QuillAudits

Audit Report
July, 2022

For

YOLO Cab

Table of Content

Executive Summary

Project Name	YoloCab
Overview	YOLO CAB is a decentralized ride-hailing platform with the goal of empowering the platform drivers and riders. YOLO CAB aspires to disrupt the existing business model and to create a fair, more efficient and transparent ride-hailing environment and a decen- tralized mobility marketplace.
Timeline	2 June, 2022 to 8 June, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse YOLO CAB ICO smart contract for quality, security, and correctness.
Deployed at	*https://testnet.bscscan.com/address/0xcebed9600c93ba011a3bd393fdecc4a843d0c7b0#code*
Fixed in	*https://github.com/Quillhash/ProjectYolocab*

17
Issues Found

🟥 High 🟨 Medium

🟩 Low 🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	**2**	**2**	**1**
Partially Resolved Issues	0	0	0	0
Resolved Issues	**1**	**1**	**6**	**4**

Types of Severities

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.

Types of Issues

Open
Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved
These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged
Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved
Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Compiler version not fixed
- ✓ Address hardcoded

Severus.finance - Audit Report

- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ 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.
- Implementation of ERC-20 token standards.
- 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

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

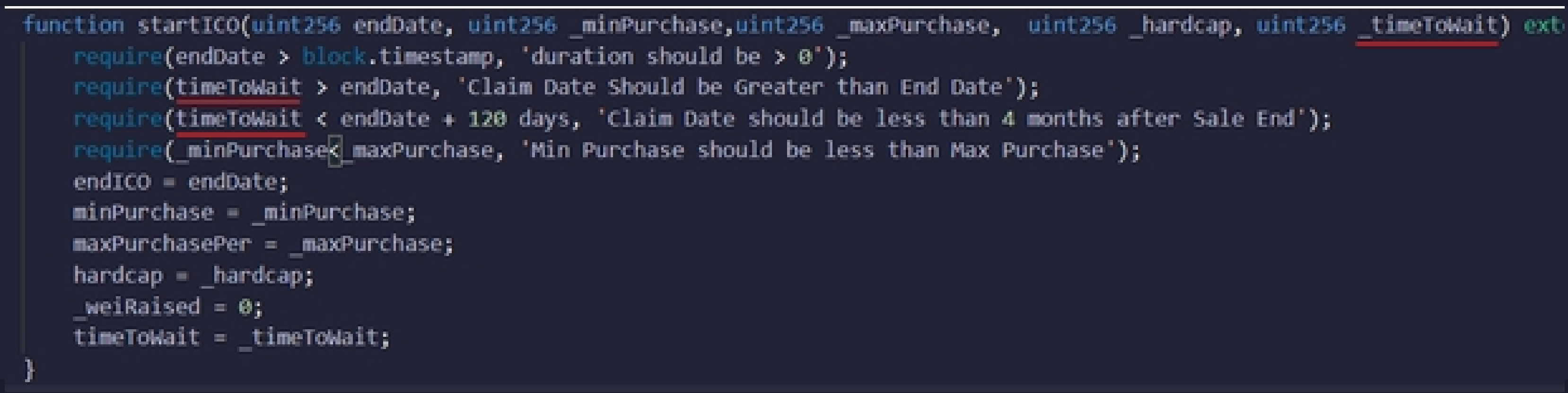
Manual Testing

High Severity Issues

1. Broken Functionality

Description

startICO function contains checks for “timeToWait should be greater than endDate” and “timeToWait should be less than endDate + 120 days” in both the checks timeToWait is used without “_” . Which is incorrect as startICO function is taking _timeToWait (with “_”) as the parameter where its taking value which is then getting assigned to timeToWait.



```
function startICO(uint256 endDate, uint256 _minPurchase,uint256 _maxPurchase,  uint256 _hardcap, uint256 _timeToWait) ext
    require(endDate > block.timestamp, 'duration should be > 0');
    require(timeToWait > endDate, 'Claim Date Should be Greater than End Date');
    require(timeToWait < endDate + 120 days, 'Claim Date should be less than 4 months after Sale End');
    require(_minPurchase<_maxPurchase, 'Min Purchase should be less than Max Purchase');
    endICO = endDate;
    minPurchase = _minPurchase;
    maxPurchasePer = _maxPurchase;
    hardcap = _hardcap;
    _weiRaised = 0;
    timeToWait = _timeToWait;
}
```

