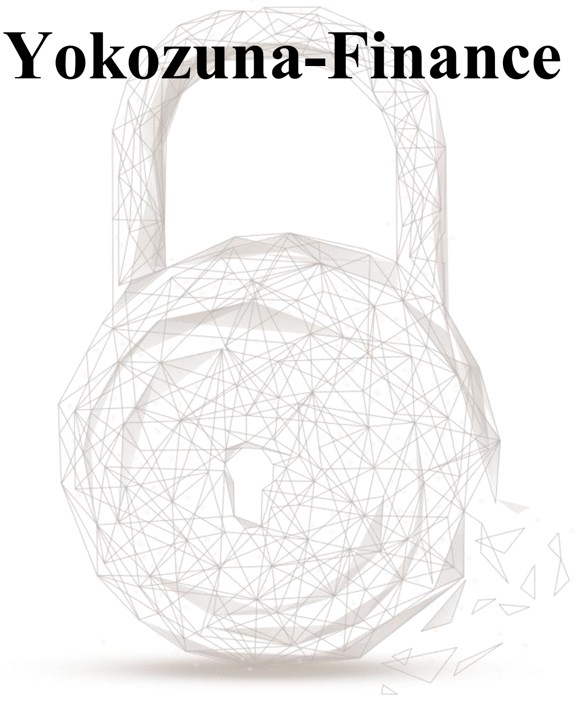
Smart Contract Audit Report for Yokozuna-Finance



Audit Number: 202111301830

Project Name: Yokozuna Finance

Deployment Platform: IOST

Project Contract Link:

https://github.com/Yokozuna-Finance/contracts/blob/main/Stake.js

https://github.com/Yokozuna-Finance/contracts/blob/main/SwapPool.js

Commit id:

Origin: a394f312961aa89d7f16ca6ae2debb9fafe4c365
Final: caa750a9909a0a27765d6811194ea23763819cd9

Audit Start Date: 2021.10.18

Audit Completion Date: 2021.11.30

Audit Result: Pass

Audit Team: Beosin Technology Co. Ltd.

Content

Audit Results Overview

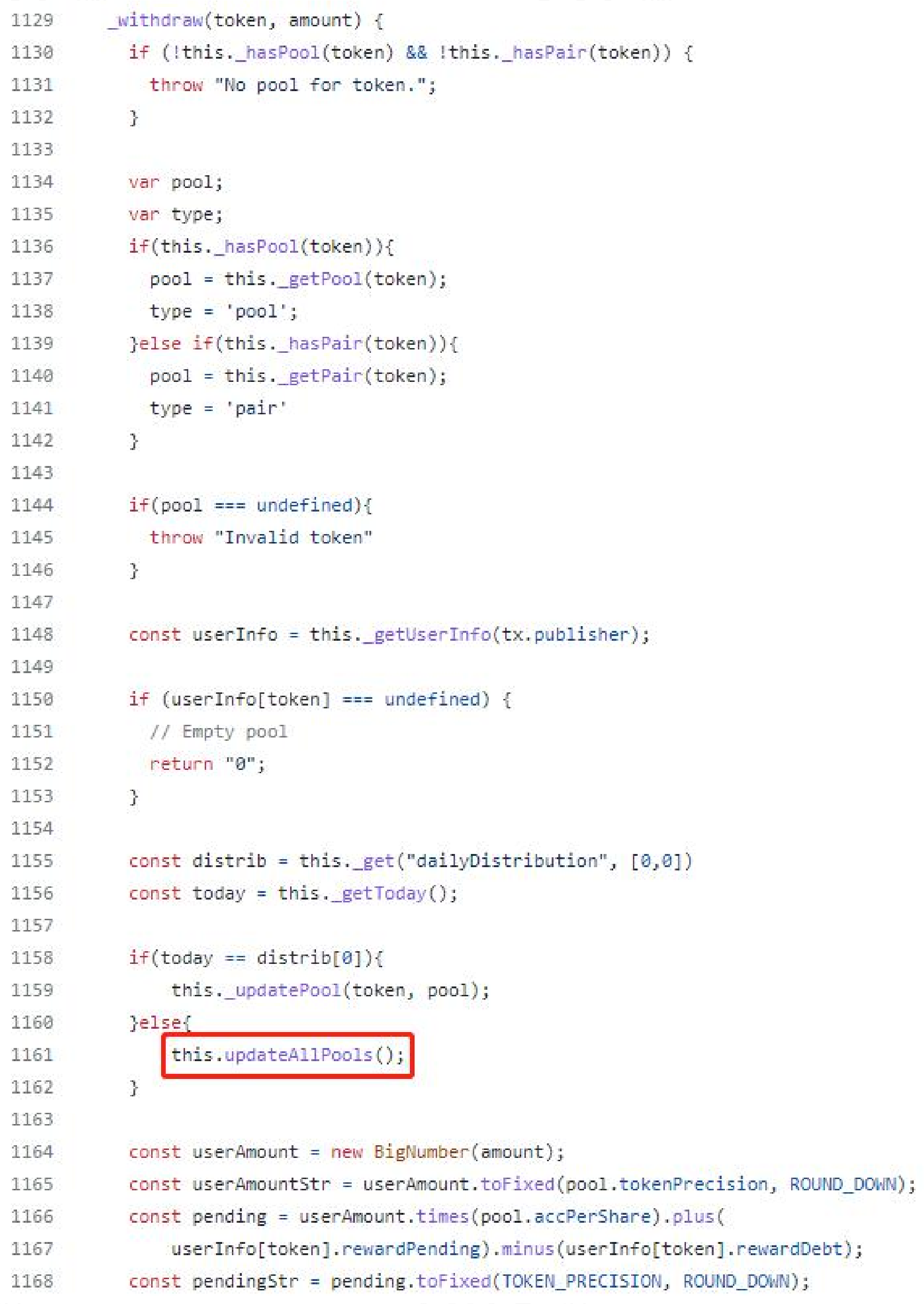
Beosin Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of Yokozuna-Finance project, including Coding Conventions, General Vulnerability and Business Security. **After auditing, the Yokozuna-Finance project was found to have 3 Critical-risk items, 1 High-risk item, 4 Medium-risk items, 1 Low-risk item, 2 Info items. As of the completion of the audit, all risk items have been fixed or properly handled. The overall result of the Yokozuna-Finance project is Pass.** The following is the detailed audit information for this project.

