

# Advanced Manual **Smart Contract Audit**

October 6, 2022

Audit requested by



0xae7af49e0280b0ffd53dedba687f390eed3d8b6e

# 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 EFG

## 7. Contract Snapshot

## 8. Website Review

## 9. Certificate of Proof

# Audit Summary

## Audit Scope

Project Name	EFG
Website	<a href="https://efgbnb.eth.limo">https://efgbnb.eth.limo</a>
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0xae7af49e0280b0ffd53dedba687f390eed3d8b6e
Audit Method	Static Analysis, Manual Review
Date of Audit	6 October 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 commissioned by EFG to perform an audit based on the following code:

<https://bscscan.com/address/0xae7af49e0280b0ffd53dedba687f390eed3d8b6e#code>

# 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	8	8	0	0
Medium-Risk	2	2	0	0
High-Risk	0	0	0	0

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

## Contract contains Reentrancy vulnerabilities

Additional information: This combination increases risk of malicious intent. While it may be justified by some complex mechanics (e.g. rebase, reflections, buyback).

More information: Slither

```
function _deposit(address _user, uint256 _amount) private {
    UserInfo storage user = userInfo[_user];

    require(user.referrer != address(0), "register first");

    require(_amount >= minDeposit, "less than min");

    require(_amount.mod(minDeposit) == 0 && _amount >= minDeposit, "mod err");

    require(user.maxDeposit == 0 || _amount >= user.maxDeposit, "less before");

    if(user.maxDeposit == 0){
        user.maxDeposit = _amount;
    }else if(user.maxDeposit == maxAddFreeze){
        addFreeze = maxAddFreeze;
    }

    uint256 unfreezeTime = block.timestamp.add(dayPerCycle).add(addFreeze);

    orderInfos[_user].push(OrderInfo(
        _amount,
        block.timestamp,
```

## Recommendation

Apply the check-effects-interactions pattern.

## Exploit scenario

```
function withdrawBalance(){
    // send userBalance[msg.sender] Ether to msg.sender
    // if msg.sender is a contract, it will call its fallback function
    if( ! (msg.sender.call.value(userBalance[msg.sender]))() ){
        throw;
    }
    userBalance[msg.sender] = 0;
}
```

Bob uses the re-entrancy bug to call withdrawBalance two times, and withdraw more than its initial deposit to the contract.

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

## Avoid relying on `block.timestamp`

`block.timestamp` can be manipulated by miners.

```
function distributePoolRewards() public {
    if(block.timestamp > lastDistribute.add(timeStep)){
        uint256 dayNow = getCurDay();
        _distributeLuckPool(dayNow);

        _distributeTopPool(dayNow);
        lastDistribute = block.timestamp;
    }
}
```

## 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.

## Too many digits

Literals with many digits are difficult to read and review.

```
function Reduceproduction()public view returns(uint256) {
    uint256 proportion = 10000;
    uint256 yearTime = 540 * 24*60*60;

    uint256 timeDifference = block.timestamp.sub(startTime);
    // 1
    if(timeDifference > 0 && timeDifference < yearTime && timeDifference > yearTime *2
        proportion = 4096;
    }
    return proportion;
}
```

## Recommendation

Use: Ether suffix, Time suffix, or The scientific notation

## Exploit scenario

```
contract MyContract{
    uint 1_ether = 1000000000000000000;
}
```

While 1\_ether looks like 1 ether, it is 10 ether. As a result, it's likely to be used incorrectly.

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

## Unchecked transfer

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

```
function EmergencyWithdrawal(uint256 _bal) public onlyOwner {  
    usdt.transfer(msg.sender, _bal);  
}
```

## 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 success);  
}  
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 transferOwnership(address newOwner) public onlyOwner {
    require(newOwner != address(0));
    _owner = newOwner;
}

function EmergencyWithdrawal(uint256 _bal) public onlyOwner {
    usdt.transfer(msg.sender, _bal);
}
```

## 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.

