# Aqua Audit Smart Contract Preliminary Audit Report

## Project Synopsis

| | |
|---|---|
| **Project Name** | **Aqua Audit** |
| **Platform** | BSC, Solidity |
| **Github Repo** | Not Provided |
| **Deployed Contract** | Not Deployed |
| **Total Duration** | Days |
| **Timeline of Audit** | **24th June April 2021 to 30th June 2021** |

## Contract Details

| | |
|---|---|
| **Total Contract(s)** | 1 |
| **Name of Contract(s)** | AqarChain |
| **Language** | Solidity |
| **Commit Hash** | **Null** |

### Contract Vulnerabilities Synopsis

| Issues | Open Issues | Closed Issues |
|---|---|---|
| Critical Severity | 3 | 0 |
| Medium Severity | 2 | 0 |
| Low Severity | 6 | 0 |
| Information | 3 | 0 |
| Total Found | 14 | 0 |

## Detailed Results

The contract has gone through several stages of the audit procedure that includes structural analysis,  automated testing, manual code review etc.

All the issues have been explained and discussed in detail below. Along with the explanation of the issue found during the audit, the recommended way to overcome the issue or improve the code quality has also been mentioned.

# A.Contract Name: AqarChain

## Critical Severity Issues

### A.1 Invalid require statement doesn't restrict user's action adequately.

**Line no - 443,465,481, 501, 520, 540,555**

**Explanation:**

The ***require*** statement at the above-mentioned lines involves an assignment operator(=) instead of an Equality Validation Operator(==).

```
554     function claim() external {
555         require(claimbool = true,"claiming amount should be true");
556
```

```
440     function seedusdt(string calldata _first,string calldata _last,s
441         require(_amount>=1000000000000000000 ,"Enter amount greate
442         require(seedamount<=70000000000000000000000000,"seed round to
443         require(seedrun = true,"seed round is not started or over");
444
```

This leads to a completely unwanted scenario where the boolean values like ***publicrun***, ***seedrun*** etc aren't being validated for TRUE or FALSE but simply being assigned a TRUE boolean value, *every time the* ***require statement*** *is executed.*

In other words, **users can execute these functions even if the Seed round or Private Round boolean value is FALSE.**

***Recommendation:***

The above-mentioned require statements should use the equality operator to impose an adequate require statement validation.

For instance,

```
  require(seedrun == true,"seed round is not started or over");
or,
  require(seedrun,"seed round is not started or over");
```

## A.2 *publicbnb* function stores Invalid Data on chain.

**Line no - 546**

**Explanation:**

As per the current design of the ***publicbnb*** function, it stores a wrong uint value to the publicamount state variable.

```
543        if(publicamount.add(msg.value.mul(getBnbRate()).mul(publicp
544          usermappublic[msg.sender]=publicUserInfo({firstname:_firs
545          amountmaptouserpublic[_id]=amountmaptouserpublic[_id].add
546          publicamount=privateamount.add(msg.value.mul(getBnbRate()
547          i++;
548          usersarr.push(msg.sender);
549          }
```

The total amount of tokens being sold at the ***Public Sale Round*** is being stored in terms of the ***Private Sale round.***

In other words, the **publicamount** state variable is being wrongly updated as it stores the value of tokens sold by adding it to the **privateamount** state varible instead of the **publicamount** state variable.

This will lead to a completely unwanted scenario where the data stored on chain about the total tokens Sold in the public round will be different from the actual tokens sold in the public round.

***Recommendation:***

The Line no 546 in the ***publicbnb*** function should be modified as follows:

```
publicamount=privateamount.add(msg.value.mul(getBnbRate()).mul(public
price).div(1e18).div(10));
```

### A.3 Invalid Require statement could make "*publicBnb*" function completely inaccessible.

**Line no - 541**

***Explanation:***

The ***publicbnb*** function includes a require statement in the above-mentioned line which checks whether or not the State Variable ***privateamount*** is less than Public Sale supply.

```
534     function publicbnb(string calldata _first,string calldata _last,string calldata _country,string
535         // user enter amount of ether which is then transfered into the smart contract and tokens to
536         require(
537              msg.value.mul(getBnbRate()).div(1e18) >= 100000000000000000000 ,
538             "the input bnb amount should be greater than hundred and less than sfivethousand"
539         );
540         require(publicrun=true,"Public sale haven't started yet");
541         require(privateamount<=100000000000000000000000,"private round token sale completed");
542
```

This is an inaccurate validation as the ***require*** statement checks the Public Sale Supply limit with the ***privateamount*** state variable instead of ***publicamount*** state variable.