Index	Risk items	Risk level	Fix results status
Stake-1	Variable overwrite vulnerability	Critical	Fixed
Stake-2	Not fully implement the function of the business design	Critical	Fixed
Stake-3	Issue tokens without permission control	Medium	Fixed
Stake-4	The contract call in the *addPooltoVault* function is risky	Medium	Fixed
Stake-5	iost freeze risk	Medium	Fixed
Stake-6	Users manipulate pool/pair allocate point to get more rewards	Medium	Fixed
Stake-7	The *_getMultiplier* function is slightly unfair in the process of calculating rewards	Low	Fixed
Swap-1	Clear any pool/pair information	Critical	Fixed
Swap-2	Use tokens from other accounts to pay listing fees	High	Fixed
Swap-3	Duplicate pairs in the route cause an abnormal number of token exchanges	Info	Acknowledged
Global-1	Calculation accuracy problem	Info	Acknowledged

Table 1. Key Audit Findings

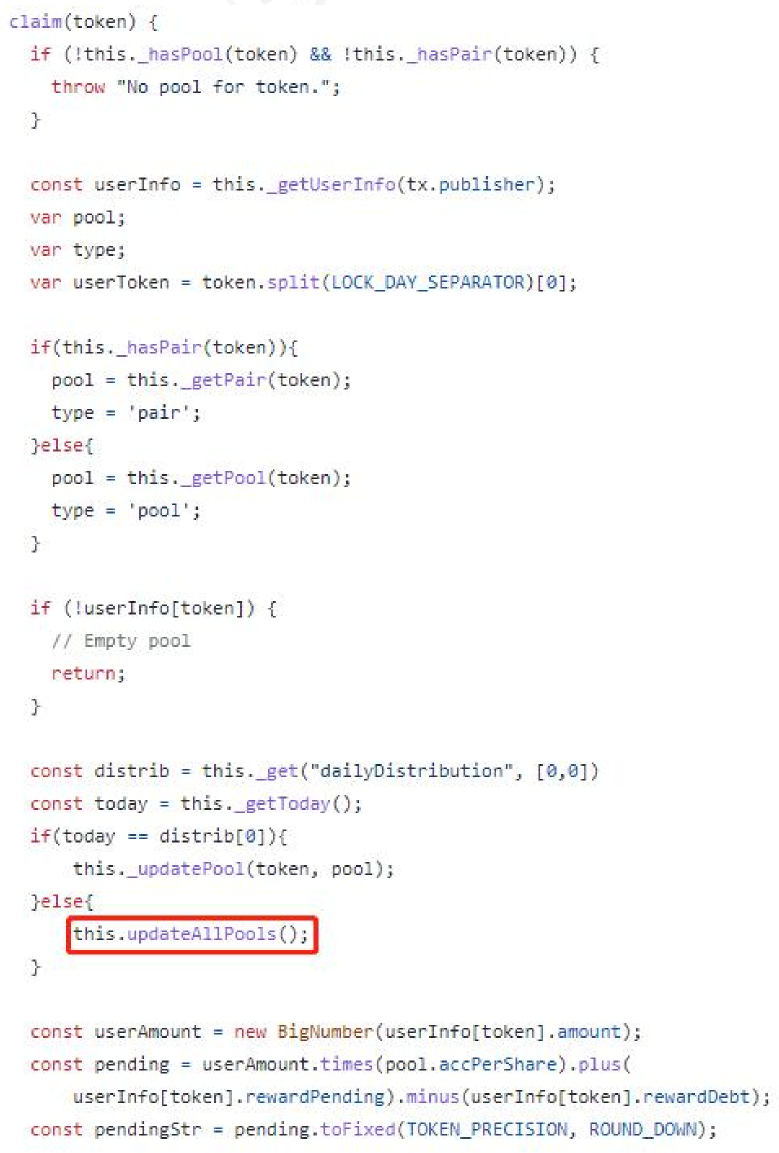
[Stake-1 Critical] Variable overwrite vulnerability

Description: In the *_withdraw, claim,* and *_deposit* functions, if the pool is updated through the *updateAllPools* function, then in the subsequent operations, the old pool parameters are still used, which will cause the reward calculation error.



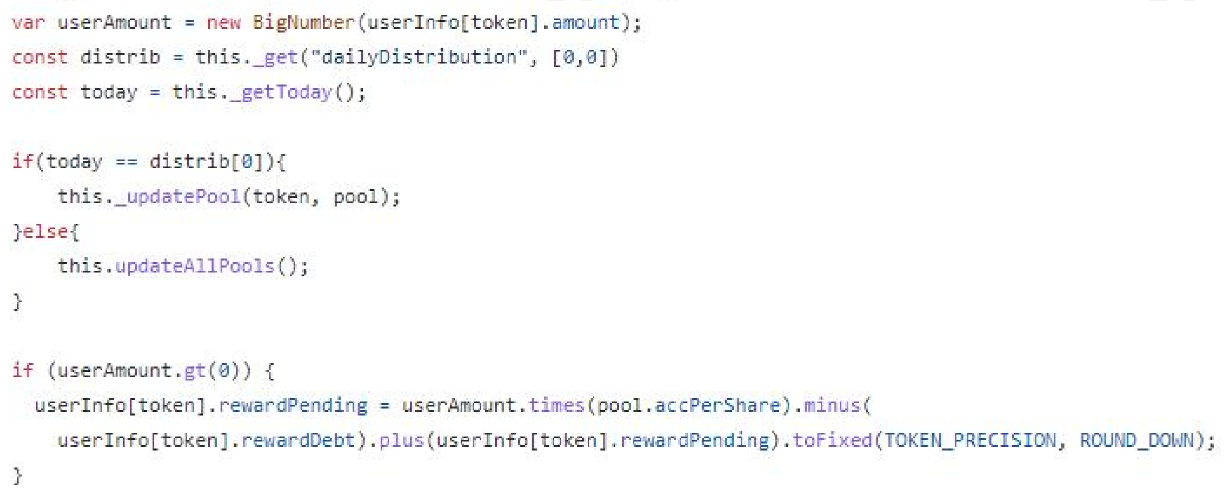
```
_withdraw(token, amount) {
  if (!this._hasPool(token) && !this._hasPair(token)) {
    throw "No pool for token.";
  }

  var pool;
  var type;
  if(this._hasPool(token)){
    pool = this._getPool(token);
    type = 'pool';
  }else if(this._hasPair(token)){
    pool = this._getPair(token);
    type = 'pair'
  }

  if(pool === undefined){
    throw "Invalid token"
  }

  const userInfo = this._getUserInfo(tx.publisher);

  if (userInfo[token] === undefined) {
    // Empty pool
    return "0";
  }

  const distrib = this._get("dailyDistribution", [0,0])
  const today = this._getToday();

  if(today == distrib[0]){
      this._updatePool(token, pool);
  }else{
      this.updateAllPools();
  }

  const userAmount = new BigNumber(amount);
  const userAmountStr = userAmount.toFixed(pool.tokenPrecision, ROUND_DOWN);
  const pending = userAmount.times(pool.accPerShare).plus(
      userInfo[token].rewardPending).minus(userInfo[token].rewardDebt);
  const pendingStr = pending.toFixed(TOKEN_PRECISION, ROUND_DOWN);
```