Remediation

Consider adding “_” before timeToWait for the highlighted places in require checks in the above image

Status

Fixed

Medium Severity Issues

2. min purchase should be less than max purchase

```
*/
  function startICO(uint256 endDate, uint256 _minPurchase,uint256 _maxPurchase,  uint256 _hardcap) external
onlyOwner icoNotActive() {
      require(endDate > block.timestamp, 'duration should be > 0');
      endICO = endDate;
      minPurchase = _minPurchase;
      maxPurchasePer = _maxPurchase;
      hardcap = _hardcap;
      _weiRaised = 0;
  }
```

Description

DoS- hardcap may be set to 0, as a consequence icoActive modifier will always revert as _weiRaised < hardcap (0 < 0) condition, will always fail. So, it means the ico has not been started and tokens can not be purchased. The possible way, one may think, is to stop the ico by calling stopICO function, but it can only be called if the icoActive modifer satisfies. The other way, could be to manually increase the hardcap, but hardcap can't be changed before the ico ends. So this lead to a complete DoS. This becomes worse if endICO is set to a value way far in future.

Recommendation

min purchase should be less than max purchase.

Status

Fixed

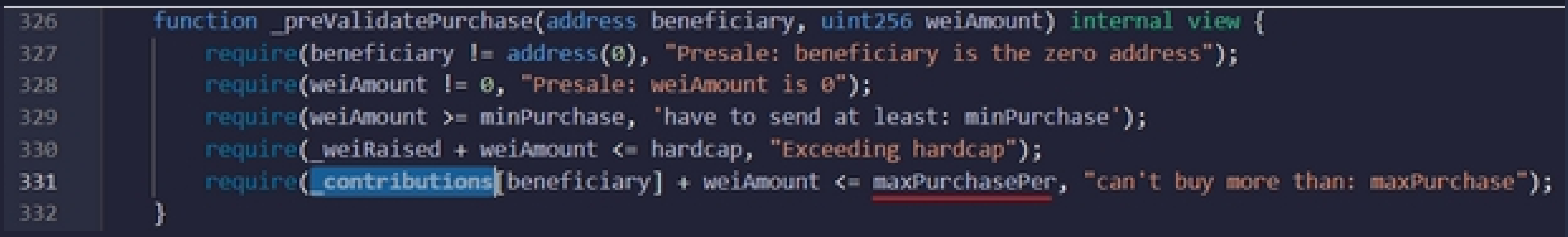
3. Check business logic 1

Description

maxPurchasePer works as how much amount of BNBs user can spend for buying tokens and that amount would be user's max purchase, it dont checks the amount of tokens that user can buy after multiplication to rate.

Example:

1. Admin starts ICO, sets maxPurchasePer to 10000000000000000000 (equals 10*(10**18))
2. User purchases tokens with buyTokens() function, e.g he purchased tokens with 1*(10**18) BNB , token rate is 10 so he gets (1*(10**18) * 10)= 10000000000000000000 (i.e 10 Tokens)
3. _preValidatePurchase function checks that a user can't buy more than maxPurchasePer, the code shows that its checking weiAmount that is getting stored in _contributions with maxPurchasePer.



```
function _preValidatePurchase(address beneficiary, uint256 weiAmount) internal view {
    require(beneficiary != address(0), "Presale: beneficiary is the zero address");
    require(weiAmount != 0, "Presale: weiAmount is 0");
    require(weiAmount >= minPurchase, 'have to send at least: minPurchase');
    require(_weiRaised + weiAmount <= hardcap, "Exceeding hardcap");
    require(_contributions[beneficiary] + weiAmount <= maxPurchasePer, "can't buy more than: maxPurchase");
}
```

4. That means user can spend more 9 * (10**18) to buy tokens, which will give 100 * (10**18) tokens (i.e 100 tokens) to user.
5. If we see the maxPurchasePer set initially then its 10000000000000000000 that is 10 tokens, but here user can buy more than 10 tokens.