```
function _unfreezeFundAndUpdateReward(address _user, uint256 _amount) private {
    UserInfo storage user = userInfo[_user];

    bool isUnfreezeCapital;
    for(uint256 i = 0; i < order.unfreeze && order.isUnfreed == false && _amount > 0; i++) {
        order.isUnfreed = true;
        isUnfreezeCapital = true;
        if(user.totalFreezed > order.amount){
            user.totalFreezed = user.totalFreezed.sub(order.amount);
        }else{
            user.totalFreezed = 0;
        }
        _removeInvalidDeposit(_user, order.amount);
        uint256 staticReward = order.amount.mul(dayRewardPercents).mul(dayPerCycle).div(timeStep);
        if(isFreezeReward){
            if(user.totalFreezed > user.totalRevenue){
                uint256 leftCapital = user.totalFreezed.sub(user.totalRevenue);
                if(staticReward > leftCapital){
```

## 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.

## Conformance to Solidity naming conventions

Allow `_` at the beginning of the `mixed_case` match for private variables and unused parameters.

```
function EmergencyWithdrawal(uint256 _bal) public onlyOwner {  
    function Reduceproduction()public view returns(uint256) {
```

## Recommendation

Follow the Solidity naming convention.

## Rule exceptions

- Allow constant variable name/symbol/decimals to be lowercase (ERC20).
- Allow `_` at the beginning of the `mixed_case` match for private variables and unused parameters.

● **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.

```
function _updateTeamNum(address _user) private {
    UserInfo storage user = userInfo[_user];
    address upline = user.referrer;
    for(uint256 i = 0; i < referDepth; i++){
        if(upline != address(0)){
            userInfo[upline].teamNum = userInfo[upline].teamNum.add(1);
            teamUsers[upline][i].push(_user);
            _updateLevel(upline);
            if(upline == defaultRefer) break;
            upline = userInfo[upline].referrer;
        }else{
            break;
        }
    }
}
```

## 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 SSTOREs, which might lead to an out-of-gas.

● **Medium-Risk:** Should be fixed, could bring problems.

## yearTime might be defined wrong

```
function Reduceproduction()public view returns(uint256) {
    uint256 proportion = 10000;
    uint256 yearTime = 540 * 24*60*60;

    uint256 timeDifference = block.timestamp.sub(startTime);
    // 1
    if(timeDifference > 0 && timeDifference < yearTime && timeDifference > yearTime
        proportion = 4096;
    }
    return proportion;
}
```

## Recommendation

Assuming ‘yearTime’ should be 1 year, it is now 540 days instead of 1 year.

● **Medium-Risk:** Should be fixed, could bring problems.

## Duplicate require statements for minDeposit

```
function _deposit(address _user, uint256 _amount) private {
  UserInfo storage user = userInfo[_user];

  require(user.referrer != address(0), "register first");

  require(_amount >= minDeposit, "less than min");

  require(_amount.mod(minDeposit) == 0 && _amount >= minDeposit, "mod err");

  require(user.maxDeposit == 0 || _amount >= user.maxDeposit, "less before");