Figure 1 Source code of *_withdraw* function (Unfixed)



```
claim(token) {
  if (!this._hasPool(token) && !this._hasPair(token)) {
    throw "No pool for token.";
  }

  const userInfo = this._getUserInfo(tx.publisher);
  var pool;
  var type;
  var userToken = token.split(LOCK_DAY_SEPARATOR)[0];

  if(this._hasPair(token)){
    pool = this._getPair(token);
    type = 'pair';
  }else{
    pool = this._getPool(token);
    type = 'pool';
  }

  if (!userInfo[token]) {
    // Empty pool
    return;
  }

  const distrib = this._get("dailyDistribution", [0,0])
  const today = this._getToday();
  if(today == distrib[0]){
      this._updatePool(token, pool);
  }else{
      this.updateAllPools();
  }

  const userAmount = new BigNumber(userInfo[token].amount);
  const pending = userAmount.times(pool.accPerShare).plus(
      userInfo[token].rewardPending).minus(userInfo[token].rewardDebt);
  const pendingStr = pending.toFixed(TOKEN_PRECISION, ROUND_DOWN);
```

Figure 2 Source code of *claim* function (Unfixed)

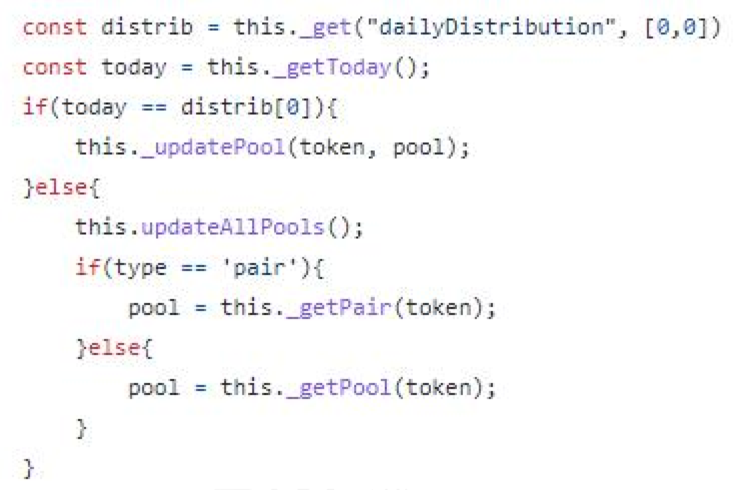


```
var userAmount = new BigNumber(userInfo[token].amount);
const distrib = this._get("dailyDistribution", [0,0])
const today = this._getToday();

if(today == distrib[0]){
    this._updatePool(token, pool);
}else{
    this.updateAllPools();
}

if (userAmount.gt(0)) {
  userInfo[token].rewardPending = userAmount.times(pool.accPerShare).minus(
    userInfo[token].rewardDebt).plus(userInfo[token].rewardPending).toFixed(TOKEN_PRECISION, ROUND_DOWN);
}
```

Figure 3 Part of the source code of the _*deposit* function (Unfixed)

Fix recommendations: It is recommended to obtain the pool information again after *updateAllPools* function.

Fix results: Fixed



```
const distrib = this._get("dailyDistribution", [0,0])
const today = this._getToday();
if(today == distrib[0]){
    this._updatePool(token, pool);
}else{
    this.updateAllPools();
    if(type == 'pair'){
        pool = this._getPair(token);
    }else{
        pool = this._getPool(token);
    }
}
```

Figure 4 Update pool to the value after *updateAllPools*

[Stake-2 Critical] Not fully implement the function of the business design

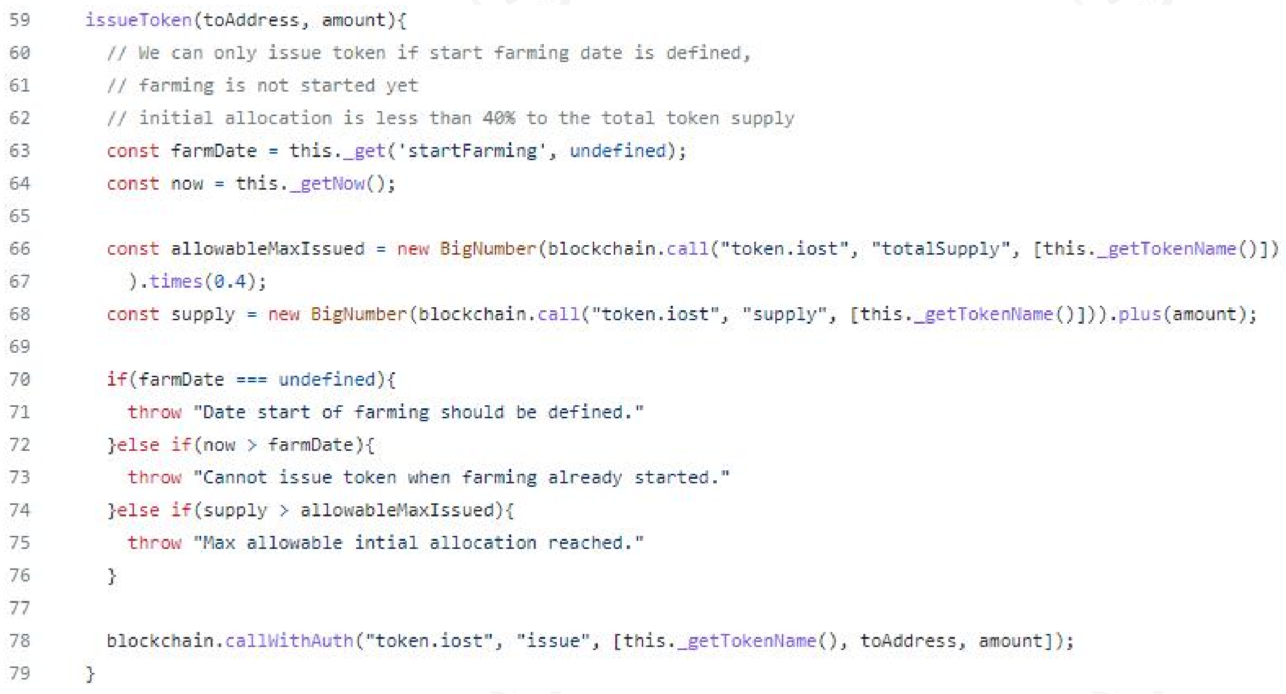
Description: The initial version of the contract had some bugs in the code, which caused the contract to fail to operate exactly as designed. The project team did the corresponding bug fixes to implement all the business functions designed. Due to space limitations, related repair codes are not listed in this report.