Remediation

Check the implemented code matches the business requirement.

Status

Acknowledged

4. Check business logic 2

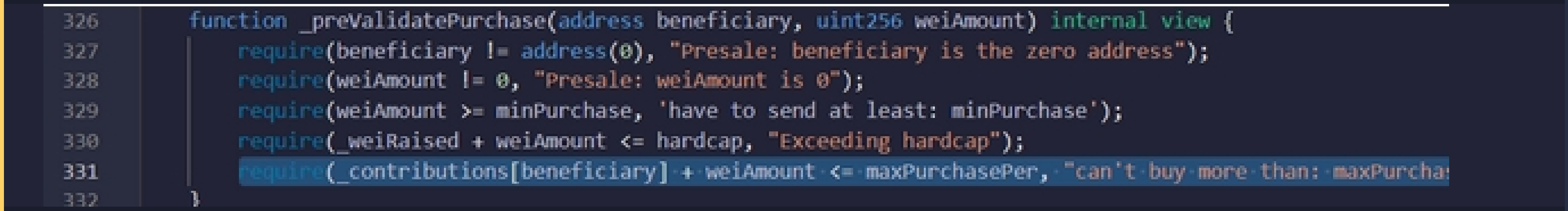
Description

While creating ICO second time with startICO() function care needs to be taken. _contributions mapping stores weiAmount that user is spending for buying tokens, In the case where user is not claiming bought tokens for first ICO and admin launches second ICO then user may not be able to buy tokens according to maxPurchasePer depending on the situation for second ico.

Example:

1. Admin starts 1st ICO.sets maxPurchasePer to 10000000000000000000 (equals 10*(10**18))
2. User A bought tokens and reached the maxPurchasePer limit.
3. User A don’t claims the token amount that he bought and now Admin launches 2nd ICO.
4. Lets say the maxPurchasePer for 2nd ICO is same as that of 1st ICO i.e 10*(10**18)
5. Now User A tries to buy tokens with buyTokens() which will call _preValidatePurchase() which checks for: require(_contributions[beneficiary] + weiAmount <= maxPurchasePer, "can't buy more than: maxPurchase");

In this case the transaction will revert here because this require statement will revert and the reason would be _contributions still holds the wei amount that user A spent but havent claimed yet.



```
function _preValidatePurchase(address beneficiary, uint256 weiAmount) internal view {
    require(beneficiary != address(0), "Presale: beneficiary is the zero address");
    require(weiAmount != 0, "Presale: weiAmount is 0");
    require(weiAmount >= minPurchase, 'have to send at least: minPurchase');
    require(_weiRaised + weiAmount <= hardcap, "Exceeding hardcap");
    require(_contributions[beneficiary] + weiAmount <= maxPurchasePer, "can't buy more than: maxPurcha
}
```

Remediation

Check business requirement, if this smart contract is getting used for more than one ICO then make sure that the condition that described in above scenario is not happening by making changes in code.

Status

Acknowledged

Low Severity Issues

5. Use of IERC20 Instead of BEP20

```
* @dev Interface of the ERC20 standard as defined in the EIP.
*/
interface IERC20 {
  /**
```

Description

Uses/refers IERC20 interface for tokens, whereas the contract is supposed to be deployed on Binance and the token is supposed to be IBEP20

Remediation

Consider using IBEP20 interface in place of IERC20.
https://github.com/bnb-chain/BEPs/blob/master/BEP20.md

Status

Fixed

6. No indexed parameters

```
event TokensPurchased(address  purchaser, uint256 value, uint256 amount);
event TokensClaimed(address  user, uint256 value, uint256 amount);
```

Description

Events tokenPurchased and tokenClaimed does not has indexed parameters.

Remediation

The indexed keyword helps you to filter the logs to find the wanted data. Thus you can search for specific items instead of getting all the logs. We recommend using indexed parameters to make it easy for offline monitoring tools to track critical operations.

Status

Fixed

7. Stopping ICO Not Possible

Description

Stopping ICO should not be possible(in order to start a new one), once the _weiRaised exceeds hardcap. The only possible way should be to wait for the current ico cycle to be ended and then start a new one, with new hardcap.

