


# Advanced Manual **Smart Contract Audit**

January 8, 2024

 [CoinsultAudits](#)

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

 [coinsult.net](https://coinsult.net)

Audit requested by



**GbtLiquidity**

0xFd71698f93Ee4D73e3000053E288db9a9FCCCCC

# 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	4	4	0	0
Medium-Risk	0	0	0	0
High-Risk	0	0	0	0

# 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 GbtLiquidity

## 7. Contract Snapshot

## 8. Website Review

## 9. Certificate of Proof

# Audit Summary

Project Name	GbtLiquidity
Website	
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0xFd71698f93Ee4D73e3000053E288db9a9FCCCCC
Audit Method	Static Analysis, Manual Review
Date of Audit	8 January 2024

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.

# Audit Scope

Coinsult was commissioned by GbtLiquidity to perform an audit based on the following code:

<https://testnet.bscscan.com/address/0xFC736F4Edf26Ad2b9C263cA8289afF50385Da799#code>

Note that we only audited the code available to us on this URL at the time of the audit. If the URL is not from any block explorer (main net), it may be subject to change. Always check the contract address on this audit report and compare it to the token you are doing research for.

## Audit Method

Coinsult's manual smart contract audit is an extensive methodical examination and analysis of the smart contract's code that is used to interact with the blockchain. This process is conducted to discover errors, issues and security vulnerabilities in the code in order to suggest improvements and ways to fix them.

### Automated Vulnerability Check

Coinsult uses software that checks for common vulnerability issues within smart contracts. We use automated tools that scan the contract for security vulnerabilities such as integer-overflow, integer-underflow, out-of-gas-situations, unchecked transfers, etc.

### Manual Code Review

Coinsult's manual code review involves a human looking at source code, line by line, to find vulnerabilities. Manual code review helps to clarify the context of coding decisions. Automated tools are faster but they cannot take the developer's intentions and general business logic into consideration.

### Used tools

- Slither: Solidity static analysis framework
- Remix: IDE Developer Tool
- CWE: Common Weakness Enumeration
- SWC: Smart Contract Weakness Classification and Test Cases
- DEX: Testnet Blockchains

# Risk Classification

Coinsult uses certain vulnerability levels, these indicate how bad a certain issue is. The higher the risk, the more strictly it is recommended to correct the error before using the contract.

Vulnerability Level	Description
● Informational	Does not compromise the functionality of the contract in any way
● Low-Risk	Won't cause any problems, but can be adjusted for improvement
● Medium-Risk	Will likely cause problems and it is recommended to adjust
● High-Risk	Will definitely cause problems, this needs to be adjusted

Coinsult has four statuses that are used for each risk level. Below we explain them briefly.

Risk Status	Description
Total	Total amount of issues within this category
Pending	Risks that have yet to be addressed by the team
Acknowledged	The team is aware of the risks but does not resolve them
Resolved	The team has resolved and remedied the risk

# SWC Attack Analysis

The Smart Contract Weakness Classification Registry (SWC Registry) is an implementation of the weakness classification scheme proposed in [EIP-1470](#). It is loosely aligned to the terminologies and structure used in the Common Weakness Enumeration (CWE) while overlaying a wide range of weakness variants that are specific to smart contracts.

ID	Description	Status
SWC-100	Function Default Visibility	Passed
SWC-101	Integer Overflow and Underflow	Passed
SWC-102	Outdated Compiler Version	Passed
SWC-103	Floating Pragma	Passed
SWC-104	Unchecked Call Return Value	Passed
SWC-105	Unprotected Ether Withdrawal	Passed
SWC-106	Unprotected SELFDESTRUCT Instruction	Passed
SWC-107	Reentrancy	Passed
SWC-108	State Variable Default Visibility	Passed
SWC-109	Uninitialized Storage Pointer	Passed
SWC-110	Assert Violation	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed
SWC-112	Delegatecall to Untrusted Callee	Passed
SWC-113	DoS with Failed Call	Passed
SWC-114	Transaction Order Dependence	Passed
SWC-115	Authorization through tx.origin	Passed