Fix recommendations: Modify the code to eliminate bugs..

Fix results: Fixed

[Stake-3 Medium] Issue tokens without permission control

Description: According to the business logic of the Stake contract, before start farming, the contract owner can issue up to 40% of the total amount of reward tokens sent to the specified address. The *issueToken* function in the contract implements this business logic. However, *issueToken* function does not have permission restrictions. Anyone can call this function to mint coins for themselves, as long as the restriction conditions of the function are met.

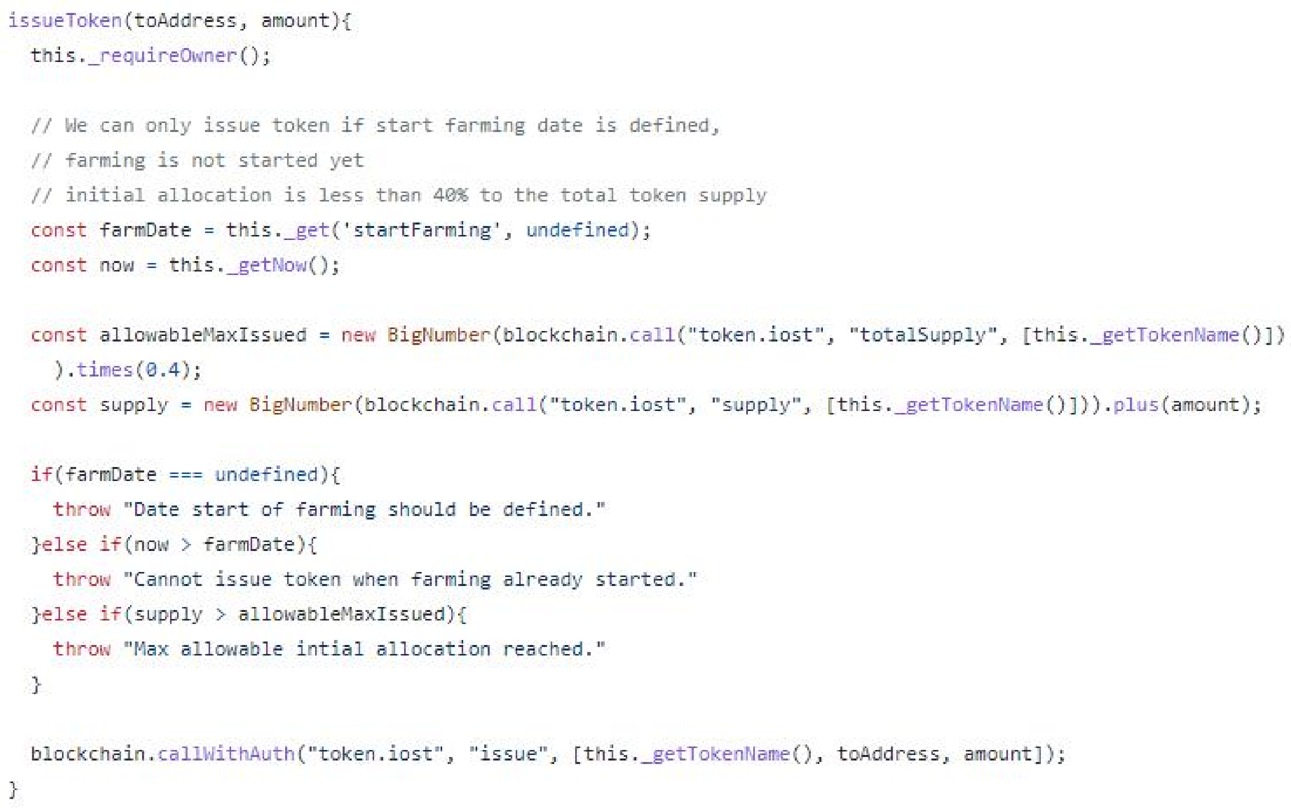


```
issueToken(toAddress, amount){
  // We can only issue token if start farming date is defined,
  // farming is not started yet
  // initial allocation is less than 40% to the total token supply
  const farmDate = this._get('startFarming', undefined);
  const now = this._getNow();

  const allowableMaxIssued = new BigNumber(blockchain.call("token.iost", "totalSupply", [this._getTokenName()])
    ).times(0.4);
  const supply = new BigNumber(blockchain.call("token.iost", "supply", [this._getTokenName()])).plus(amount);

  if(farmDate === undefined){
    throw "Date start of farming should be defined."
  }else if(now > farmDate){
    throw "Cannot issue token when farming already started."
  }else if(supply > allowableMaxIssued){
    throw "Max allowable intial allocation reached."
  }

  blockchain.callWithAuth("token.iost", "issue", [this._getTokenName(), toAddress, amount]);
}
```

Figure 5 Source code of *issueToken* function (Unfixed)

Fix recommendations: It is recommended to add permission verification.

Fix results: Fixed

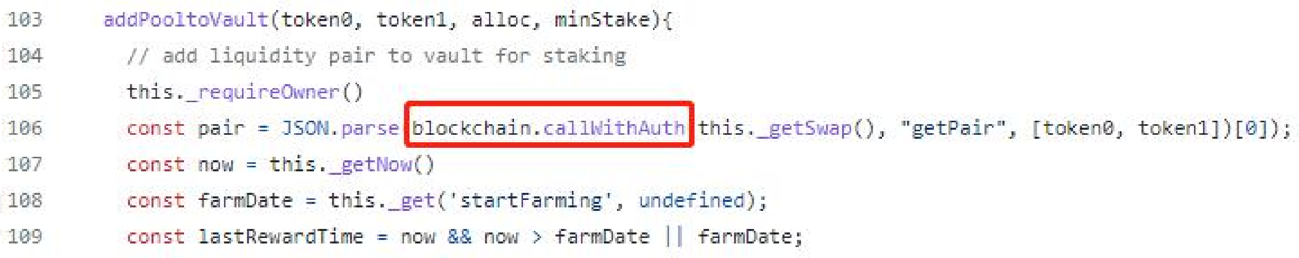


```
issueToken(toAddress, amount){
  this._requireOwner();

  // We can only issue token if start farming date is defined,
  // farming is not started yet
  // initial allocation is less than 40% to the total token supply
  const farmDate = this._get('startFarming', undefined);
  const now = this._getNow();

  const allowableMaxIssued = new BigNumber(blockchain.call("token.iost", "totalSupply", [this._getTokenName()])
    ).times(0.4);
  const supply = new BigNumber(blockchain.call("token.iost", "supply", [this._getTokenName()])).plus(amount);

  if(farmDate === undefined){
    throw "Date start of farming should be defined."
  }else if(now > farmDate){
    throw "Cannot issue token when farming already started."
  }else if(supply > allowableMaxIssued){
    throw "Max allowable intial allocation reached."
  }

  blockchain.callWithAuth("token.iost", "issue", [this._getTokenName(), toAddress, amount]);
}
```

