

Advanced Manual **Smart Contract Audit**

September 4, 2022

Audit requested by



Web3gold

0xF70AA7d6eC1813F38b87d661Fb7E22888EE2979E

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 Web3gold

7. Contract Snapshot

8. Website Review

9. Certificate of Proof

Audit Summary

Audit Scope

Project Name	Web3gold
Website	https://web3gold.finance/
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0xF70AA7d6eC1813F38b87d661Fb7E22888EE2979E
Audit Method	Static Analysis, Manual Review
Date of Audit	4 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	0xf359404297bc7b98e855b86deea5cb270a273166	1,000,000,000	100.0000%

Source Code

Coinsult was commissioned by Web3gold to perform an audit based on the following code:

<https://bscscan.com/address/0xF70AA7d6eC1813F38b87d661Fb7E22888EE2979E#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	6	6	0	0
● Medium-Risk	2	2	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 can 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 can 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 _transferFrom(address sender, address recipient, uint256 amount) internal returns (bool) {
    if(inSwap){ return _basicTransfer(sender, recipient, amount); }

    if(!authorizations[sender] &&& !authorizations[recipient]){
        require(tradingOpen,"Trading not open yet");
    }

    if (!authorizations[sender] &&& recipient != address(this) &&& recipient != address(pair) &&& recipient != address(msg.sender)) {
        uint256 heldTokens = balanceOf(recipient);
        require((heldTokens + amount) <= _maxWalletToken,"Total Holding is currently limited");
    }

    if (sender == pair &&&
        buyCooldownEnabled &&&
        !isTimelockExempt[recipient]) {
        require(cooldownTimer[recipient] < block.timestamp,"Please wait for 1min between two swaps");
        cooldownTimer[recipient] = block.timestamp + cooldownTimerInterval;
    }

    if(recipient == pair) { checkTxLimit(sender, amount); }

    if(shouldSwapBack()){ swapBack(); }
```


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 public swapThreshold = _totalSupply * 5 / 1000000;
```

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.

No zero address validation for some functions

Detect missing zero address validation.

```
function transferOwnership(address payable adr) public onlyOwner {
    owner = adr;
    authorizations[adr] = true;
    emit OwnershipTransferred(adr);
}
```

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.

● **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 swapBack() internal swapping {
    uint256 dynamicLiquidityFee = isOverLiquified(targetLiquidity, targetLiquidityDenominator) ? 0 : li
    uint256 amountToLiquify = swapThreshold.mul(dynamicLiquidityFee).div(totalFee).div(2);
    uint256 amountToSwap = swapThreshold.sub(amountToLiquify);

    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = WBNB;

    uint256 balanceBefore = address(this).balance;

    router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        amountToSwap,
        0,
        path,
        address(this),
        block.timestamp
    );

    uint256 amountBNB = address(this).balance.sub(balanceBefore);

    uint256 totalBNBFee = totalFee.sub(dynamicLiquidityFee.div(2));
```

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.

Write after write

Variables that are written but never read and written again.

```
function swapBack() internal swapping {
    uint256 dynamicLiquidityFee = isOverLiquified(targetLiquidity, targetLiquidityDenominator) ? 0 : li
    uint256 amountToLiquify = swapThreshold.mul(dynamicLiquidityFee).div(totalFee).div(2);
    uint256 amountToSwap = swapThreshold.sub(amountToLiquify);

    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = WBNB;

    uint256 balanceBefore = address(this).balance;

    router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        amountToSwap,
        0,
        path,
        address(this),
        block.timestamp
    );

    uint256 amountBNB = address(this).balance.sub(balanceBefore);

    uint256 totalBNBFee = totalFee.sub(dynamicLiquidityFee.div(2));
```

Recommendation

Fix or remove the writes.

Exploit scenario

```
```solidity
contract Buggy{
 function my_func() external initializer{
 // ...
 a = b;
 a = c;
 // ..
 }
}
```

`a` is first assigned to `b`, and then to `c`. As a result the first write does nothing.

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

## Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setMaxTxPercent_base1000(uint256 maxTXPercentage_base1000) external onlyOwner() {
 _maxTxAmount = (_totalSupply * maxTXPercentage_base1000) / 1000;
}
```

## 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.

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

`setfee()` states that fees cannot be set more than 24% but sell fee can be set more than 24%.

