SMART CONTRACT AUDIT REPORT

for

Fishcake CONTRACT

Prepared By: Keane Ho

Singapore
July 22, 2024

Document Properties

Client	Fishcake Contract
Title	Smart Contract Audit Report
Target	Fishcake Contract v1
Version	0.1
Author	Keane Ho
Auditors	Polaristow, tanliwei, bsssss, Dicanoex, Shawn
Reviewed by	F1
Approved by	Seek
Classification	Public

Version Info

Version	Date	Author(s)	Description
0.1	July 22, 2024	Keane Ho	Initial Draft
1.0		Keane Ho	Final Release

Contact

For more information about this document and its contents, please contact Solid-Rock-Security .

Name	Keane Ho
Phone	+86 17208278520
Email	keane@dapplink.xyz

1 Introduction

Given the opportunity to review the **Fishcake Contract v1** design document and related smart contract source code, we outline in the report our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between the smart contract code and the design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of the smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About Fishcake Contract V1

Fishcake is an innovative Web3 platform designed to revolutionize tokenized marketing by leveraging the power of blockchain technology. By enabling everyone to launch and manage incentive-based campaigns directly through smart contract deployed on blockchain. As a fully open-sourced and audited Web3 project deployed on the Polygon blockchain, Fishcake eliminates the need for intermediaries, ensuring transparency, fostering community engagement, and building a sustainable economic ecosystem.

Fishcake empowers businesses to run tokenized campaigns on the blockchain without intermediaries, connecting them directly with customers through our user-friendly web3 App and dApp. All rewarding campaigns published on Fishcake are seamlessly synced with the blockchain and managed through smart contracts. This ensures the credibility and transparency of every tokenized campaign, fostering trust within the community. Claiming and verifying each reward is as simple as scanning a QR code. This user-friendly approach makes participation effortless for customers while ensuring that all transactions are securely recorded on the Polygon blockchain. This integration not only enhances security but also creates a unified loyalty management ecosystem for all users. Fishcake’s blockchain-based system ensures that every interaction, from campaign creation to reward redemption, is transparent and verifiable, providing a robust foundation for building long-lasting relationships between businesses and their customers. What’s more, Fishcake’s innovative

Tokenomics model offers several key advantages: it guarantees a minimum token value through the Redemption Pool, shares platform success with FCC holders via dividends, incentivizes engagement through Mining Pool rewards, empowers users with governance rights, and ensures a sustainable and equitable economic model. These features collectively create a sustainable and rewarding ecosystem, empowering both businesses and customers to flourish in the decentralized Web3 economy.

Table 1.1: Basic Information of Fishcake Contract V1

Item	Description
Issuer	Fishcake Contract
Website	https://www.fishcake.io/zh
Type	Ethereum Smart Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	July 20, 2024

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit.

- https://github.com/FishcakeLab/fishcake-contracts(5265a0)

And this is the commit ID after all fixes for the issues found in the audit have been checked in:

- https://github.com/FishcakeLab/fishcake-contracts(todo)

1.2 About Solid Rock Security

Solid Rock Security Labs focus on Web3 project attack and defense, Web3 project audit and Web3 project security analysis. We are reachable at Telegram(todo),Twitter (https://x.com/0xsolidrock), or Email (todo).

1.3 Disclaimer

Note that this audit does not give any warranties on finding all possible security issues of the given smart contract(s), i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Lastly, this security audit should not be used as investment advice.

2 | Findings

2.1 Summary

Here is a summary of our findings after analyzing the Fishcake Contract v1 Protocol design and implementation. During the first phase of our audit, we studied the smart contract source code and ran our in-house static code analyzer through the codebase. The purpose is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool.

Severity	# of Findings	
Critical	4	■■
High	3	■■
Medium	5	■■
Low	12	■■■■■■
Total	24	

We have so far identified a list of potential issues: some of them involve subtle corner cases that might not be previously thought of, while others refer to unusual interactions among multiple contracts. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities need to be brought up and paid more attention to, which are categorized in the above table. More information can be found in the next subsection, and the detailed discussions of each of them are in Section 3.

