Coinsult

Request your audit at coinsult.net

Advanced Manual
Smart Contract Audit

September 6, 2022

Audit requested by

Staking Farm

0x6dd67a04d593f08530afa69f77e0b6df00132bb5

Table of Contents

1. Audit Summary

1.1 Audit scope

1.2 Tokenomics

1.3 Source Code

2. Disclaimer

3. Global Overview

3.1 Informational issues

3.2 Low-risk issues

3.3 Medium-risk issues

3.4 High-risk issues

4. Vulnerabilities Findings

5. Contract Privileges

5.1 Maximum Fee Limit Check

5.2 Contract Pausability Check

5.3 Max Transaction Amount Check

5.4 Exclude From Fees Check

5.5 Ability to Mint Check

5.6 Ability to Blacklist Check

5.7 Owner Privileges Check

6. Notes

6.1 Notes by Coinsult

6.2 Notes by Staking Farm

7. Contract Snapshot

8. Website Review

9. Certificate of Proof

Audit Summary

Audit Scope

Project Name	Staking Farm
Website	https://fistdao.finance
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0x6dd67a04d593f08530afa69f77e0b6df00132bb5
Audit Method	Static Analysis, Manual Review
Date of Audit	6 September 2022

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Tokenomics

Not available

Source Code

Coinsult was comissioned by Staking Farm to perform an audit based on the following code:

https://bscscan.com/address/0x6dd67a04d593f08530afa69f77e0b6df00132bb5#code

Staking Contract

Disclaimer

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Coinsult is not responsible if a project turns out to be a scam, rug-pull or honeypot. We only provide a detailed analysis for your own research.

Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

The information provided in this audit is for informational purposes only and should not be considered investment advice. Coinsult does not endorse, recommend, support or suggest to invest in any project.

Coinsult can not be held responsible for when a project turns out to be a rug-pull, honeypot or scam.

Global Overview

Manual Code Review

In this audit report we will highlight the following issues:

Vulnerability Level	Total	Pending	Acknowledged	Resolved
Informational	0	0	0	0
Low-Risk	7	7	0	0
Medium-Risk	0	0	0	0
High-Risk	0	0	0	0

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Avoid relying on block.timestamp

block.timestamp can be manipulated by miners.

```
function updatePool(uint256 _pid) internal validatePoolByPid(_pid) {
    PoolInfo storage pool = poolInfo[_pid];
    if (block.number &lt;= pool.lastUpdateBlock) {
        return;
    }
    uint256 lpSupply = pool.lpToken.balanceOf(address(this));
    if (lpSupply == 0) {
        pool.lastUpdateBlock = block.number;
        currentAccTokenPerShare[pool.lastUpdateBlock] = pool.accTokenPerShare;
        return;
    }
    uint256 blockInterval = getBlockInterval(pool.lastUpdateBlock, block.number);
    uint256 tokenReward = blockInterval.mul(pool.tokenPerBlock);
    pool.accTokenPerShare = pool.accTokenPerShare.add(tokenReward.mul(1e12).div(lpSupply));
    pool.lastUpdateBlock = block.number;
    currentAccTokenPerShare[pool.lastUpdateBlock] = pool.accTokenPerShare;
}
```

Recommendation

Do not use `block.timestamp`, now or `blockhash` as a source of randomness

Exploit scenario

```
contract Game {

    uint reward_determining_number;

    function guessing() external{
      reward_determining_number = uint256(block.blockhash(10000)) % 10;
    }
}
```

Eve is a miner. Eve calls `guessing` and re-orders the block containing the transaction. As a result, Eve wins the game.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

No zero address validation for some functions

Detect missing zero address validation.

```
function setVoteContract(address _newVoteContract) public onlyOwner {
    voteContract = _newVoteContract;
}
```

Recommendation

Check that the new address is not zero.

Exploit scenario

```
contract C {

  modifier onlyAdmin {
    if (msg.sender != owner) throw;
    _;
  }

  function updateOwner(address newOwner) onlyAdmin external {
    owner = newOwner;
  }
}
```

Bob calls `updateOwner` without specifying the `newOwner`, soBob loses ownership of the contract.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Functions that send Ether to arbitrary destinations

Unprotected call to a function sending Ether to an arbitrary address.

```
function retrieve() public onlyOwner {
    uint256 rewardTokenBalance = IERC20(rewardToken).balanceOf(address(this));
    safeTokenTransfer(msg.sender, rewardTokenBalance);
}
```

Recommendation

Ensure that an arbitrary user cannot withdraw unauthorized funds.