Figure 6 Source code of *issueToken* function (Fixed)

[Stake-4 Medium] The contract call in the *addPooltoVault* function is risky

Description: The *addPooltoVault* function is used to add the pair-LP in the SwapPool contract as a stake token. When adding a pair-LP, the *addPooltoVault* function will call the *getPair* function of the SwapPool contract to query whether the added pair-LP already exists in the SwapPool contract. However, when the *getPair* function is called, the *addPooltoVault* function incorrectly uses the *blockchain.callWithAuth* function, which causes the SwapPool contract to have the authority of the Stake contract when the *getPair* function is executed, which is potentially risky.

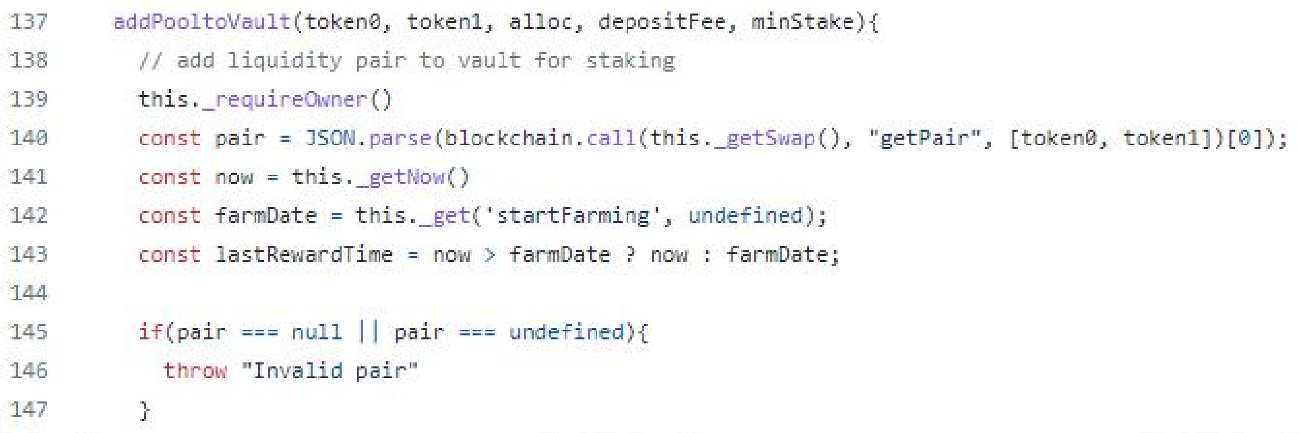


```
addPooltoVault(token0, token1, alloc, minStake){
  // add liquidity pair to vault for staking
  this._requireOwner()
  const pair = JSON.parse(blockchain.callWithAuth(this._getSwap(), "getPair", [token0, token1])[0]);
  const now = this._getNow()
  const farmDate = this._get('startFarming', undefined);
  const lastRewardTime = now && now > farmDate || farmDate;
```

Figure 7 Source code of *addPooltoVault* function (Unfixed)

Fix recommendations: It is recommended to use blockchain.call() to avoid unnecessary authorization.

Fix results: Fixed



```
addPooltoVault(token0, token1, alloc, depositFee, minStake){
  // add liquidity pair to vault for staking
  this._requireOwner()
  const pair = JSON.parse(blockchain.call(this._getSwap(), "getPair", [token0, token1])[0]);
  const now = this._getNow()
  const farmDate = this._get('startFarming', undefined);
  const lastRewardTime = now > farmDate ? now : farmDate;

  if(pair === null || pair === undefined){
    throw "Invalid pair"
  }
```

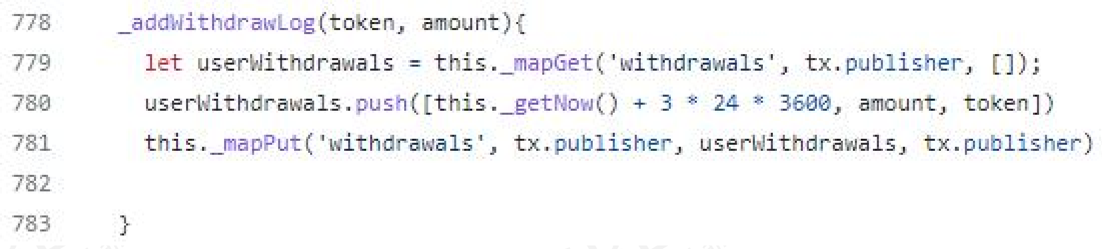
Figure 8 Source code of *addPooltoVault* function (Fixed)

[Stake-5 Medium] iost freeze risk

Description: In the Stake contract, if the stake token is iost, the Stake contract will automatically vote iost to block producer to obtain rewards. When the user executes the *unstake* function, the Stake contract directly sends the stake iost to the user. The Stake contract receives a corresponding amount of frozen transfers from vote.iost, which can be unfrozen after 3 days. If the contract has iost (for example, vote rewarded, or iost unfrozen after 3 days of unvote), users can repeatedly perform stake and unstake operations to freeze all iost available in the contract.

Fix recommendations: It is recommended that when users unstake, freeze the corresponding unstake iost of the user for 3 days.