Status

Acknowledged

8. _contributions can be made private

```
mapping (address => uint256) public _contributions;
mapping (address => uint256) public totalContributions;
mapping (address => uint256) public maxPurchase;
mapping (address => uint256) public claimed;
```

Description

_contributions mapping comes with a public getter function, will show user contribution in BNB, whereas checkContribution and checkContributionExt will show contributions in tokens.

Recommendation

To avoid confusion,we recommend to make _contributions private.

Status

Fixed

9. Claim Time Should be ahead of ICO EndTime

Description

claiming time(timeToWait) should be ahead of the ico ending time, else it may happen that the user claims its current tokens, resets its contributions and then purchase more tokens in the same ico cycle.

Status

Fixed

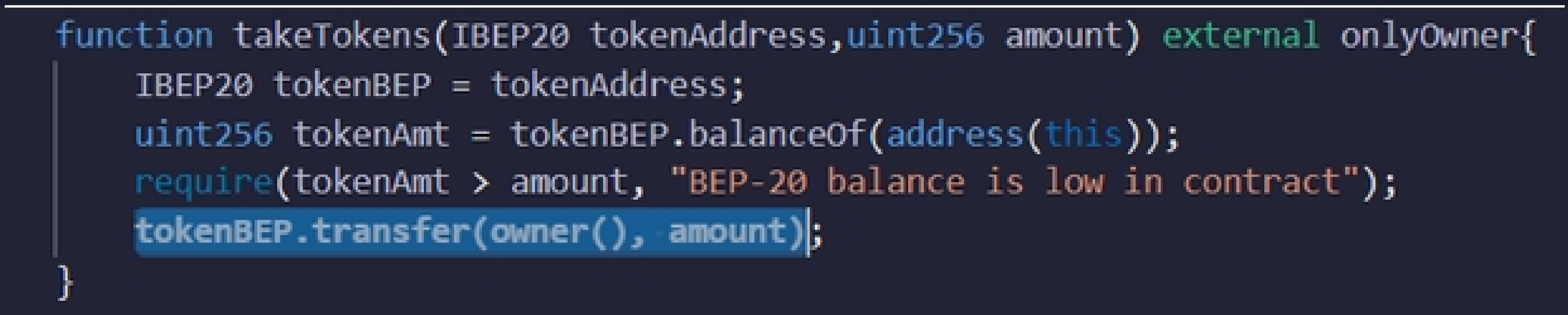
10. Missing Value check

Description

setRate(): missing value checks. Can be used to set any rate. For instance, _rate can be set to 0, rugpulling user's funds, or to a very large value, and allowing anyone to get a large amount of tokens

function setMinPurchase(), setMaxPurchase(): no value checks, minPurchase can be set more than maxPurchasePer

function takeTokens: doesn't check return value for token transfer. Consider checking return value for token transfer in takeTokens() so that it will revert the transaction in the case a type of token is used which don't reverts on token transfer failure.



```
function takeTokens(IBEP20 tokenAddress,uint256 amount) external onlyOwner{
    IBEP20 tokenBEP = tokenAddress;
    uint256 tokenAmt = tokenBEP.balanceOf(address(this));
    require(tokenAmt > amount, "BEP-20 balance is low in contract");
    tokenBEP.transfer(owner(), amount);
}
```

Recommendation

Consider adding checks.