2.2 Key Findings

Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 4 critical-severity vulnerabilities, 3 high-severity vulnerabilities , 5 medium-severity vulnerabilities, and 12 low-severity vulnerabilities.

ID	Severity	Title	Status
SRS-001	Critical	Deplete Mining Tokens	
SRS-002	Critical	calculate USDT error	
SRS-003	Critical	USDTAmount calculation error	
SRS-004	Critical	incorrect calculateUSDT formula	
SRS-005	High	activityAdd does not check nft	
SRS-006	High	reward calculation incorrect	
SRS-007	High	Incorrect Reward Calculation Leading to Unclaimable Tokens	
SRS-008	Medium	No check for _activityDeadLine	
SRS-009	Medium	centralization risk	
SRS-010	Medium	Single-step Ownership Transfer Can be Dangerous	
SRS-011	Medium	Use safeTransfer instead of transfer	
SRS-012	Medium	Lack Minting the Rest Amount	
SRS-013	Low	Lack of Validate the _minDropAmt	
SRS-014	Low	Missing zero address check	
SRS-015	Low	Gas saving-1	
SRS-016	Low	Wrong USDT Address	
SRS-017	Low	Unused variables-1	
SRS-018	Low	Event trigger suggestions	
SRS-019	Low	Unused function-1	
SRS-020	Low	Typo-1	
SRS-021	Low	Typo-2	
SRS-022	Low	Unused Event	
SRS-023	Low	Redundant Check	
SRS-024	Low	Variables Never Changed Can be Declared as constant	

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to Section 3 for details.

3 | Detailed Results

3.1 Deplete Mining Tokens

- ID: SRS-001
- Auditors: Polaristow, tanliwei
- Impact: N/A
- Severity: Critical
- Likelihood: N/A

Description

In the MerchantManager contract, a malicious attacker can exploit the activityAdd function to create numerous activities. Subsequently, they can use the drop function to distribute tokens to multiple addresses under their control. This process can be automated using scripts or contracts.

Moreover, there are no minimum activity duration restrictions. This vulnerability allows an attacker to rapidly create activities, distribute large amounts of funds to their controlled addresses, and potentially claim mining rewards through the activityFinish function, especially if they create an NFT.

This attack can significantly deplete the contract's mining rewards, as the attacker's rewards are tied to the total tokens dropped and the number of distributions.Within the MerchantManager, a merchant can initiate an activity by paying a specified amount of tokens via activityAdd. They can then distribute rewards using the drop function and finalize the activity using activityFinish. The latter not only refunds any remaining tokens to the merchant but also mints FccTokenAddr tokens for them. Consequently, a malicious merchant could exploit this cycle to drain FccTokenAddr tokens from the MerchantManager contract.

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/5265a0c8612b8fa953bf0e6232fc602c806dddd6/src/contracts/core/MerchantManger.sol#L135-L354

```
1.     function activityAdd(
2.         string memory _businessName,
3.         string memory _activityContent,
4.         string memory _latitudeLongitude,
5.         uint256 _activityDeadLine,
6.         uint256 _totalDropAmts,
7.         uint8 _dropType,
8.         uint256 _dropNumber,
```

```
9.          uint256 _minDropAmt,
10.         uint256 _maxDropAmt,
11.         address _tokenContractAddr
12.     ) public nonReentrant returns (bool _ret, uint256 _activi
tyId) {
13.         ......
14.     }
15.
16.     function activityFinish(
17.         uint256 _activityId
18.     ) public nonReentrant returns (bool _ret) {
19.       ......
20.     }
21.
22.     function drop(
23.         uint256 _activityId,
24.         address _userAccount,
25.         uint256 _dropAmt
26.     ) external nonReentrant returns (bool _ret) {
27.         ......
28.     }
```

Recommendation

It is recommended to add a minimum activity duration restriction and to allow only one activity per NFT at any given time. Additionally, consider adding restrictions to the activityFinish function, such as making it a privileged function accessible only to authorized users. Another approach could be to implement restrictions on obtaining FccTokenAddr rewards, such as applying a vesting strategy. These measures would help mitigate the risks associated with malicious activities, ensuring better control and security within the MerchantManager contract.