  if(user.maxDeposit == 0){
    user.maxDeposit = _amount;
  }else if(user.maxDeposit < _amount){
    user.maxDeposit = _amount;
  }
}
```

## Recommendation

`_amount >= minDeposit` is checked 2 times, remove it once.



## Other Owner Privileges Check

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

⚠ Owner can emergency withdraw funds

# Notes

## Notes by EFG

No notes provided by the team.

## Notes by Coinsult

Contract has been checked for semantical errors, logic / intentions from the contract developer have not been tested.

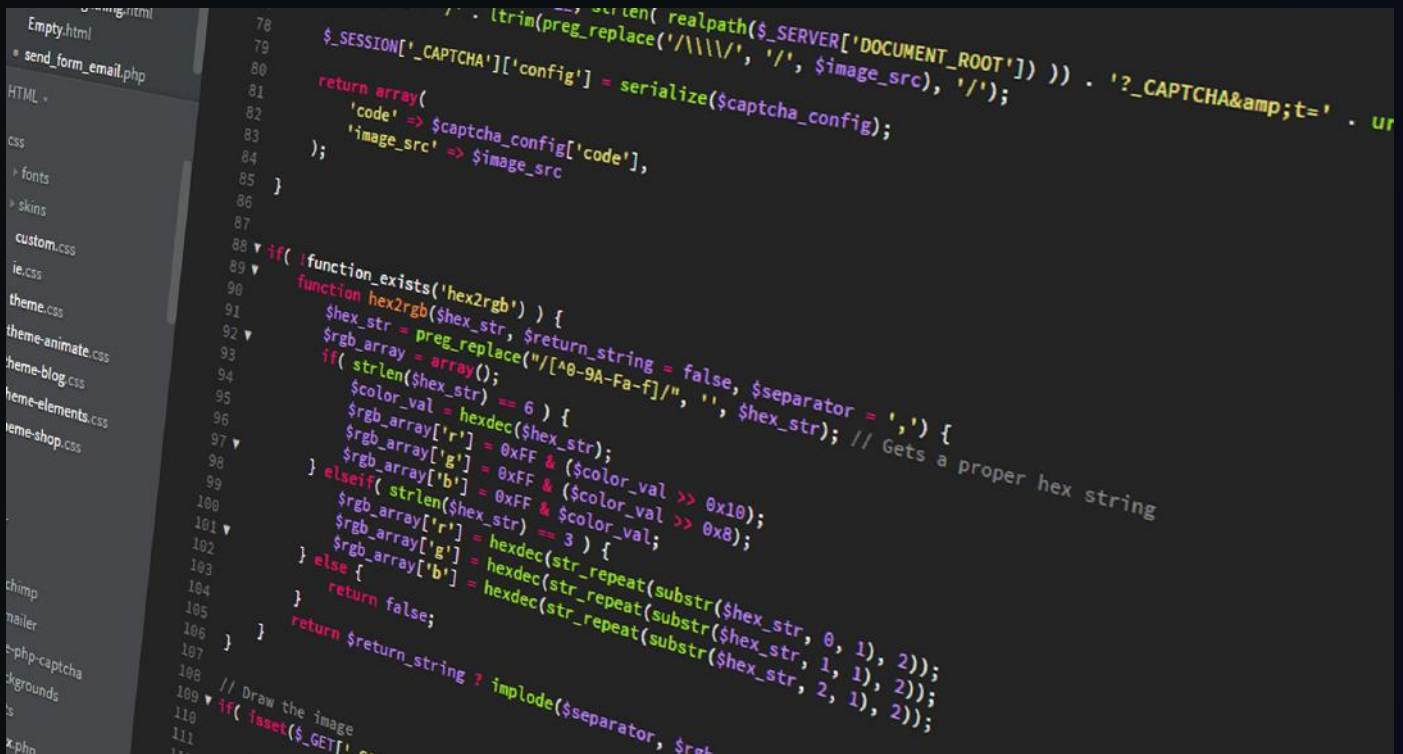
# Contract Snapshot

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





```
contract EFG {  
    using SafeMath for uint256;  
    address _owner;  
  
    IERC20 public usdt;  
    uint256 private constant baseDivider = 10000;  
    uint256 private constant feePercents = 200;  
    uint256 public constant minDeposit = 50e18;  
    uint256 public constant maxDeposit = 2000e18;  
    uint256 private constant freezeIncomePercents = 3000;
```

# 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.



**Not loading fast enough to perform checks.**

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

## EFG

Audited by Coinsult.net



Date: 6 October 2022

✓ Advanced Manual Smart Contract Audit

End of report  
**Smart Contract Audit**

Request your smart contract audit / KYC

**[t.me/coinsult\\_tg](https://t.me/coinsult_tg)**