SALUS SECURITY

AUG 2025

CODE SECURITY ASSESSMENT

QKN

Overview

Project Summary

- Name: QKN
- Platform: EVM-compatible chains
- Language: Solidity
- Address:
    - coffeSaleProxy: 0x0a23271fa7a8dfb619e4e0eada11925a796f56fa
    - masterChef: 0xA8c6AC8357f767EFcDee1753309d615a40334B69
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	QKN
Version	v2
Type	Solidity
Dates	Aug 26 2025
Logs	Aug 25 2025; Aug 26 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	1
Total Low-Severity issues	4
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	Wrong totalAllocpoints update logic	Medium	Business Logic	Mitigated
2	Reward may be calculated incorrectly	Low	Business Logic	Acknowledged
3	Price Increase Limited to One Day	Low	Business Logic	Acknowledged
4	Missmatch With comments	Low	Access Control	Acknowledged
5	Missing events for functions that change critical state	Low	Logging	Acknowledged
6	Missing two-step transfer ownership pattern	Informational	Business logic	Acknowledged
7	Lack of Security Contact Information for Responsible Disclosure	Informational	Configuration	Acknowledged

2.2 Notable Findings

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

1. Wrong totalAllocpoints update logic

Severity: Medium	Category: Business Logic
Target: - masterChef.sol	

Description

When a pool is created and receives its first deposit (assuming the deposit amount is x, and x is not 0), totalAllocPoint = 0. At this time, the logic in the first if branch of the `updatePoolAllocPoint` function will be executed, and `totalAllocPoint` will be updated to x. However, in the last if branch, this value is updated again to `totalAllocPoint - oldAllocPoint + pool.allocPoint`, which equals x - 0 + x = 2x. In this case, although there is only one pool, this pool accounts for only 50% instead of the expected 100%.

This will result in the tokens minted to the contract not being fully distributed as rewards, leaving a portion locked in the contract, and also causing users to receive fewer rewards than expected.

Proof of Concept:

```
powerToken.mint(address(this), 1e18);
powerToken.approve(address(chef), 1e18);
uint poolId = chef.createPool();
chef.deposit(1e18);

vm.startPrank(address(0xabcd));
chef.joinPool(1);
powerToken.mint(address(0xabcd), 1e18);
powerToken.approve(address(chef), 1e18);
chef.deposit(1e18);
vm.stopPrank();

skip(100 minutes);

chef.withdraw(1e18);

vm.startPrank(address(0xabcd));
chef.withdraw(1e18);
assertEq(chef.pendingSushi(address(0xabcd)), 0);
assertGt(token.balanceOf(address(chef)), 0);
vm.stopPrank();
```

Recommendation

Consider modifying the update method of `totalAllocPoint` to ensure that it is updated to the correct value each time.

Status

The impact of this issue diminishes as totalAllocPoint increases, with the error ratio eventually becoming negligible. Based on the current on-chain data, the deviation is already minor, and the project team has stated that it is acceptable.

2. Reward may be calculated incorrectly.

Severity: Low	Category: Business Logic

Target:
- masterChef.sol

Description

masterChef.sol:L589 - L596

```
function pendingSushi(address _user) external view returns (uint256) {
    ...
    if (block.timestamp > pool.lastRewardTime && pool.allocPoint > 0 && totalAllocPoint
> 0) {
        uint256 timeElapsed = block.timestamp - pool.lastRewardTime;
        uint256 poolTimeReward = (timeElapsed * currentRewardPerSecond() *
pool.allocPoint) / totalAllocPoint;
       ...
}
```

masterChef.sol:L612 - L613

```
function updatePool() public {
    ...
    uint256 timeElapsed = block.timestamp - pool.lastRewardTime;
    uint256 sushiReward = timeElapsed * currentRewardPerSecond();
    ...
}
```

masterChef.sol:L666 - L667

```
function updatePoolReward(uint256 poolId) public {
    ...
    if (currentTime > pool.lastRewardTime) {
        uint256 timeElapsed = currentTime - pool.lastRewardTime;
        uint256 poolTimeReward = (timeElapsed * currentRewardPerSecond() *
pool.allocPoint) / totalAllocPoint;
        ...
    }
}
```

After calling the `Halving` function to reduce rewards by half, `BlockRewards` is updated to the halved value. If `lastRewardTime` is before the halving point and the current time is after, the entire reward calculation will use the post-halving `BlockRewards`. As a result, rewards accrued between `lastRewardTime` and the halving time are also calculated using the reduced rate, leading to a smaller-than-expected reward. This discrepancy causes an incorrect `accRewardPerShare`, which in turn results in subsequent users' rewards being miscalculated and deviating from the expected values.