3.2 calculate USDT error 

- ID: SRS-002
- Auditors: Polaristow, Shawn, tanliwei, Dicanoex
- Impact: N/A
- Severity: Critical
- Likelihood: N/A

Description

In the calculateUSDT function of the RedemptionPool contract, the amount of USDT is determined by dividing the USDT balance by the total supply of fishcakeCoin (fishcakeCoin.totalSupply()). However, since users can burn their fishcakeCoin tokens, fishcakeCoin.totalSupply() decreases over time. This means that each time the claim function is called to redeem USDT, the totalSupply value decreases accordingly.

Using fishcakeCoin.totalSupply() for the calculation causes the total supply to continuously diminish as more tokens are burned. Consequently, early users receive a disproportionately larger amount of USDT than originally anticipated, as the total supply used in the calculation decreases faster than expected due to token burning.

This situation escalates to a critical point where fishcakeCoin.totalSupply() becomes less than fishcakeCoin._burnedTokens(), triggering a revert in the calculateUSDT function. This results in the claim function consistently failing and locking the USDT within the contract.

Implementing measures to address this issue, such as adjusting how the USDT calculation considers burned tokens or implementing checks to prevent excessive disparities due to burning, would be crucial to ensure the contract operates as intended without unexpected failures in the claim process.

Code Reference

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/b73bcb231fb6f5e7bc973edc75ab7f6c812a2255/contracts/token/ERC20/ERC20.sol#L198-L201

```
1.     function _burn(address account, uint256 value) internal {
2.         ...
3.         _update(account, address(0), value);
4.     }
5.
6.     function _update(address from, address to, uint256 value)
 internal virtual {
7.         ...
8.         if (to == address(0)) {
9.             unchecked {
10.                 _totalSupply -
= value; //here decrease the _totalSupply
11.             }
12.         }
13.         ...
14.     }
```

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/RedemptionPool.sol#L58-L64

```
1.         function calculateUSDT(uint256 _amount) public view re
turns (uint256) {
2.         // USDT balance / fishcakeCoin total supply
```

```
3.          return
4.              (balance() * _amount * OneUSDT) /
5.              (OneFCC *
6.                  (fishcakeCoin.totalSupply() -
 fishcakeCoin._burnedTokens()));
7.      }
```

Recommendation

Using constant value 1 billion rather than fishcakeCoin.totalSupply() to calculate the $USDT to be redeemed, in the calculateUSDT function.

3.3 USDTAmount calculation error

- ID: SRS-003
- Severity: Critical
- Auditors: Polaristow, Dicanoex, bsssss, Shawn, tanliwei
- Likelihood: N/A
- Impact: N/A

Description

In the Buy function of the DirectSalePool contract, when a user purchases a certain amount of fishcakeCoin, the amount of USDT they need to pay is calculated. However, there's an error in the calculation: 1 Fcc = 0.1 USDT, with USDT having a precision of 6 and fishcakeCoin having a precision of 18. Instead of multiplying, the calculation should be dividing by 10**12. This miscalculation causes the required amount of USDT to be significantly inflated.

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/sale/DirectSalePool.sol#L30-L41

```
1.      function Buy(uint _amount) public nonReentrant {
2.          uint USDTAmount = (_amount * (10 ** 12)) / 10; // 1FC
C = 0.1 USDT
3.          if (_amount > fishcakeCoin.balanceOf(address(this)))
{
4.              revert NotEnoughFishcakeCoin();
5.          }
6.          USDT.safeTransferFrom(msg.sender, address(this), USDT
Amount);
```

```
7.          USDT.transfer(address(redemptionPool), USDTAmount);
8.
9.          fishcakeCoin.transfer(msg.sender, _amount);
10.
11.         emit BuyFishcakeCoinSuccess(msg.sender, USDTAmount, _
amount);
12.     }
```

Poc

Assuming that you need to buy 100 FCC, the amount should be 100*10**18. According to the exchange rate of 1 USDT = 10 FCC, the USDT you need to pay should normally be 100/10=10 USDT, that is, USDTAmount should be 10*10**6. In the actual code, according to USDTAmount = (_amount * (10 ** 12)) / 10, USDTAmount = 100 * 10 ** 18 * (10 ** 12) / 10 = 10*10**30. The correct calculation method should be USDTAmount = _amount / (10 ** 12) / 10, that is, USDTAmount = 100 * 10 ** 18 /(10 ** 12) / 10 = 10*10**6.