Exploit scenario

```
contract ArbitrarySend{
    address destination;
    function setDestination(){
        destination = msg.sender;
    }

    function withdraw() public{
        destination.transfer(this.balance);
    }
}
```

Bob calls `setDestination` and `withdraw`. As a result he withdraws the contract's balance.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Unchecked transfer

The return value of an external transfer/transferFrom call is not checked.

```
function safeTokenTransfer(address _to, uint256 _amount) internal {
    uint256 rewardTokenBalance = IERC20(rewardToken).balanceOf(address(this));
    if (_amount &gt; rewardTokenBalance) {
        IERC20(rewardToken).transfer(_to, rewardTokenBalance);
    } else {
        IERC20(rewardToken).transfer( _to, _amount);
    }
}
```

Recommendation

Use `SafeERC20`, or ensure that the transfer/transferFrom return value is checked.

Exploit scenario

```
contract Token {
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool succ
}
contract MyBank{
    mapping(address => uint) balances;
    Token token;
    function deposit(uint amount) public{
        token.transferFrom(msg.sender, address(this), amount);
        balances[msg.sender] += amount;
    }
}
```

Several tokens do not revert in case of failure and return false. If one of these tokens is used in `MyBank`, `deposit` will not revert if the transfer fails, and an attacker can call `deposit` for free..

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setRewardTokenContract(address _token) public onlyOwner {
    rewardToken = IERC20(_token);
}

function setRewardTokenAccuracy(uint8 _newAccuracy) public onlyOwner {
    rewardTokenAccuracy = 10 ** _newAccuracy;
}

function setLPTokenAccuracy(uint8 _newAccuracy) public onlyOwner {
    lpTokenAccuracy = 10 ** _newAccuracy;
}

function setVoteContract(address _newVoteContract) public onlyOwner {
    voteContract = _newVoteContract;
}

function setEnableContract(bool _value) public onlyOwner {
    enableContract = _value;
}
```

Recommendation

Emit an event for critical parameter changes.

Exploit scenario

```
contract C {

  modifier onlyAdmin {
    if (msg.sender != owner) throw;
    _;
  }

  function updateOwner(address newOwner) onlyAdmin external {
    owner = newOwner;
  }
}
```

`updateOwner()` has no event, so it is difficult to track off-chain changes in the buy price.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Boolean equality

Detects the comparison to boolean constants.

```
require(enableContract == true,"emergencyWithdraw: The contract has not started");
require(emergencyWithdrawSwitch == true,"emergencyWithdraw: Management does not turn on the eme
```

Recommendation

Remove the equality to the boolean constant.

Exploit scenario

```
contract A {
    function f(bool x) public {
        // ...
        if (x == true) { // bad!
           // ...
        }
        // ...
    }
}
```

Boolean constants can be used directly and do not need to be compare to `true` or `false`.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Costly operations inside a loop

Costly operations inside a loop might waste gas, so optimizations are justified.

```
for (uint256 i = 0; i &lt; user.depositInfo.length; i++) {
    if(user.depositInfo[i].active == true) {
        uint256 pending = user.depositInfo[i].amount.mul((accTokenPerShare.sub(currentAccTokenP
        totalPending = totalPending.add(pending);
    }
}
```

Recommendation

Use a local variable to hold the loop computation result.

Exploit scenario

```
contract CostlyOperationsInLoop{

    function bad() external{
        for (uint i=0; i < loop_count; i++){
            state_variable++;
        }
    }