Status

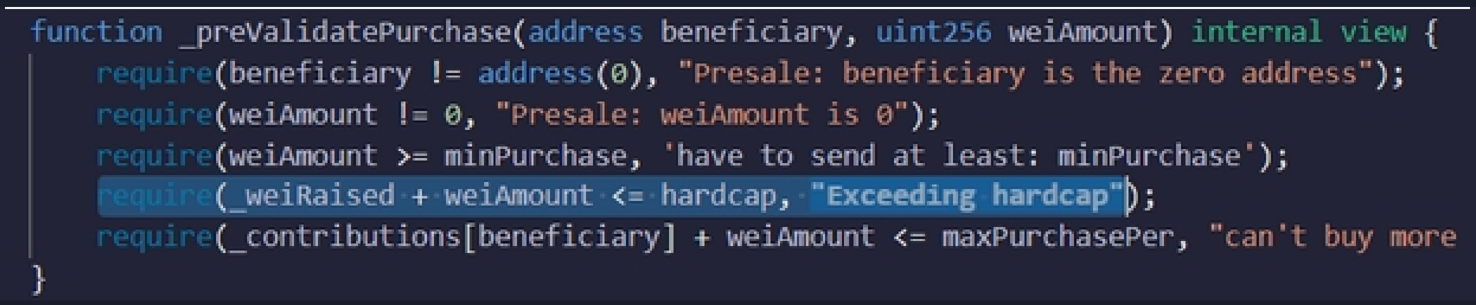
Acknowledged

Auditor's Note: For last paragraph in description above. Here the value that is returned by tokenBEP.transfer(owner(), amount); needs to be checked by required statement to be true, if it returns false then it should revert saying transfer failed error message, if non reverting ERC20 token is used then in this case it would be helpful to know if transfer is failing.

11. Possible DOS

Description

hardcap may be set to 0, in the case if it unintentionally gets set to zero then call to buyTokens() function will revert because of this check on the line 328 in _preValidatePurchase() internal function.



```
function _preValidatePurchase(address beneficiary, uint256 weiAmount) internal view {
    require(beneficiary != address(0), "Presale: beneficiary is the zero address");
    require(weiAmount != 0, "Presale: weiAmount is 0");
    require(weiAmount >= minPurchase, 'have to send at least: minPurchase');
    require(_weiRaised + weiAmount <= hardcap, "Exceeding hardcap");
    require(_contributions[beneficiary] + weiAmount <= maxPurchasePer, "can't buy more
}
```

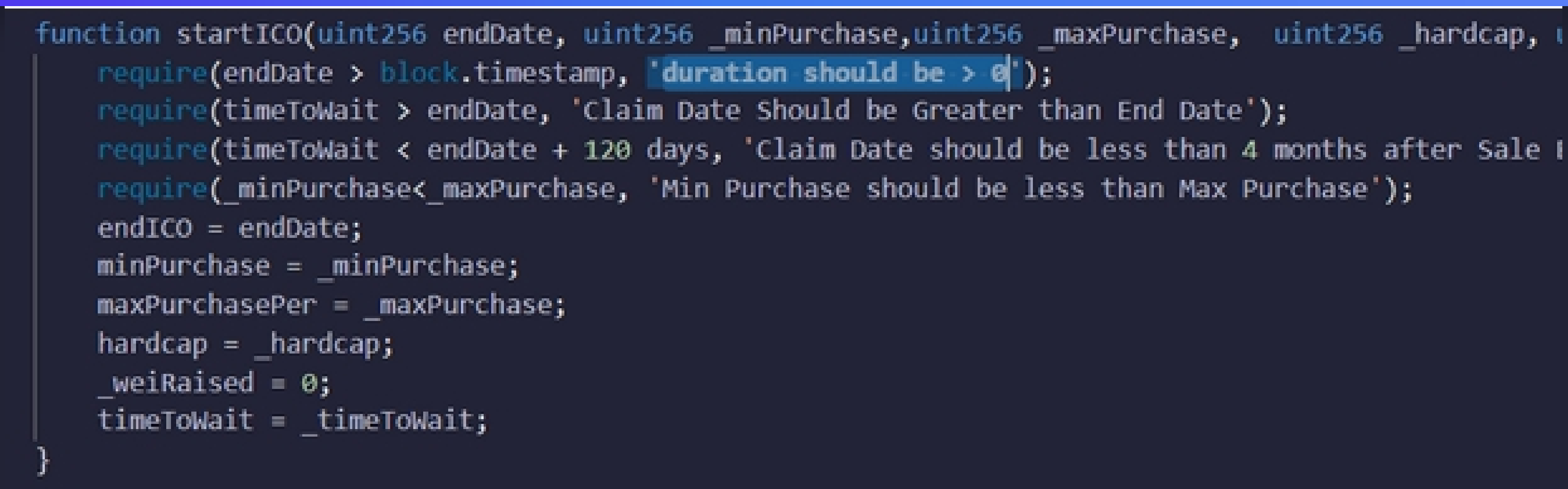
Recommendation

Check can be added in startICO function to check the entered _hardcap is greater than zero.