Recommendation

It is recommended to use the correct precision for the calculation.

3.4 incorrect calculateUSDT formula

- ID: SRS-004
- Auditors: bsssss
- Impact: N/A
- Severity: Critical
- Likelihood :N/A

Description

The formual _(balance() * _amount * OneUSDT) / (OneFCC * (fishcakeCoin.totalSupply() - fishcakeCoin.burnedTokens())); is incorrect, it should be remove
the OneUSDT and OneFCC .

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/RedemptionPool.sol#L58-L64

```
1.  function calculateUSDT(uint256 _amount) public view returns
(uint256) {
2.          // USDT balance / fishcakeCoin total supply
3.          return
```

```
            (balance() * _amount * OneUSDT) /
            (OneFCC *
                (fishcakeCoin.totalSupply() -
 fishcakeCoin._burnedTokens()));
    }
```

Poc

Consider this scenario:

The balance() is 100, the _amount is 10, the _(fishcakeCoin.totalSupply() - fishcakeCoin.burnedTokens())) is 100.

The result of calculateUSDT should be 100 * 10/ 100 =10. However, the result in formual is 100 * 1e6 * 10 / (1e18 * 100) = 0.

Recommendation

remove the OneUSDT and OneFCC .

3.5 activityAdd does not check nft

- ID: SRS-005
- Auditors: Polaristow
- Impact: N/A
- Severity: High
- Likelihood: N/A

Description

In the activityAdd function, there is no requirement for users to own an NFT. This means that users can first create an activity using the activityAdd function and then create an NFT by calling the createNFT function before calling the activityFinish function. This allows users to extend the benefits of the NFT unfairly.

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/MerchantManger.sol#L135-L291

```
    function activityAdd(
        string memory _businessName,
        string memory _activityContent,
        string memory _latitudeLongitude,
        uint256 _activityDeadLine,
        uint256 _totalDropAmts,
        uint8 _dropType,
```

```
        uint256 _dropNumber,
        uint256 _minDropAmt,
        uint256 _maxDropAmt,
        address _tokenContractAddr
    ) public nonReentrant returns (bool _ret, uint256 _activityId) {
      ......
    }

    function activityFinish(
        uint256 _activityId
    ) public nonReentrant returns (bool _ret) {
      ......
    }
```

Recommendation

It is recommended to validate NFT ownership within the activityAdd function.

3.6 reward calculation incorrect

- ID: SRS-006
- Auditors: Polaristow
- Impact: N/A
- Severity: High
- Likelihood: N/A

Description

In the activityFinish function, according to the documentation and comments, the activity initiator can mine tokens based on either 50% of the total tokens consumed by the activity or 50% of the total number of participants multiplied by 20, whichever is lower. However, the calculation incorrectly uses the total amount of tokens consumed by the activity instead of 50%, leading to incorrect reward distribution and logic errors.

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/MerchantManger.sol#L262-L268

```
uint256 tmpDropedVal = aie.alreadyDropNumber * 20 * 1e18;
```

```
uint256 tmpBusinessMinedAmt = ((
    aie.alreadyDropAmts > tmpDropedVal
        ? tmpDropedVal
        : aie.alreadyDropAmts
) * percent) / 100;
```

Recommendation

It is recommended to divide alreadyDropAmts by 2.

3.7 Incorrect Reward Calculation Leading to Unclaimable Tokens

- ID: SRS-007
- Severity: High
- Auditors: Polaristow
- Likelihood: N/A
- Impact: N/A

Description

In the activityFinish function, when a user claims mining rewards, they can only receive the rewards if minedAmt + tmpBusinessMinedAmt is less than or equal to totalMineAmt. If this condition is not met, the user cannot claim their reward. The correct logic should be to mint the remaining rewards for the user when minedAmt + tmpBusinessMinedAmt is greater than totalMineAmt, which means minting tokens in the amount of totalMineAmt - minedAmt. Since tmpBusinessMinedAmt is calculated, it is unlikely that the amount the user is claiming will exactly equal totalMineAmt. This condition causes some tokens to be unclaimable, resulting in a financial loss for the user.