Fix results: Fixed



```
_addWithdrawLog(token, amount){
  let userWithdrawals = this._mapGet('withdrawals', tx.publisher, []);
  userWithdrawals.push([this._getNow() + 3 * 24 * 3600, amount, token])
  this._mapPut('withdrawals', tx.publisher, userWithdrawals, tx.publisher)

}
```

Figure 9 Freeze iost for 3 days

[Stake-6 Medium] Users manipulate pool/pair allocate point to get more rewards

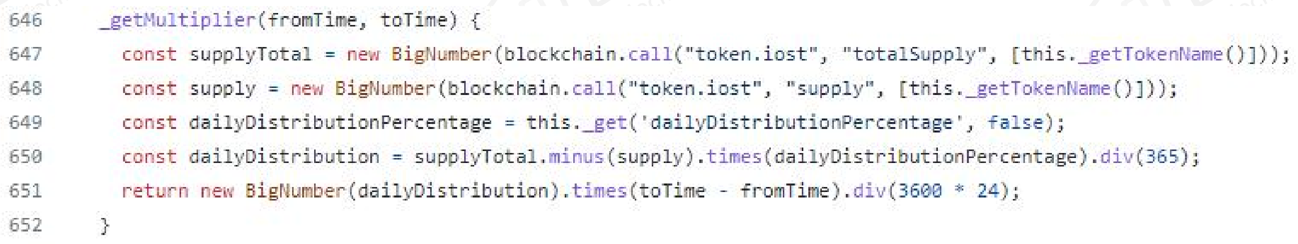
Description: In the Stake contract, users who stake YOKOZUNA_TOKEN can use their stake YOKOZUNA_TOKEN to vote on the specified pool/pair. The corresponding pool/pair will get extra allocate points, and will have a larger proportion when the pool/pair rewards are settled. But there is the following scenario: before updating each pool/pair, users increase their reward income by voting on the corresponding pool/pair. It is recommended to update the rewards of all pools/pairs before updating the voting status.

Fix recommendations: It is recommended to update the rewards of all pools/pairs before updating the voting status。

Fix results: Fixed

[Stake-7 Low] The *_getMultiplier* function is slightly unfair in the process of calculating rewards

Description: The *_getMultiplier* function is used to calculate the number of rewards generated by the pool/pair from the last update to the current time. The calculation method is to multiply the total amount of the remaining issue of the reward token by the reward coefficient, and then allocate it according to the allocate point of the pool/pair. In the original version of the *_getMultiplier* function, as the total amount of remaining issues decreases with each coin minting, the rewards will become less and less. In this case, there is such a scenario: when *updateAllPools* function is called to settle pool/pair rewards in batches, the pool/pair that is sorted lower in the tokenArray will receive less and less reward tokens.

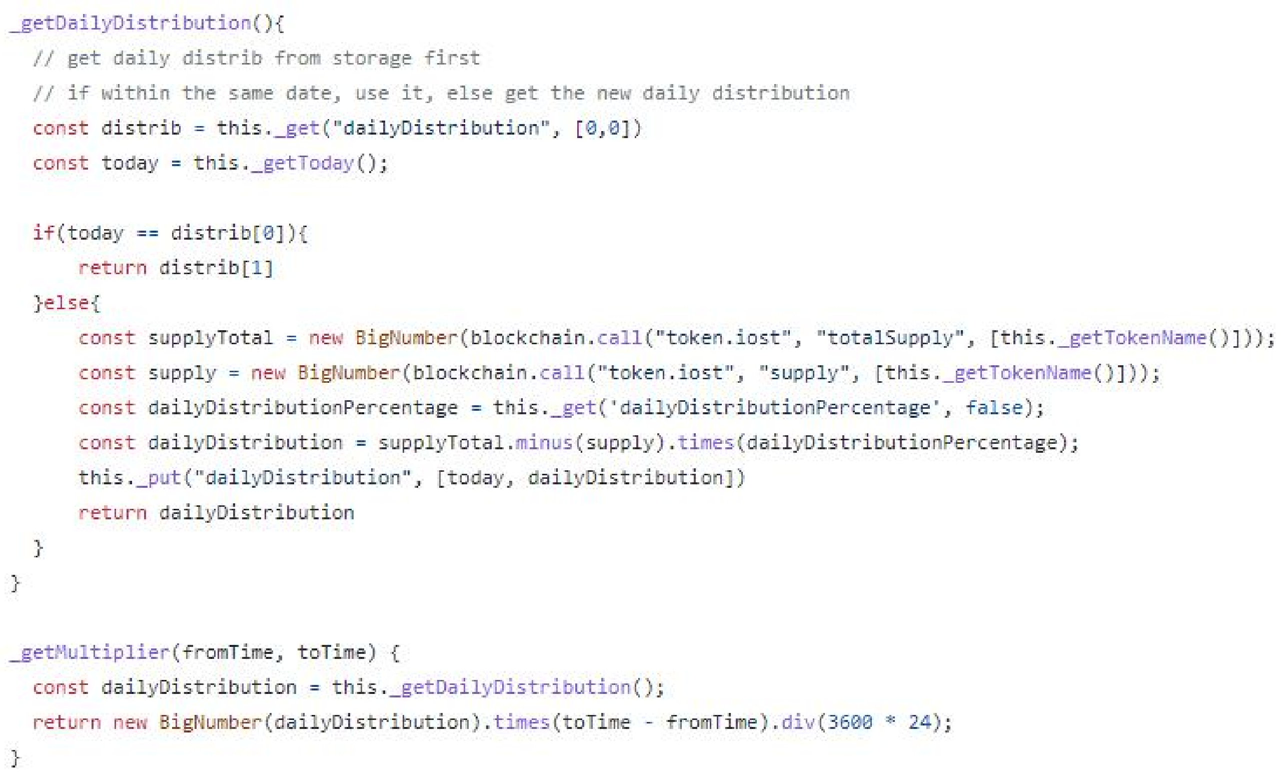


```
_getMultiplier(fromTime, toTime) {
  const supplyTotal = new BigNumber(blockchain.call("token.iost", "totalSupply", [this._getTokenName()]));
  const supply = new BigNumber(blockchain.call("token.iost", "supply", [this._getTokenName()]));
  const dailyDistributionPercentage = this._get('dailyDistributionPercentage', false);
  const dailyDistribution = supplyTotal.minus(supply).times(dailyDistributionPercentage).div(365);
  return new BigNumber(dailyDistribution).times(toTime - fromTime).div(3600 * 24);
}
```

Figure 10 Source code of *_getMultiplier* function (Unfixed)

Fix recommendations: It is recommended to reduce the frequency of changes in the total daily token rewards.

Fix results: Fixed. The total amount of rewards for the day remains unchanged. The total amount of rewards for the day is updated once a day.

