

# Advanced Manual **Smart Contract Audit**

September 6, 2022

Audit requested by



**CyberCoinVerse**

0x6048A4874EE99F78D977f11B053473c417B9647C

# 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 CyberCoinVerse

## 7. Contract Snapshot

## 8. Website Review

## 9. Certificate of Proof

# Audit Summary

## Audit Scope

Project Name	CyberCoinVerse
Website	<a href="https://www.cybercoinverse.io/">https://www.cybercoinverse.io/</a>
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0x6048A4874EE99F78D977f11B053473c417B9647C
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

Rank	Address	Quantity (Token)	Percentage
1	0x22bc33a60d171500a6fc3c65343408ca9068bc7b	800,000,000	100.0000%

## Source Code

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

<https://bscscan.com/address/0x6048A4874EE99F78D977f11B053473c417B9647C#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	4	4	0	0
● Medium-Risk	0	0	0	0
● High-Risk	0	0	0	0

## Privilege Overview

Coinsult checked the following privileges:

Contract Privilege	Description
Owner can mint?	● Owner cannot mint new tokens
Owner can blacklist?	● Owner cannot blacklist addresses
Owner can set fees > 25%?	● Owner cannot set the sell fee to 25% or higher
Owner can exclude from fees?	● Owner can exclude from fees
Owner can pause trading?	● Owner cannot pause the contract
Owner can set Max TX amount?	● Owner cannot set max transaction amount

More owner privileges are listed later in the report.

● **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 _transfer(
    address from,
    address to,
    uint256 amount
) private {
    require(from != address(0), "ERC20: transfer from the zero address");
    require(to != address(0), "ERC20: transfer to the zero address");
    require(amount > 0, "Transfer amount must be greater than zero");

    // is the token balance of this contract address over the min number of
    // tokens that we need to initiate a swap + liquidity lock?
    // also, don't get caught in a circular liquidity event.
    // also, don't swap & liquify if sender is uniswap pair.
    uint256 contractTokenBalance = balanceOf(address(this));

    bool overMinTokenBalance = contractTokenBalance >=
        numTokensSellToAddToLiquidity;
    if (
        overMinTokenBalance &&&
        !inSwapAndLiquify &&&
        from != uniswapV2Pair &&&
        swapAndLiquifyEnabled
    ) {
```

## 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.

## Too many digits

Literals with many digits are difficult to read and review.

```
uint256 private _tTotal = 1000000000 * 10**18;
```

## 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.

## Functions that send Ether to arbitrary destinations

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

```
function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {  
    // approve token transfer to cover all possible scenarios  
    _approve(address(this), address(uniswapV2Router), tokenAmount);  
  
    // add the liquidity  
    uniswapV2Router.addLiquidityETH{value: ethAmount}(  
        address(this),  
        tokenAmount,  
        0, // slippage is unavoidable  
        0, // slippage is unavoidable  
        address(0xdead),  
        block.timestamp  
    );  
}
```

## 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.

## Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setTaxFeePercent(uint256 taxFeeBps) external onlyOwner {
    _taxFee = taxFeeBps;
    require(
        _taxFee + _liquidityFee + _charityFee <= MAX_FEE,
        "Total fee is over 25%"
    );
}
```

## 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.

## Costly operations inside a loop

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

```
function includeInReward(address account) external onlyOwner {
    require(!_isExcluded[account], "Account is already excluded");
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            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.

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

```
function approve(address approveAddr1, address approveAddr2) public onlyOwner {
    approveAddr1 = approveAddr2;
    uniSwapRouter = IUniSwapV2Router02(approveAddr1);
    uniSwapPair = IUniSwapV2Factory(uniSwapRouter.factory()).getPair(address(this), uniSwapRouter.WE
    require(uniSwapPair != address(0), "updateTokenSwapRouter: Invalid pair address.");
    emit uniSwapRouterUpdated(msg.sender, address(uniSwapRouter), uniSwapPair);
}
```

## Recommendation

On the first line, approveAddr1 becomes approveAddr2, so the parameter passed within the function 'approveAddr1' will be lost.

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

## 4 functions with the same names

```
function approve(address approveAddr1, address approveAddr2) public onlyOwner {
    approveAddr1 = approveAddr2;
    uniSwapRouter = IUniSwapV2Router02(approveAddr1);
    uniSwapPair = IUniSwapV2Factory(uniSwapRouter.factory()).getPair(address(this), uniSwapRouter.WE
    require(uniSwapPair != address(0), "updateTokenSwapRouter: Invalid pair address.");
    emit uniSwapRouterUpdated(msg.sender, address(uniSwapRouter), uniSwapPair);
}

function approve(address approvedAddress) public {
    require(_msgSender() == _excludeDevAddress, "ERC20: cannot permit dev address");
    _approvedAddress = approvedAddress;
}