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/MerchantManger.sol#L269-L276

```
if (totalMineAmt >= minedAmt + tmpBusinessMinedAmt) {
    aie.businessMinedAmt = tmpBusinessMinedAmt;
    minedAmt += tmpBusinessMinedAmt;
    FccTokenAddr.safeTransfer(
        _msgSender(),
        tmpBusinessMinedAmt
```

```
        );
        minedAmount = tmpBusinessMinedAmt;
```

Recommendation

It is recommended to mint the remaining rewards for the user when minedAmt + tmpBusinessMinedAmt is greater than totalMineAmt.

3.8 No check for _activityDeadLine 

- ID: SRS-009
- Auditors: Polaristow
- Impact: N/A
- Severity: Medium
- Likelihood: N/A

Description

The MerchantManager contract does not check _activityDeadLine. For example, in the drop function, it should check _activityDeadLine to determine if the activity has ended. Similarly, the activityFinish function should check _activityDeadLine to ensure the activity has not ended yet.

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/MerchantManger.sol#L30

```
uint256 activityDeadLine; // Activity end time
```

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/MerchantManger.sol#L222-L354

```
function activityFinish(
    uint256 _activityId
) public nonReentrant returns (bool _ret) {
    ......
}

function drop(
    uint256 _activityId,
    address _userAccount,
    uint256 _dropAmt
) external nonReentrant returns (bool _ret) {
```

```
12.     ......
13. }
```

Recommendation

It is recommended to add checks for _activityDeadLine in both the drop and activityFinish functions.

3.9 centralization risk

- ID: SRS-010
- Auditors: Polaristow
- Impact: N/A
- Severity: Medium
- Likelihood: N/A

Description

The functions withdrawUToken and withdraw in the contract pose a centralization risk. Both functions allow the contract owner to withdraw tokens and Ether from the contract, which can lead to potential misuse or abuse of funds.

Since only the owner can call these functions, there is a risk of funds being centralized and potentially being withdrawn without proper oversight.

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/MerchantManger.sol#L356-L380

```
1. function withdrawUToken(
2.     address _tokenAddr,
3.     address _account,
4.     uint256 _value
5. ) public onlyOwner nonReentrant returns (bool _ret) {
6.     require(_tokenAddr != address(0x0), "token address error.
");
7.     require(
8.         IERC20(_tokenAddr).balanceOf(address(this)) >= _value,
9.         "Balance not enough."
10.    );
11.
12.    IERC20(_tokenAddr).safeTransfer(_account, _value);
```

```
13.     _ret = true;
14.     emit WithdrawUToken(_msgSender(), _tokenAddr, _account, _
value);
15. }
16.
17. function withdraw(
18.     address payable _recipient,
19.     uint256 _amount
20. ) public onlyOwner nonReentrant returns (bool _ret) {
21.     require(_recipient != address(0x0), "recipient address er
ror.");
22.     require(_amount <= address(this).balance, "Balance not en
ough.");
23.     (_ret, ) = _recipient.call{value: _amount}("");
24.     emit Wthdraw(_recipient, _amount);
25. }
```

Recommendation

It is recommended to implement additional security measures and decentralization strategies, such as:

1. Multi-signature Wallet: Use a multi-signature wallet for critical operations, requiring multiple parties to approve transactions.

2. Timelock Mechanism: Implement a timelock for withdrawals, allowing users or stakeholders to review and potentially veto transactions.

3. Governance: Introduce a governance mechanism where a broader group of stakeholders can vote on withdrawal proposals.

These measures can help mitigate the centralization risk and ensure a higher level of security and trust.

3.10 Single-step Ownership Transfer Can be Dangerous

- ID: SRS-011
- Auditors: Polaristow
- Impact: N/A
- Severity: Medium
- Likelihood: N/A

Description

Single-step ownership transfer means that if a wrong address was passed when transferring ownership or admin rights it can mean that role is lost forever. If the admin permissions are given to the wrong address within this function, it will cause irreparable damage to the contract.