```
function setFees(uint256 _liquidityFee, uint256 _developmentFee, uint256 _treasuryFee, uint256 _burnFee) {
 liquidityFee = _liquidityFee;
 developmentFee = _developmentFee;
 treasuryFee = _treasuryFee;
 burnFee = _burnFee;
 totalFee = _liquidityFee.add(_developmentFee).add(_treasuryFee).add(_burnFee);
 feeDenominator = _feeDenominator;
 require(totalFee <= 0.24 * feeDenominator, "Fees cannot exceed 24%");
 emit Transfer(sender, burnFeeReceiver, burnTokens);
}
```

## Recommendation

rewrite functions so total buy and total sell fees cannot be set above 24%

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

TransferFrom can return wrong require statement message (returns “wait 1 minute” but cooldown time can be set up to 10 minutes)

```
if (sender == pair &&
 buyCooldownEnabled &&
 !isTimelockExempt[recipient]) {
 require(cooldownTimer[recipient] < block.timestamp, "Please wait for 1min between
 cooldownTimer[recipient] = block.timestamp + cooldownTimerInterval;
}

// enable cooldown between trades
function cooldownEnabled(bool _status, uint8 _interval) public onlyOwner {
 require(_interval < 60, "Can't set cooldown interval to more than 10 minutes");
}
```

## Recommendation

Change return message to valid message or change cooldownEnabled function

# 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 can 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 can 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.

- ⚠ Owner can exclude addresses from time constraint
- ⚠ Owner can exclude addresses from transaction amount constraint
- ⚠ Owner can change max wallet balance

# Notes

## Notes by Web3gold

No notes provided by the team.

## Notes by Coinsult

 No notes provided by Coinsult

# Contract Snapshot

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

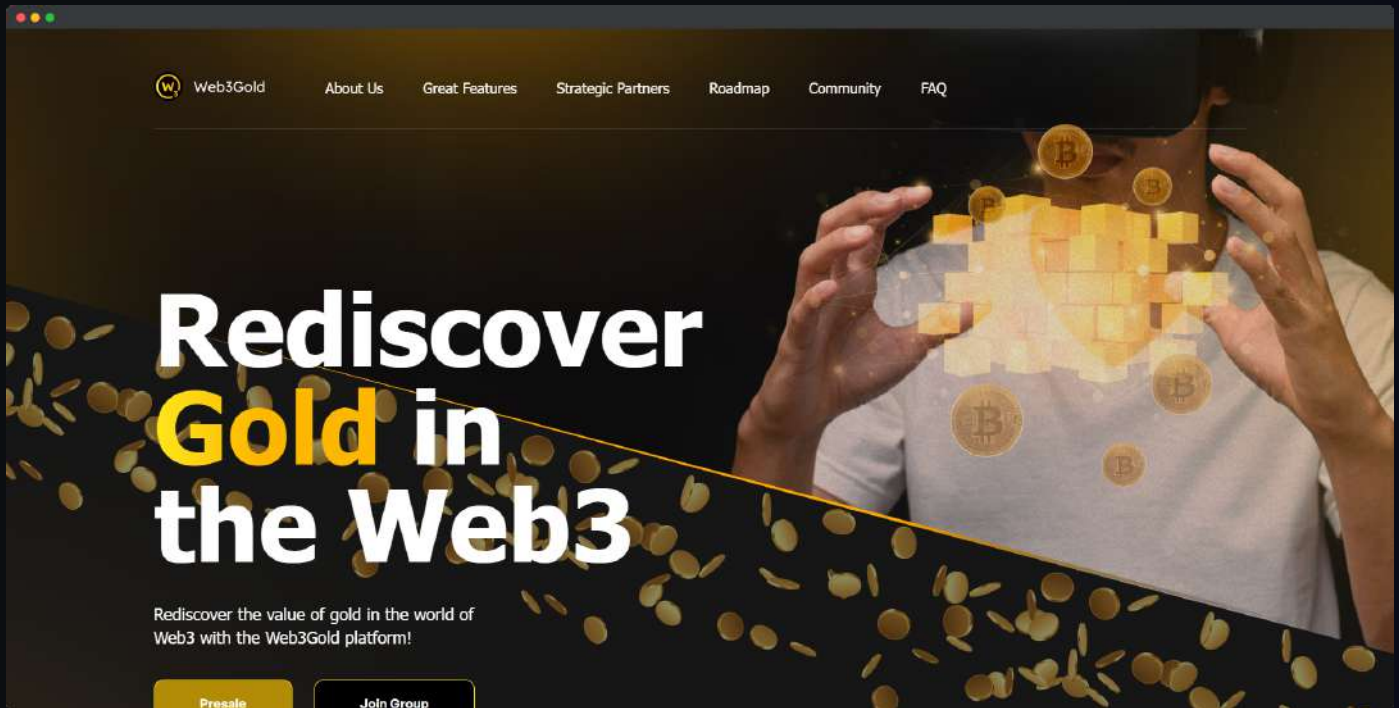
```
contract WEB3GOLD is IBEP20, Auth {
 using SafeMath for uint256;

 address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c;
 address DEAD = 0x00;
 address ZERO = 0x00;

 string constant _name = "WEB3GOLD Value-Of-Gold";
 string constant _symbol = "W3GOLD";
 uint8 constant _decimals = 18;
```

# 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

**Web3gold**  
Audited by Coinsult.net



**Date: 4 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)**