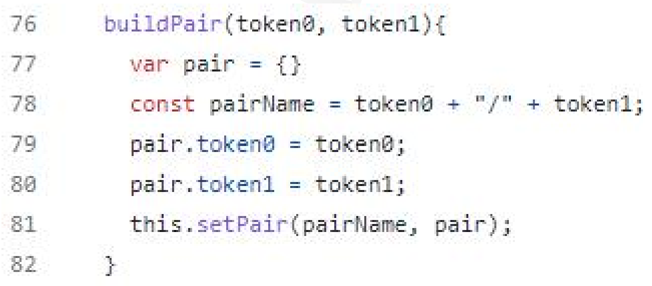


```
_getDailyDistribution(){
  // get daily distrib from storage first
  // if within the same date, use it, else get the new daily distribution
  const distrib = this._get("dailyDistribution", [0,0])
  const today = this._getToday();

  if(today == distrib[0]){
      return distrib[1]
  }else{
      const supplyTotal = new BigNumber(blockchain.call("token.iost", "totalSupply", [this._getTokenName()]));
      const supply = new BigNumber(blockchain.call("token.iost", "supply", [this._getTokenName()]));
      const dailyDistributionPercentage = this._get('dailyDistributionPercentage', false);
      const dailyDistribution = supplyTotal.minus(supply).times(dailyDistributionPercentage);
      this._put("dailyDistribution", [today, dailyDistribution])
      return dailyDistribution
  }
}

_getMultiplier(fromTime, toTime) {
  const dailyDistribution = this._getDailyDistribution();
  return new BigNumber(dailyDistribution).times(toTime - fromTime).div(3600 * 24);
}
```

Figure 11 Source code of *_getMultiplier* function (Fixed)

[Swap-1 Critical] Clear any pool/pair information

Description: The *buildPair* function does not detect whether the pair already exists, and may overwrite any pair information.



```
buildPair(token0, token1){
  var pair = {}
  const pairName = token0 + "/" + token1;
  pair.token0 = token0;
  pair.token1 = token1;
  this.setPair(pairName, pair);
}
```

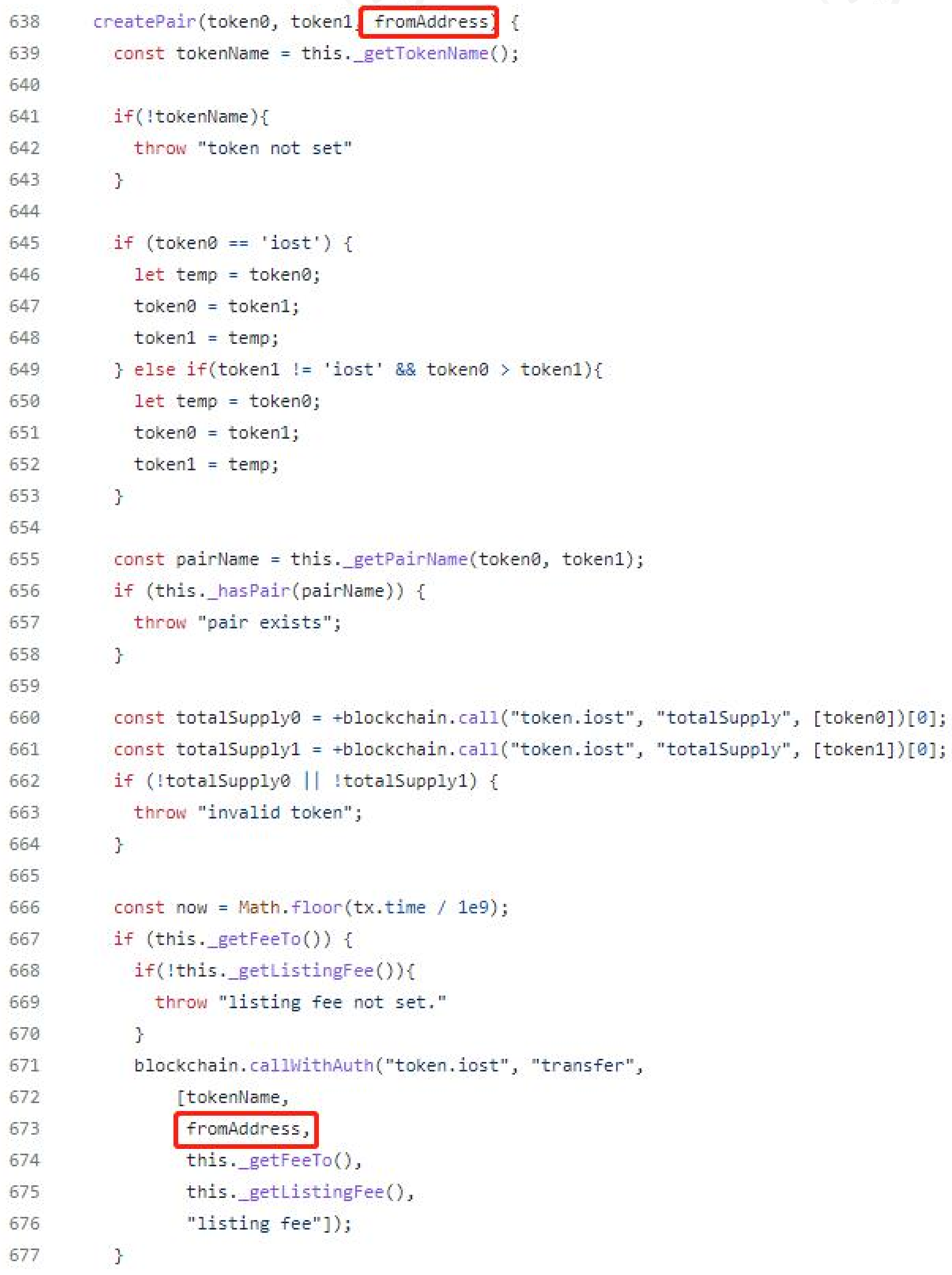
Figure 12 Source code of *buildPair* function (Unfixed)

Fix recommendations: It is recommended to change to an internal function or delete it.

Fix results: Fixed. The *buildPair* function has been removed.

[Swap-2 High] Use tokens from other accounts to pay listing fees

Description: In the *createPair* function, the user can specify another address as fromAddress to pay the listing fee for creating tokens. For example, the user can specify the SwapPool contract address as the fromAddress address. Then the transaction will deduct the listing fee from the SwapPool contract.