Proof of Concept:

```
powerToken.mint(address(this), 1e18);
powerToken.approve(address(chef), 1e18);
uint poolId = chef.createPool();
assertEq(poolId, 1);
chef.deposit(1e18);
assertEq(chef.pendingSushi(address(this)), 0);
```

```
skip(chef.HALVING_INTERVAL());
uint rewards = chef.pendingSushi(address(this));

chef.Halving();
assertLt(chef.pendingSushi(address(this)), rewards);
```

Recommendation

Consider updating the global reward calculation parameters for the period between `lastRewardTime` and the timestamp when the `Halving` function is called, before actually performing the halving. This ensures that subsequent reward calculations remain correct.

Status

This issue has been acknowledged by the team, and promised to notify users

3. Price Increase Limited to One Day

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

Description

coffeeSaleProxy.sol:L239 - L244

```
function updatePrice() public {
    if (block.timestamp >= lastPriceUpdateTime + 1 days) {
        powerPerUsdt = powerPerUsdt * (PRICE_DENOMINATOR + priceIncreasePerDay) /
PRICE_DENOMINATOR;
        lastPriceUpdateTime += 1 days;
    }
}
```

The `updatePrice` function increases the price in the contract based on the number of days elapsed, but it only increases the price for one day in the contract. Even if the `updatePrice` function has not been called for a long time, it will only increase the price for one day, which is inconsistent with the contract design

Recommendation

It is recommended that prices be increased based on the actual time elapsed.

Status

This issue has been acknowledged by the team.

4. Missmatch With comments

Severity: Low	Category: Access Control
Target: - masterChef.sol	

Description

coffeeSaleProxy.sol:L874 - L894

```
// 手動更新指定礦池的分配點數(管理員功能)
function updatePoolAllocPointManually(uint256 poolId) external {
    require(poolId > 0 && poolId < nextPoolId, "Invalid pool ID");
    require(pools[poolId].creator != address(0), "Pool does not exist");
    updatePoolAllocPoint(poolId);
}

function setmaxPower() public {
    maxPower = maxPower * 2;
}


// 批量更新礦池收益(可選, 用於管理員或緊急情況)
function updateMultiplePoolsReward(uint256[] calldata poolIds) external {
    for (uint256 i = 0; i < poolIds.length; i++) {
        uint256 poolId = poolIds[i];
        if (poolId > 0 && poolId < nextPoolId && pools[poolId].creator != address(0)) {
            updatePoolReward(poolId);
        }
    }
}
```

The comments in the contract indicate that all functions are administrator functions, but in fact, these functions have not been given administrator call permissions. Although the `updatePoolAllocPointManually()` and `updateMultiplePoolsReward()` functions do not cause any problems, the `setmaxPower()` function causes `maxPower` to be infinitely amplified.

Recommendation

It is recommended to add permission controls to several functions as noted in the comments.

Status

This issue has been acknowledged by the team.

5. Missing events for functions that change critical state

Severity: Low	Category: Logging
Target: - All	

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.

In the `coffeeSaleProxy` contract, events are lacking in the `updatePrice()`, `setUsdtReceiver()`, `withdraw()`functions.

In the `MasterChef` contract, events are lacking in the `Halving()`, `updatePool()`, `updatePoolAllocPoint()`, `updatePoolReward()`, `createPool()`, `joinPool()` ,`leavePool()`,`takerWithdraw()`,`updatePoolAllocPointManually()`,`setmaxPower()`,`updateMultiplePoolsReward()`functions.

Recommendation

It is recommended to emit events for critical state changes.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

6. Missing two-step transfer ownership pattern

Severity: Informational	Category: Business logic
Target: - All	

Description

All contracts inherit from the `Ownable` contract. This contract does not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

Consider using the Ownable2Step contract from OpenZeppelin instead.

Status

This issue has been acknowledged by the team.

7. Lack of Security Contact Information for Responsible Disclosure

Severity: Informational	Category: Configuration
Target: - All	

Description

All contracts do not specify a security contact point. Including a designated security contact (such as an email address or ENS name) in the contract's NatSpec header facilitates responsible vulnerability disclosure. This makes it easier for external researchers to quickly reach the appropriate team in the event a vulnerability is identified, helping minimize the time window between discovery and mitigation. The Ethereum community has begun standardizing this practice using the `@custom:security-contact` tag, adopted by tools such as OpenZeppelin Wizard and ethereum-lists.

Recommendation

Consider adding a NatSpec comment at the top of the contract with a `@custom:security-contact` field pointing to the preferred disclosure channel.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in address 0x0a23271Fa7a8DFB619e4e0eadA11925a796f56fa and 0xA8c6AC8357f767EFcDee1753309d615a40334B69:

File	SHA-1 hash
coffeeSaleProxy.sol	e85a3a43d4807140aa37385460c8ac1d6a3800e1
masterChef.sol	4d20c465700a2d9b2ae0ec5a7b13433d239d5009