SWC-116	Block values as a proxy for time	Passed
SWC-117	Signature Malleability	Passed
SWC-118	Incorrect Constructor Name	Passed
SWC-119	Shadowing State Variables	Passed
SWC-120	Weak Sources of Randomness from Chain Attributes	Passed
SWC-121	Missing Protection against Signature Replay Attacks	Passed
SWC-122	Lack of Proper Signature Verification	Passed
SWC-123	Requirement Violation	Passed
SWC-124	Write to Arbitrary Storage Location	Passed
SWC-125	Incorrect Inheritance Order	Passed
SWC-126	Insufficient Gas Griefing	Passed
SWC-127	Arbitrary Jump with Function Type Variable	Passed
SWC-128	DoS With Block Gas Limit	Passed
SWC-129	Typographical Error	Passed
SWC-130	Right-To-Left-Override control character (U+202E)	Passed
SWC-131	Presence of unused variables	Passed
SWC-132	Unexpected Ether balance	Passed
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Passed
SWC-134	Message call with hardcoded gas amount	Passed
SWC-135	Code With No Effects	Passed
SWC-136	Unencrypted Private Data On-Chain	Passed



Error Code	Description
CS-01	Use uint256 instead of uint

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

### Use uint256 instead of uint

```
function addLiquidity(uint amount) external {
  require(amount >= _minAdd, "Liquidity:The admission amount is too low");
  uint realAmount = amount.div(1 * 10 ** _decimals);
  require(realAmount.mul(1 * 10 ** _decimals) == amount, "Liquidity:Add only integers");
  address account = _msgSender();
  uint invest = Broker(_broker).userInvestTotal(account) +
    Broker(_broker1).userInvestTotal(account);
  uint liquidited = _userInAmountTotal[account];
```

### Recommendation

uint is an alias for uint256 and is not recommended for use. The variable size should be clarified, as this can cause issues when encoding data with selectors if the alias is mistakenly used within the signature string.

Error Code	Description
SLT: 056	Missing Zero Address Validation

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

## No zero address validation for some functions

Detect missing zero address validation.

```
function setAddress(address param, uint status) external onlyOwner {
  if (status == 0) {
    _token = param;
  } else if (status == 1) {
    _broker = param;
  } else if (status == 2) {
    _broker1 = param;
  }
}

function setUint(uint param, uint status) external onlyOwner {
  if (status == 0) {
    _rewardTimes = param;
  } else if (status == 1) {
    _minAdd = param;
  } else if (status == 2) {
    _helpLp = param;
  }
}
```

## 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, so Bob loses ownership of the contract.

Error Code	Description
SLT: 054	Missing Events Arithmetic

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

## Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setAddress(address param, uint status) external onlyOwner {
  if (status == 0) {
    _token = param;
  } else if (status == 1) {
    _broker = param;
  } else if (status == 2) {
    _broker1 = param;
  }
}

function setUint(uint param, uint status) external onlyOwner {
  if (status == 0) {
    _rewardTimes = param;
  } else if (status == 1) {
    _minAdd = param;
  } else if (status == 2) {
    _helpLp = param;
  }
}
```

## 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.

Error Code	Description
CS: 071	Using safemath in Solidity 0.8.0+

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

### Using safemath in Solidity 0.8.0+

SafeMath is generally not needed starting with Solidity 0.8, since the compiler now has built in overflow checking.


```
library SafeMath {
/**
 * @dev Returns the addition of two unsigned integers, with an overflow flag.
 *
 * _Available since v3.4._
 */
function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    unchecked {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }
}

/**
 * @dev Returns the subtraction of two unsigned integers, with an overflow flag.
```

### Recommendation

Check if you really need SafeMath and consider removing it.

Error Code	Description
CSH-01	Reliance on third parties

 **High-Risk:** Must be fixed, will bring problems.

### Reliance on third parties

-

### Recommendation

This contract interacts a lot with 3rd party contracts, we will consider them black boxes that work 100% well.

## Other Owner Privileges Check

Error Code	Description
CEN-100	Centralization: Operator Priviliges

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

✅ No other important owner privileges to mention.

# Notes

## Notes by GbtLiquidity

No notes provided by the team.

## Notes by Coinconsult

No notes provided by Coinconsult

# Contract Snapshot

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

```
contract GbtLiquidity is Context, Ownable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