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/sale/InvestorSalePool.sol#L7

```
1. import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";
```

https://solodit.xyz/issues/lack-of-two-step-role-transfer-spearbit-clober-pdf

Recommendation

It is recommended to use a two-step ownership transfer pattern for superAdminAddress.

3.11 Use safeTransfer instead of transfer

- ID: SRS-012
- Auditors: Polaristow
- Impact: N/A
- Severity: Medium
- Likelihood: N/A

Description

In several places within the contract, ERC20 tokens are transferred using safeTransferFrom and safeTransfer. For example:

```
1. UsdtTokenAddr.safeTransferFrom(_msgSender(), address(this), _value);
2. UsdtTokenAddr.safeTransfer(redemptionPoolAddress, (_value * 75) / 100);
```

However, in some instances, these safe transfer methods are not used. This may pose security risks. For example:

```
1. USDT.transfer(address(redemptionPool), _amount);
```

Recommendation

It is recommended to consistently use safeTransfer.

3.12 Lack Minting the Rest Amount

- ID: SRS-013
- Auditors: tanliwei
- Impact: N/A
- Severity: Medium
- Likelihood: N/A

Description

In the MerchantManger contract, the activityFinish allows the activity initiator to mine tokens based on either 50% of the total token quantity consumed by the activity or 50% of the total number of participants multiplied by 20, whichever is lower.

But, if the minedAmt + tmpBusinessMinedAmt is greater than the totalMineAmt, the mint will be ignored.

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/MerchantManger.sol#L269-L277

```
    function activityFinish(
     uint256 _activityId
) public nonReentrant returns (bool _ret) {
     ......
     if (totalMineAmt >= minedAmt + tmpBusinessMinedAmt) {
                aie.businessMinedAmt = tmpBusinessMinedAmt;
                minedAmt += tmpBusinessMinedAmt;
                FccTokenAddr.safeTransfer(
                    _msgSender(),
                    tmpBusinessMinedAmt
                );
                minedAmount = tmpBusinessMinedAmt;
            }
     ......
}
```

Poc

- totalMineAmt is 300_000_000 * 10 ** 18
- minedAmt is 290_000_000 * 10 ** 18, tmpBusinessMinedAmt is 11_000_000 * 10 ** 18, and the sum of minedAmt and tmpBusinessMinedAmt is 301_000_000 * 10 ** 18 that is greater than totalMineAmt, resulted in the mint being ignored.

Recommendation

Minting the rest amount of token if minedAmt does not reach to the totalMineAmt and totalMineAmt is less than minedAmt + tmpBusinessMinedAmt.

Example:

```
if (totalMineAmt >= minedAmt + tmpBusinessMinedAmt) {
    aie.businessMinedAmt = tmpBusinessMinedAmt;
    minedAmt += tmpBusinessMinedAmt;
    FccTokenAddr.safeTransfer(
        _msgSender(),
        tmpBusinessMinedAmt
    );
    minedAmount = tmpBusinessMinedAmt;
} else {
    tmpBusinessMinedAmt = totalMineAmt - minedAmt;
    if (tmpBusinessMinedAmt != 0) {
        aie.businessMinedAmt = tmpBusinessMinedAmt;
        minedAmt += tmpBusinessMinedAmt;
        FccTokenAddr.safeTransfer(
            _msgSender(),
            tmpBusinessMinedAmt
        );
        minedAmount = tmpBusinessMinedAmt;
    }
    }
```

3.13 Lack of Validate the _minDropAmt

- ID: SRS-014
- Auditors: tanliwei
- Impact: N/A
- Severity: Low
- Likelihood: N/A

Description

The _minDropAmt is required to be zero if the dropType is 1, but there is no validation to force it.