Status

Fixed

12. Incorrect error message



```
function startICO(uint256 endDate, uint256 _minPurchase,uint256 _maxPurchase,  uint256 _hardcap, 
    require(endDate > block.timestamp, 'duration should be > 0');
    require(timeToWait > endDate, 'Claim Date Should be Greater than End Date');
    require(timeToWait < endDate + 120 days, 'Claim Date should be less than 4 months after Sale 
    require(_minPurchase<_maxPurchase, 'Min Purchase should be less than Max Purchase');
    endICO = endDate;
    minPurchase = _minPurchase;
    maxPurchasePer = _maxPurchase;
    hardcap = _hardcap;
    _weiRaised = 0;
    timeToWait = _timeToWait;
}
```

Description

The require check that startICO() does for “endDate > block.timestamp” checks that end date of ico is greater than current timestamp but the error message don’t matches what it checks.

Recommendation

Consider adding correct error message to avoid confusion.

Status

Fixed

Informational Issues

13. Unnecessary copying of local variable into another local variable

```
function buyTokens() public payable nonReentrant icoActive{
uint256 amount = msg.value;
uint256 weiAmount = amount;
```

Status

Fixed

14. timeToWait can be set to a value way far in future

Description

constructor() - Malicious owner can set timeToWait to a value way far in future, thus disallowing anyone to claim their tokens, until and unless if owner decides to reduce/remove the claim time with function changeWaitTime

```
 constructor (uint256 rate, IERC20 token,uint256 _timeToWait)  {
    require(rate > 0, "Pre-Sale: rate is 0");
    require(address(token) != address(0), "Pre-Sale: token is the zero address");

    _rate = rate;
    _token = token;
    timeToWait = _timeToWait;
 }
```

Status

Fixed

Note: timeToWait is not getting used passed in constructor in the current fix and its getting checked in startICO function.

15. Dead Code

Description

1. Unused mapping maxPurchase and unused private variable _tokenDecimals

2. Function setMinPurchase() and setMaxPurchase() use icoNotActive modifier which ensures that this functions can't be called when ico is active, in this case these two functions can only be called after ico ends, But when ico ends new min and max purchase limit can be specified while calling startICO() which eliminates need of setMinPurchase and setMaxPurchase, hence these two functions can be removed.

Recommendation

Consider removing unused/redundant code.

Note: _tokenDecimal is getting used in current fix.
Auditor's Note: As written in description 2nd point, check the need of setMinPurchase() and setMaxPurchase() otherwise remove two functions.

Status

Acknowledged

16. Public functions can be marked external to save gas

Recommendation

buyTokens and takeTokens for instance can be made External inorder to save gas.

Status

Fixed

17. Two different functions facilitating the same business logic

Description

checkContributionExt and checkContribution: two different functions facilitating the same business logic.

Recommendation

Use public one, as it can be called both internally and externally.

Status

Fixed

Functional Testing

Some of the tests performed are mentioned below

- ✓ Should get the owner of the contract.
- ✓ Should get the Minimum Purchase.
- ✓ Should get the Maximum Purchase.
- ✓ Should get bnbCollected.
- ✓ Should get the _weiraised.
- ✓ Should get the Hardcap
- ✓ Should return the contribution
- ✓ Only the owner can change the wait time.
- ✓ Only Owner can transferOwnership
- ✓ Should Revert if ChangeWaitTime exceeds Previous wait time.
- ✓ Should Revert if the value entered < Min Purchase
- ✓ Should Revert if the value entered > Max Purchase
- ✓ Should Revert BuyToken when ICO isNotActive
- ✓ Should be able to claim tokens after timeToWait elapses.

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the Yolocab Smart Contract. We performed our audit according to the procedure described above.

Some issues of Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.At the end Yolocab team resolved few issues and Acknowledged other remaining Issues.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Yolocab Platform. 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 Yolocab Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

500+
Audits Completed

$15B
Secured

500K
Lines of Code Audited

Follow Our Journey

Audit Report
July, 2022

For

YOLO Cab

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com