    function good() external{
      uint local_variable = state_variable;
      for (uint i=0; i < loop_count; i++){
        local_variable++;
      }
      state_variable = local_variable;
    }
}
```

Incrementing `state_variable` in a loop incurs a lot of gas because of expensive `SSTORE`s, which might lead to an `out-of-gas`.

Other Owner Privileges Check

Coinsult lists all important contract methods which the owner can interact with.

⚠️ Owner can set a new ‘voteContract’, this contract can be any contract and Coinsult did not audit this unverified contract.

⚠️ Owner can pause the contract

⚠️ Owner can set a new reward token

Notes

Notes by Staking Farm

No notes provided by the team.

Notes by Coinsult

⚠️ Contract interacts with ‘voteContract’ which is an external contract, not audited by Coinsult and is unverified.

https://bscscan.com/address/0x75b50940e77c778989e15ae582de3ed7ed991683

Contract Snapshot

This is how the constructor of the contract looked at the time of auditing the smart contract.

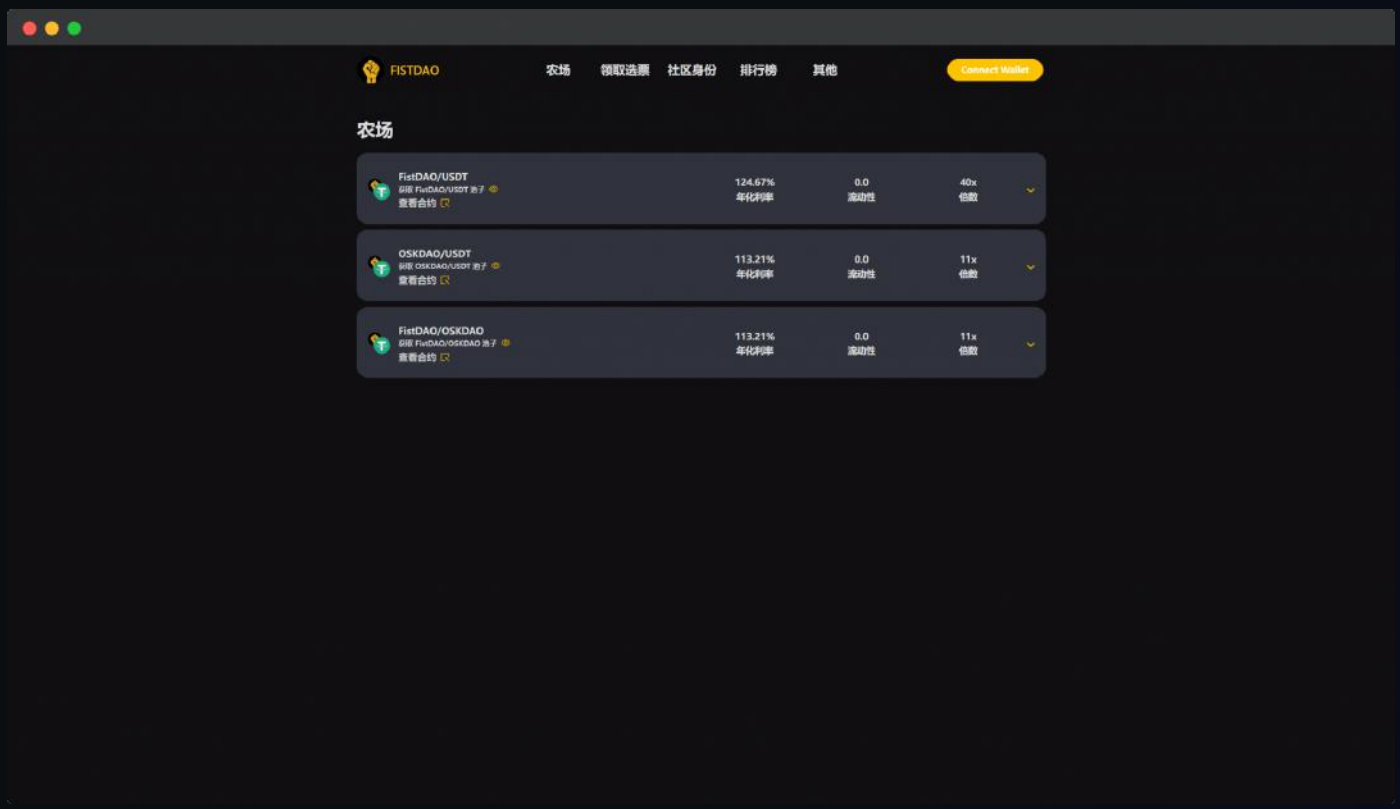
```
contract FistDAOStakeContract is Ownable {
 using SafeMath for uint256;
 using SafeERC20 for IERC20;
 address public voteContract;

 struct UserInfo {
     uint256 depositAmount;
     uint256 mintAmount;
     DepositInfo[] depositInfo;
 }

struct DepositInfo {
     uint256 amount;
     uint256 rewardDebt;
     uint256 depositBlock;
     bool active;
 }
```

Website Review

Coinsult checks the website completely manually and looks for visual, technical and textual errors. We also look at the security, speed and accessibility of the website. In short, a complete check to see if the website meets the current standard of the web development industry.



Type of check	Description
Mobile friendly?	The website is mobile friendly
Contains jQuery errors?	The website does not contain jQuery errors
Is SSL secured?	The website is SSL secured
Contains spelling errors?	The website does not contain spelling errors

Certificate of Proof

Not KYC verified by Coinsult

Staking Farm

Audited by Coinsult.net

Date: 6 September 2022

✔ Advanced Manual Smart Contract Audit

Coinsult

coinsult.net

End of report

Smart Contract Audit

Request your smart contract audit / KYC

t.me/coinsult_tg