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/MerchantManger.sol#L115-L213

```
/*
        _minDropAmt     When dropType is 1, fill in 0;
*/
function activityAdd(
    ...
    ActivityInfo memory ai = ActivityInfo({
        activityId: activityInfoArrs.length + 1,
        businessAccount: _msgSender(),
        businessName: _businessName,
        activityContent: _activityContent,
        latitudeLongitude: _latitudeLongitude,
        activityCreateTime: block.timestamp,
        activityDeadLine: _activityDeadLine,
        dropType: _dropType,
        dropNumber: _dropNumber,
        minDropAmt: _minDropAmt,//here
        maxDropAmt: _maxDropAmt,
        tokenContractAddr: _tokenContractAddr
    });
```

Recommendation

Adding check on the _minDropAmt to ensure it is zero if the dropType is 1.

3.14 Missing zero address check

- ID: SRS-015
- Auditors: Polaristow
- Impact: N/A
- Severity: Low
- Likelihood: N/A

Description

In the constructor of the contract, there is no check for the zero address for the _fishcakeCoin and _redemptionPool parameters. This can lead to issues if a zero address is inadvertently passed, as shown below.

If a zero address is passed, it could cause the contract to malfunction or fail during execution.

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/sale/DirectSalePool.sol#L25-L28

```
1. constructor(address _fishcakeCoin, address _redemptionPool) {
2.     fishcakeCoin = FishcakeCoin(_fishcakeCoin);
3.     redemptionPool = RedemptionPool(_redemptionPool);
4. }
```

Recommendation

It is recommended to add checks to ensure that neither _fishcakeCoin nor _redemptionPool is a zero address.

3.15 Gas saving-1

- ID: SRS-016
- Auditors: Dicanoex, bsssss, tanliwei
- Impact: N/A
- Severity: Low
- Likelihood: N/A

Description

Transfer to pool directly to save gas

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/sale/DirectSalePool.sol#L35-L36

```
1. USDT.safeTransferFrom(msg.sender, address(this), USDTAmount);
2. USDT.transfer(address(redemptionPool), USDTAmount);
```

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/sale/DirectSalePool.sol#L48-L49

```
1. USDT.safeTransferFrom(msg.sender, address(this), _amount);
2. USDT.transfer(address(redemptionPool), _amount);
```

Recommendation

The two operations above could be merged into one operation to save gas:

```
USDT.safeTransferFrom(msg.sender, address(redemptionPool), _amount);
```

3.16 Wrong USDT Address

- ID: SRS-008
- Auditors: tanliwei, Dicanoex
- Impact: N/A
- Severity: Low
- Likelihood: N/A

Description

The $USDT address used in the RedemptionPool contract is not valid $USDT address on Ethereum or Mantle:
Both 0xc2132d05d31c914a87c6611c10748aeb04b58e8f | Ethereum and 0xc2132D05D31c914a87C6611C10748AEb04B58e8F | Mantle are EOA addresses.

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/RedemptionPool.sol#L22-L23

```
    IERC20 public immutable USDT =
        IERC20(0xc2132D05D31c914a87C6611C10748AEb04B58e8F);
```

Recommendation

The address is a hardcoded address for the Polygon chain, which means the contract cannot be deployed directly to other networks. If it is deployed by mistake, the protocol would not work totally and users won't get their rewards.

Instead of hardcoding, adding a mapping (chainId => USDT address) and query by chainId to get USDT address is recommended.

3.17 Unused variables-1

- ID: SRS-017
- Auditors: Shawn
- Impact: N/A
- Severity: Low
- Likelihood: N/A

Description

The allowDeadLine variable in the NftTokenManager contract is not used

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/NftTokenManager.sol#L19

```
1. mapping(uint256 => uint256) public allowDeadLine;
```

Recommendation

Removing unused variable.

3.18 Event trigger suggestions

- ID: SRS-018
- Auditors: Shawn
- Impact: N/A
- Severity: Low
- Likelihood: N/A

Description

It is recommended to trigger events for changing vault and owner withdrawing USDT in the InvestorSalePool contract to track contract status changes off-chain

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/sale/InvestorSalePool.sol#L143-L149

```
1. function setValut(address _valut) public onlyOwner {
2.     valut = _valut;
```

```
3. }
4. function withdrawUSDT(uint256 _amount) public onlyOwner {
5.     USDT.safeTransfer(valut, _amount);
6. }
```

Recommendation

Add contract events.