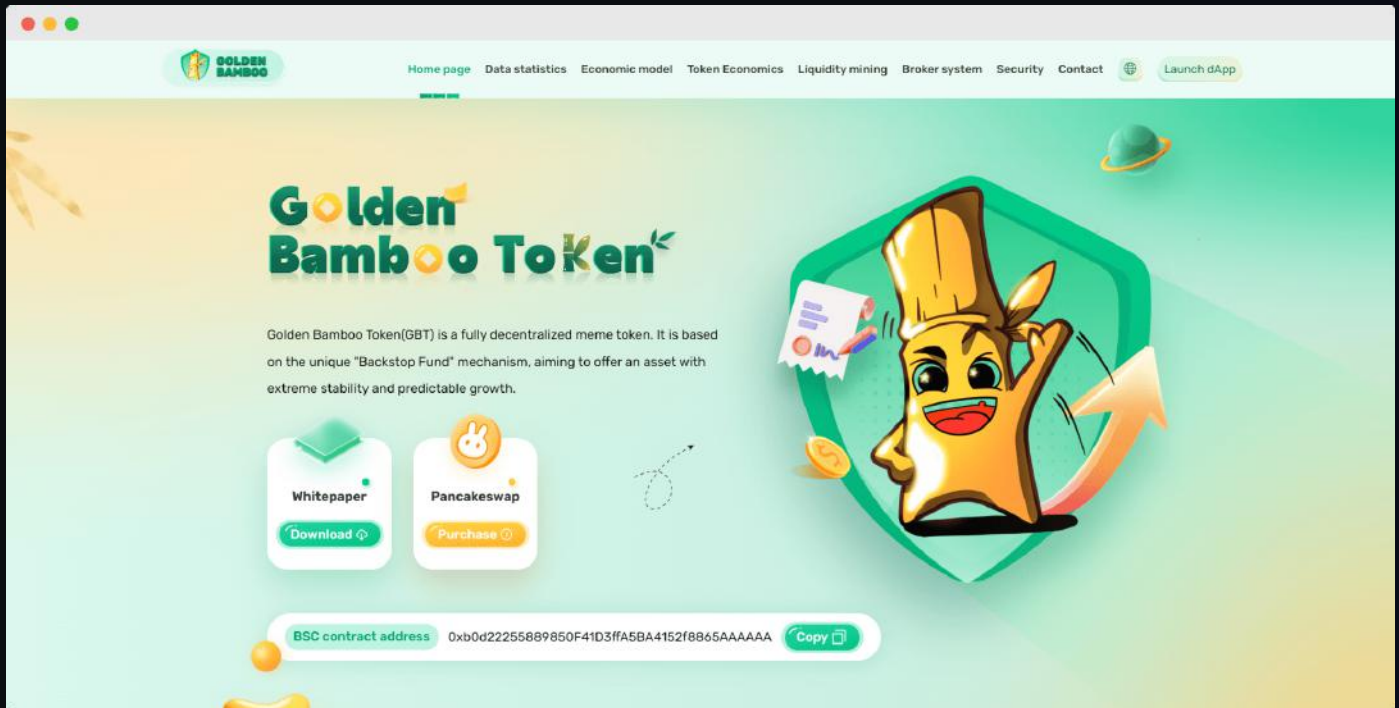
    address public _usdt;
    address public _token;
    address public _broker;
    address public _broker1;
    uint private _decimals;
    // Reward multiplier
    uint public _rewardTimes;
    //Minimum added liquidity amount
    uint public _minAdd;
    // Minimum LP for entry
    uint public _minLp;
    // LP amount to trigger stop of new entries
    uint public _helpLp;
    // Number of online orders
    uint public _currentOrderNum;

    // Virtual U Pool quantity
    uint public virtualUPool;
    // Virtual LP Pool quantity
    uint public virtualLPPool;
```



# 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

## GbtLiquidity

Audited by Coinsult.net



Date: 8 January 2024

✓ Advanced Manual Smart Contract Audit

# 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.

# End of report

## **Smart Contract Audit**

 CoinsultAudits

 info@coinsult.net

 coinsult.net

Request your smart contract audit / KYC

**t.me/coinsult\_tg**