QuillAudits

Audit Report
April, 2022

For

WATCHDOG
Protocol

Contents

Overview

Watchdog by Watchdog Finance

Watchdog Protocol is a safety-focused, decentralized due diligence and investment platform, designed to set a new standard for diligent investment.

Scope of Audit

Source code for this Audit is taken from:-
https://drive.google.com/file/d/1sGoyqvUx8NhmVS_fdyYNlHnsrANd6uCP/view?usp=sharing

Fixed In: https://drive.google.com/file/d/10LsE-OA_L6VSxQybpuGA6vDfN3jUThif/view?usp=sharing

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC20 transfer() does not return boolean
- ERC20 approve() race
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis
Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis
Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption
In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Mythril, Slither, Surya, Solhint.

Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

Risk-level	Description
High	A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.
Medium	The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.
Low	Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.
Informational	These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	1	0	2
Closed	0	1	0	4

Functional Testing Results

Some of the tests performed are mentioned below:

- Should be able approve tokens
- Should be able to transfer tokens
- Should be able send all ETH to owner with rescue()
- Token transfer takes a determined amount of fee if shouldTakeFee returns true..
- _transferFrom uses _basicTransfer if liquify is executing.
- Only owner can set critical addresses and values
- Dont takes fees if it’s not launched

Issues Found – Code Review / Manual Testing

High severity issues

No issues found

Medium severity issues

1. **Approve Race Condition:**

   Changing allowance with [#L99] approve function brings the risk of race condition/frontrunning. Someone may use both old allowance and new allowance by frontrunning transaction, resulting in approval of tokens more than the sender wanted to allow if the spender tries to frontrun, as it directly assigns increased/decreased ‘amount’ to ‘_allowances[msg.sender][spender]’ instead of adding/subtracting value to increase/decrease current allowance.

```
 99     function approve(address spender, uint256 amount) public override returns (bool) {
100         _allowances[msg.sender][spender] = amount;
101         emit Approval(msg.sender, spender, amount);
102         return true;
103     }
104
```

Exploit Scenario:

1. Address A allows 50 tokens to Address B using approve() , Now A again wants to approve more tokens or lets say wants to increase allowance to 60 tokens (i.e wants to increase allowance 10 more tokens).

2. A again approves 60 tokens to B.

3. B sees this second approve transaction sent by A and tries to frontrun this transaction and spends the previous 50 tokens.

4. Once the seconds approve transaction from A get mined, B’s token approval would be 60 tokens.

5. B again spends these 60 tokens.

6. Resulting in A allowing B to spend 110 tokens.

Recommendation
Consider using mechanisms like increaseAllowance and decreaseAllowance from OpenZeppelin ERC20 smart contract implementation.

Status: Fixed

2. **Missing Zero Address Checks**

Contract lacks zero address checks, hence are prone to be initialized with zero addresses.

[#L235] setEcosystemAddress lacks zero address check for addy

[#L353] setIsFeeExempt lacks zero address check for holder

[#L379] setLiquidityReceiver lacks zero address check for _autoLiquidityReceiver

[#L383] setMarketingReceiver lacks zero address check for _receiver

[#L419] setBusdSettings lacks zero address check for busd

[#L406] setSniper lacks zero address check for snipy - fixed

Recommendation
Consider adding zero address checks in order to avoid risks of incorrect address initializations.

Status: Acknowledged
Auditors comment: Watchdog team added zero address check for setSniper and has acknowledged some issues.
“setEcosystemAddress, setLiquidityReceiver : This is by design as to leave the option to burn reward/LP” - Watchdog team

Low severity issues

No issues found

Informational issues

1. **Frontrunning/Sandwich attack:**

[#L318] buyBUSD() contains router interactions for [#L322] swapExactETHForTokensSupportingFeeOnTransferTokens, in this case the minimum amount of tokens to receive is 0, and hence the transaction is prone to frontrunning/sandwich attacks.
Exploit Scenario:

1. Contract initiates WBNB to BUSD trade:
   Here r1 and r2 are pool reserves.
   r1(WBNB) : 44963321486070818714195 6
   r2(BUSD):185500273218915585533344943
   Amount In: 250628076974462050000000 WBNB
   Amount Out for current pools : 662850861722780296683325 45 BUSD

2. Attacker frontruns above WBNB to BUSD trade initiated by contract before it executes and executes his WBNB to BUSD trade to change pool balances.
   Amount In: 450760115148579636232538000 WBNB
   Amount Out: 185314958260654930602742201 BUSD

3. Now, Contracts's trade gets executed:
   r1(WBNB): 451209748363440344419679956
   r2(BUSD): 18531495826065493060274 2
   Amount In: 250628076974462050000000 WBNB
   Amount Out: 10262048567503484563 7 BUSD
   (Amount Out is less than what it could be in normal transaction)

4. Attacker backruns contract transaction to make profit with BUSD to WBNB trade.
   r1(WBNB): 451460376440414806469679956
   r2(BUSD): 185212337774979895757105
   Amount In: 185314958260654930602742201 BUSD
   Amount Out: 451008487982902968022059206 WBNB
   Attacker makes profit of: 2483728343233317895212 06 WBNB

```
318     function buyBUSD(uint256 amount, address receiver) internal {
319         address[] memory path = new address[](2);
320         path[0] = router.WETH();
321         path[1] = address(BUSD);
322         router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: amount}(
323             0,
324             path,
325             receiver,
326             block.timestamp
327         );
328     }
329
```

Recommendation
The feasibility of attack is low as it also depends on amount of ETH the Watchdog token contract holds, However we recommend adding a minimum amount to receive greater than zero while swapping WBNB/ETH to BUSD eg: using getAmountOut from router.

Status: Acknowledged

2. **Critical Address Change**

When privileged roles are being changed, it is recommended to follow a two-step approach: 1) The current privileged role proposes a new address for the change 2) The newly proposed address then claims the privileged role in a separate transaction. This two-step change allows accidental proposals to be corrected instead of leaving the system operationally with no/malicious privileged role.
However setEcosystemAddress, setIsFeeExempt, setLiquidityReceiver, setMarketingReceiver, setRouter, setBusdSettings functions set critical addresses following a one step process.

Recommendation
Consider switching to a two-step process for transferring critical and privileged roles

Status: Acknowledged

3. **Overflow/underflow protection**

   Having overflow/underflow vulnerabilities is very common for smart contracts. In this contract many arithmetic calculations are used without a safemath library.

   Recommendation
   We recommend using SafeMath or a solidity version greater than 0.8

   Status: Fixed

4. **Different pragma directives are used**

   Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly.

   Recommendation
   Consider using the latest compiler version and locking pragma.

   Status: Fixed

5. **Owner can set max fee greater than 33%**

   [#L361] setFees and [#L370] setSaleFees calculates 33% by 1000/3 (feeDenominator / 3) which is 333. And not 330.

   Recommendation
   Consider reviewing logic so that the total of all fees would be less or equal to 33%.

   Status: Fixed

   Auditors' Comment: Watchdog team would be allowing maximum tax amount of 33.3%

6. **Some variables can be constant**

   DEAD and ZERO address type variables are not getting modified after initial declaration and can be declared as constants.

   Recommendation
   Consider declaring these variables as constants.

   Status: Fixed

Closing Summary

Some issues of Medium and Low severity were found. Some suggestions and best practices are also provided in order to improve the code quality and security posture.At the end, most of the issues are Fixed and Acknowledged by the Auditee.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the Watchdog. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Watchdog team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
April, 2022

For

WATCHDOG
Protocol

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com