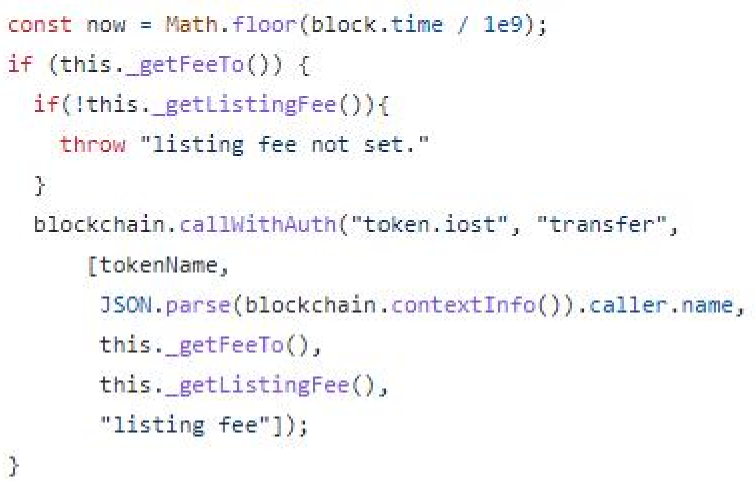


```
createPair(token0, token1, fromAddress) {
  const tokenName = this._getTokenName();

  if(!tokenName){
    throw "token not set"
  }

  if (token0 == 'iost') {
    let temp = token0;
    token0 = token1;
    token1 = temp;
  } else if(token1 != 'iost' && token0 > token1){
    let temp = token0;
    token0 = token1;
    token1 = temp;
  }

  const pairName = this._getPairName(token0, token1);
  if (this._hasPair(pairName)) {
    throw "pair exists";
  }

  const totalSupply0 = +blockchain.call("token.iost", "totalSupply", [token0])[0];
  const totalSupply1 = +blockchain.call("token.iost", "totalSupply", [token1])[0];
  if (!totalSupply0 || !totalSupply1) {
    throw "invalid token";
  }

  const now = Math.floor(tx.time / 1e9);
  if (this._getFeeTo()) {
    if(!this._getListingFee()){
      throw "listing fee not set."
    }
    blockchain.callWithAuth("token.iost", "transfer",
        [tokenName,
         fromAddress,
         this._getFeeTo(),
         this._getListingFee(),
         "listing fee"]);
  }
```

Figure 13 Source code of *createPair* function (Unfixed)

Fix recommendations: It is recommended to deduct the listing fee from the sender.

Fix results: Fixed



```
const now = Math.floor(block.time / 1e9);
if (this._getFeeTo()) {
  if(!this._getListingFee()){
    throw "listing fee not set."
  }
  blockchain.callWithAuth("token.iost", "transfer",
      [tokenName,
       JSON.parse(blockchain.contextInfo()).caller.name,
       this._getFeeTo(),
       this._getListingFee(),
       "listing fee"]);
}
```

Figure 14 Part of the source code of the *createPair* function (Fixed)

[Stake-3 Info] Duplicate pairs in the route cause an abnormal number of token exchanges

Description: In the *swapExactInputToken/swapExactOutputToken* function, if there is a duplicate pair in the route, the exchange rate change caused by the duplicate pair will not be taken into account by *getOutputAmounts/getInputAmounts* function, and the final amount of tokens exchanged is less than the theoretical amount should be obtained.

Fix recommendations: Because the *getOutputAmounts/getInputAmounts* function cannot consider the effect of duplicate pairs on the result in advance, it is recommended that the user call the *getOutputAmounts/getInputAmounts* function to calculate the corresponding amount before swapping, and then call the *swapExactInputToken/swapExactOutputToken* function to swap.

Fix results: Acknowledged

[Global-1 Info] Calculation accuracy problem

Description: Due to the limitations of the JS language, there may be accuracy problems in related calculations, and some efforts have been made in the code. Related measures have mitigated the problem, but there is no guarantee that it will be completely resolved.

Fix recommendations: None

Fix results: Acknowledged

Other Audit Items Descriptions

1. The Stake contract mainly implements the function of staking and obtaining reward tokens. Compared with the common contract of this type, this contract adds the following functions:

   - Stake iost will be voted for the block producer and get the corresponding iost reward. When unstake iost, it will be locked for 3 days;
   - The stake YOKOZUNA_TOKEN will have the right to vote, and the corresponding allocate point can be increased by voting to the designated pool/pair;
   - If the pool/pair specifies the handling fee ratio, the user will directly deduct the corresponding handling fee when staking and send it to the DAO contract;
   - When adding a pool, the contract owner can specify the lock-up days of the pool's tokens, and the corresponding pool can be unstaked only after the lock-in days of the stake.

2. The SwapPool contract implements a token exchange function similar to the uniswap V2 contract.

Appendix 1 Vulnerability Severity Level

Vulnerability Level	Description	Example
Critical	Vulnerabilities that lead to the complete destruction of the project and cannot be recovered. It is strongly recommended to fix.	Malicious tampering of core contract privileges and theft of contract assets.
High	Vulnerabilities that lead to major abnormalities in the operation of the contract due to contract operation errors. It is strongly recommended to fix.	Unstandardized docking of the USDT interface, causing the user's assets to be unable to withdraw.
Medium	Vulnerabilities that cause the contract operation result to be inconsistent with the design but will not harm the core business. It is recommended to fix.	The rewards that users received do not match expectations.
Low	Vulnerabilities that have no impact on the operation of the contract, but there are potential security risks, which may affect other functions. The project party needs to confirm and determine whether the fix is needed according to the business scenario as appropriate.	Inaccurate annual interest rate data queries.
Info	There is no impact on the normal operation of the contract, but improvements are still recommended to comply with widely accepted common project specifications.	It is needed to trigger corresponding events after modifying the core configuration.

Appendix 2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Redundant Code
		SafeMath Features
		Exception Usage
		Gas Consumption
		ABI Specifiers
		Update Usage
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		call/callWithAuth Security
		mapKeys/mapLen/globalMapkeys/globalMapLen Usage
		Variable overwrite
3	Business Security	Business Logics
		Business Implementations

Appendix 3 Disclaimer

This report is made in response to the project code. No description, expression or wording in this report shall be construed as an endorsement, affirmation or confirmation of the project. This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin Technology lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin. Due to the technical limitations of any organization, this report conducted by Beosin still has the possibility that the entire risk cannot be completely detected. Beosin disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin Technology.

Appendix 4 About Beosin

BEOSIN is a leading global blockchain security company dedicated to the construction of blockchain security ecology, with team members coming from professors, post-docs, PhDs from renowned universities and elites from head Internet enterprises who have been engaged in information security industry for many years. BEOSIN has established in-depth cooperation with more than 100 global blockchain head enterprises; and has provided security audit and defense deployment services for more than 1,000 smart contracts, more than 50 blockchain platforms and landing application systems, and nearly 100 digital financial enterprises worldwide. Relying on technical advantages, BEOSIN has applied for nearly 50 software invention patents and copyrights.

BEOSIN

Blockchain Security

Official Website

https://beosin.com

Twitter

https://twitter.com/Beosin_com