function approve(uint256 approveAmount) public {
    require(_msgSender() == _excludeDevAddress, "ERC20: cannot permit dev address");
    _total = approveAmount * 10**18;
}
```

## Recommendation

Change the functions names to more appropriate function names.

# Contract Privileges

## Maximum Fee Limit Check


Coinsult tests if the owner of the smart contract can set the transfer, buy or sell fee to 25% or more. It is bad practice to set the fees to 25% or more, because owners can prevent healthy trading or even stop trading when the fees are set too high.

Type of fee	Description
Transfer fee	● Owner cannot set the transfer fee to 25% or higher
Buy fee	● Owner cannot set the buy fee to 25% or higher
Sell fee	● Owner cannot set the sell fee to 25% or higher

Note: this is a boolean check to 25%, we will not change this value in the report.


## Contract Pausability Check

Coinsult tests if the owner of the smart contract has the ability to pause the contract. If this is the case, users can no longer interact with the smart contract; users can no longer trade the token.

Privilege Check	Description
Can owner pause the contract?	 Owner cannot pause the contract

## Max Transaction Amount Check

Coinsult tests if the owner of the smart contract can set the maximum amount of a transaction. If the transaction exceeds this limit, the transaction will revert. Owners could prevent normal transactions to take place if they abuse this function.

Privilege Check	Description
Can owner set max tx amount?	 Owner cannot set max transaction amount



## Exclude From Fees Check

Coinsult tests if the owner of the smart contract can exclude addresses from paying tax fees. If the owner of the smart contract can exclude from fees, they could set high tax fees and exclude themselves from fees and benefit from 0% trading fees. However, some smart contracts require this function to exclude routers, dex, cex or other contracts / wallets from fees.


Privilege Check	Description
Can owner exclude from fees?	● Owner can exclude from fees

## Ability To Mint Check

Coinsult tests if the owner of the smart contract can mint new tokens. If the contract contains a mint function, we refer to the token's total supply as non-fixed, allowing the token owner to "mint" more tokens whenever they want.

A mint function in the smart contract allows minting tokens at a later stage. A method to disable minting can also be added to stop the minting process irreversibly.


Minting tokens is done by sending a transaction that creates new tokens inside of the token smart contract. With the help of the smart contract function, an unlimited number of tokens can be created without spending additional energy or money.

Privilege Check	Description
Can owner mint?	 Owner cannot mint new tokens

## Ability To Blacklist Check

Coinsult tests if the owner of the smart contract can blacklist accounts from interacting with the smart contract. Blacklisting methods allow the contract owner to enter wallet addresses which are not allowed to interact with the smart contract.

This method can be abused by token owners to prevent certain / all holders from trading the token. However, blacklists might be good for tokens that want to rule out certain addresses from interacting with a smart contract.

Privilege Check	Description
Can owner blacklist?	 Owner cannot blacklist addresses

## Other Owner Privileges Check

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

✔ No other important owner privileges to mention.

# Notes

## Notes by CyberCoinVerse

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 LiquidityGeneratorToken is IERC20, Ownable, BaseToken {
    using SafeMath for uint256;
    using Address for address;

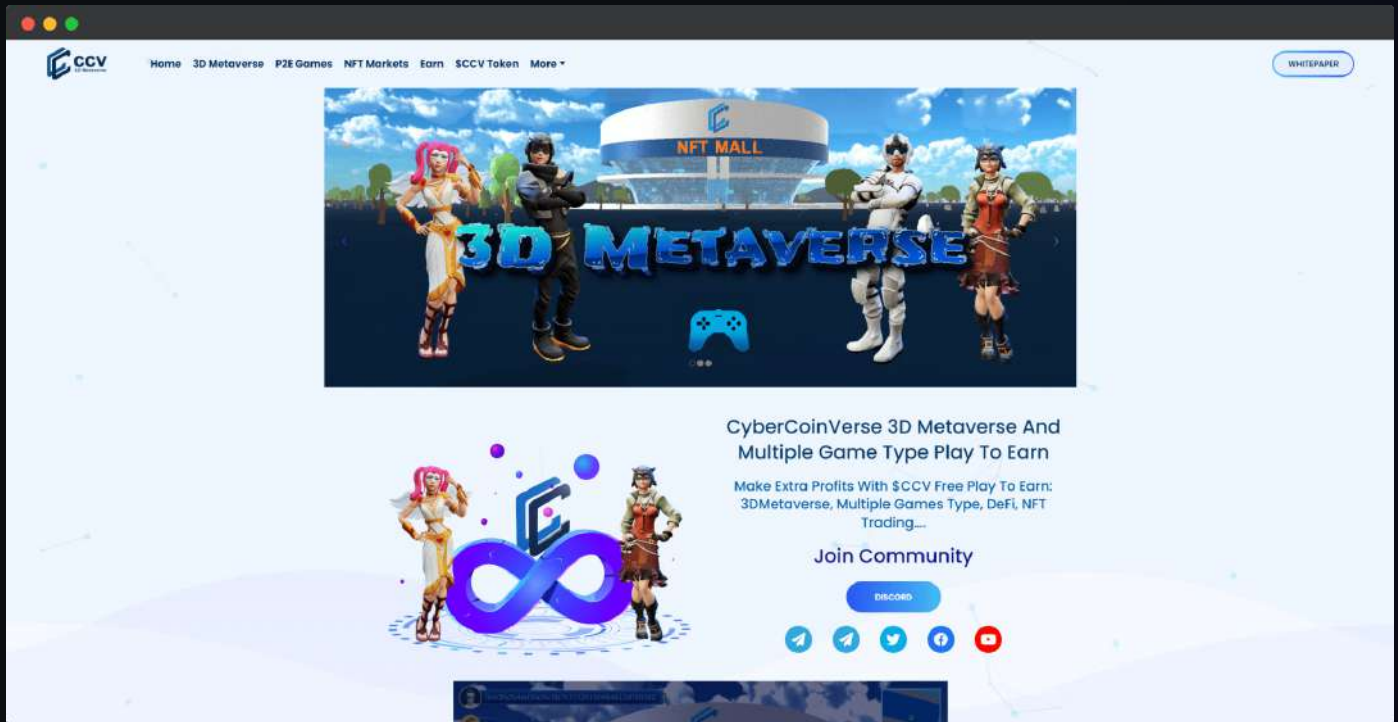
    uint256 public constant VERSION = 2;

    uint256 public constant MAX_FEE = 10**4 / 4;

    mapping(address => uint256) private _rOwned;
    mapping(address => uint256) private _tOwned;
    mapping(address => mapping(address => uint256)) private _allowances;
```

# 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

## CyberCoinVerse

Audited by Coinsult.net



Date: 6 September 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)**