variables.

12. Redundant code

Severity: Informational	Category: Redundancy

Target:
- KZN.sol

Description

KZN.sol:L433-L442

```
constructor() payable {
    //...
    operationsAddress = payable(msg.sender); // Operations Address

    _isExcludedFromFee[owner()] = true;
    _isExcludedFromFee[address(this)] = true;
    _isExcludedFromFee[operationsAddress] = true;

    excludeFromMaxTransaction(owner(), true);
    excludeFromMaxTransaction(address(this), true);
    excludeFromMaxTransaction(address(0xdead), true);
    excludeFromMaxTransaction(operationsAddress, true);
    //...
}
```

In the constructor, the owner and operationsAddress are the same address, so there is a code redundancy and msg.sender is recommended instead.

KZN.sol:L748-L770

```
function _transfer(
    address from,
    address to,
    uint256 amount
) private {
    //...
    if(earlyBuyPenaltyInEffect() && automatedMarketMakerPairs[from] &&
!automatedMarketMakerPairs[to]){

        if(!boughtEarly[to]){
            boughtEarly[to] = true;
            botsCaught += 1;
            emit CaughtEarlyBuyer(to);
        }

        _taxFee = _buyTaxFee;
        _liquidityFee = _buyLiquidityFee + _buyOperationsFee;
        if(_liquidityFee > 0){
            buyOrSellSwitch = BUY;
        }
    }

        // Buy
    if (automatedMarketMakerPairs[from]) {
        _taxFee = _buyTaxFee;
        _liquidityFee = _buyLiquidityFee + _buyOperationsFee;
        if(_liquidityFee > 0){
            buyOrSellSwitch = BUY;
        }
```

```
    }
    //...
}
```

In the above code in the _transfer(), if the first if-condition is met, a duplicate piece of code will be executed twice. It is recommended to remove the duplicate code in the first if-condition.

KZN.sol:L802

```
function swapBack() private lockTheSwap {
    //...
    if(contractBalance == 0 || totalTokensToSwap == 0) {return;}
    //...
}
```

The check of contractBalance is redundant in swapBack(). Because when swapback() is called, it satisfies the "contractBalance >= minimumTokensBeforeSwap" condition.

KZN.sol:L1218-L1219

```
function launch(address[] memory wallets, uint256[] memory amountsInTokens, uint256
blocksForPenalty) external onlyOwner {
    //...
    excludeFromMaxTransaction(address(lpPair), true);
    _setAutomatedMarketMakerPair(address(lpPair), true);
    //...
}
```

In launch(), the calling to excludeFromMaxTransaction() is redundant, because the _setAutomatedMarketMakerPair() includes the functionality of excludeFromMaxTransaction().

Recommendation

Consider removing the redundant code as suggestions.

13. Missing zero address check

Severity: Informational	Category: Data Validation
Target: - KZN.sol	

Description

It is considered a security best practice to verify addresses against the zero address in the constructor or setting. However, this precautionary step is absent for the variables highlighted below.
KZN.sol:L1166-L1176

```
function transferForeignToken(address _token, address _to)
    external
    onlyOwner
    returns (bool _sent)
{
    require(_token != address(0), "_token address cannot be 0");
    require(_token != address(this), "Can't withdraw native tokens");
    uint256 _contractBalance = IERC20(_token).balanceOf(address(this));
    _sent = IERC20(_token).transfer(_to, _contractBalance);
    emit TransferForeignToken(_token, _contractBalance);
}
```

Recommendation

Consider adding zero-address check.

Appendix

Appendix 1 - Files in Scope

This audit covered the following file in address 0x3e9fbb8c168dF1f6cfAA372e5bF7cE5F162A7617:

File	SHA-1 hash
KZN.sol	da39a3ee5e6b4b0d3255bfef95601890afd80709