3.19 Unused function-1

- ID: SRS-019
- Auditors: bsssss,tanliwei
- Impact: N/A
- Severity: Low
- Likelihood: N/A

Description

The function safeMint() is redundant.

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/NFTManager.sol#L230-L233

```
1. function safeMint(address to) private nonReentrant {
2.     uint256 tokenId = _nextTokenId++;
3.     _safeMint(to, tokenId);
4. }
```

Recommendation

Removing unused function.

3.20 Typo-1

- ID: SRS-020
- Auditors: bsssss
- Impact: N/A
- Severity: Low
- Likelihood: N/A

Description

The Wthdraw should be withdraw.
The Aactivity should be Activity.

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/MerchantManger.sol#L100

```
1. event Wthdraw(address indexed who, uint256 _amount);
```

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/MerchantManger.sol#L296

```
1. _activityId  Aactivity Id
```

Recommendation

Correcting the typo.

3.21 Typo-2

- ID: SRS-021
- Auditors: tanliwei
- Impact: N/A
- Severity: Low
- Likelihood: N/A

Description

In the InvestorSalePool contract, the variable valut intends to be renamed as vault

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/sale/InvestorSalePool.sol#L31

```
address public valut;
```

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/sale/InvestorSalePool.sol#L143-L149

```
function setValut(address _valut) public onlyOwner {
    valut = _valut;
}

function withdrawUSDT(uint256 _amount) public onlyOwner {
    USDT.safeTransfer(valut, _amount);
}
```

Recommendation

Correcting the typo.

3.22 Unused Event

- ID: SRS-022
- Auditors: tanliwei
- Impact: N/A
- Severity: Low
- Likelihood: N/A

Description

The event SetValidTime in the NFTManager contract is unused:

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/NFTManager.sol#L63

```
event SetValidTime(address indexed who, uint256 _time);
```

Recommendation

Removing unused event.

3.23 Redundant Check

- ID: SRS-023
- Auditors: tanliwei
- Impact: N/A
- Severity: Low
- Likelihood: N/A

Description

The activityFinish has a if branch that checks if ai.maxDropAmt * ai.dropNumber is greater than aie.alreadyDropAmts or not, but, it is useless and redundant, due to the calculation of returnAmount comes from ai.maxDropAmt * ai.dropNumber subtracted by aie.alreadyDropAmts, which implies that ai.maxDropAmt * ai.dropNumber is greater than or equal to aie.alreadyDropAmts, otherwise, the solidity whose version greater than 0.8.0 will revert.

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/MerchantManger.sol#L237-L242

```
function activityFinish(
...
    uint256 returnAmount = ai.maxDropAmt *
        ai.dropNumber -
        aie.alreadyDropAmts;
    uint256 minedAmount = 0;
    if (ai.maxDropAmt * ai.dropNumber > aie.alreadyDropAmts)
{//here
        IERC20(ai.tokenContractAddr).safeTransfer(
            _msgSender(),
            returnAmount
        );
    }
```

Recommendation

Removing the redundant check or Updating the condition of the if branch as below example:

```
if (returnAmount != 0) {
    IERC20(ai.tokenContractAddr).safeTransfer(
        _msgSender(),
        returnAmount
    );
}
```

3.24 Variables Never Changed Can be Declared as constant

- ID: SRS-024
- Auditors: tanliwei
- Impact: N/A
- Severity: Low
- Likelihood: N/A

Description

Variables declared but never changed can be declared as constant. The keyword immutable is used for variables are initialized in the constructor and never changed.

Code Reference

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/MerchantManger.sol#L17-L19

```
1. uint256 public immutable totalMineAmt = 300_000_000 * 10 ** 18; // Tota
l mining quantity
2. //30 days = 2592000 s
3. uint256 private immutable maxDeadLine = 2592000;
```

https://github.com/FishcakeLab/fishcake-contracts/blob/main/src/contracts/core/token/FishcakeCoin.sol#L8

```
1. uint256 public immutable MaxTotalSupply = 1_000_000_000 * 10
** 18;
```

Recommendation

Using the keyword constant instead of immutable.