Moreover, since the ***Private Sale*** supply(**12,000,000**)is comparatively larger than the ***Public Sale supply(1,000,000***), the ***privateamount*** state variable might be greater than the Public Sale supply limit at any instance. At that point, the ***require*** statement at line 541 shall never qualify.
This will lead to an unexpected scenario where the ***publicbnb*** function will be completely inaccessible and will never get executed.

***Recommendation:***

The require statement of ***publicbnb*** function at the above mentioned line should be updated as follows:

```
require(publicamount<=1000000 ether,"Public token sale completed");
```

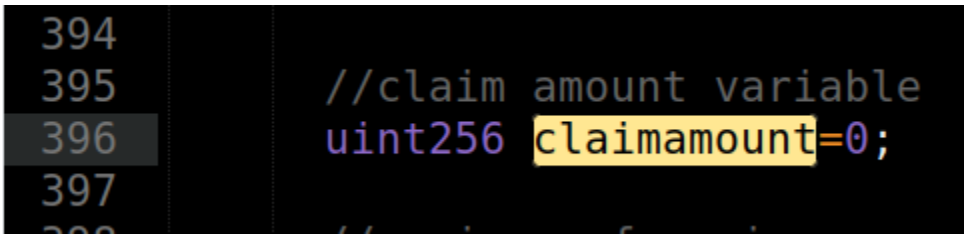
## Medium Severity Issues

### A.4 State Variable "*claimamount*" has no significant usage in the Protocol.

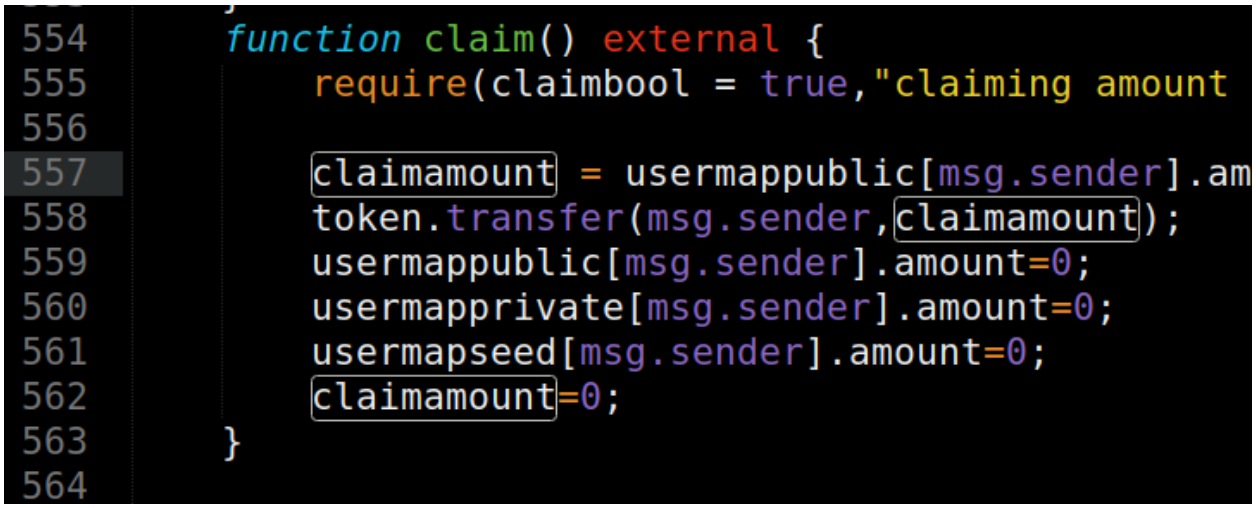
**Line no - 396,557,562**

***Explanation:***

*The State variable claim amount has no significant usage in the Contract, as per the current design of the protocol.*




*The variable is used to store the total claimable amount of a user which is then transferred to the user. However, once transferred, the claimable state variable is assigned a Zero Value again.*




In simpler terms, the ***claimamount*** state variable shall always hold a **Zero** Value and never symbolize any imperative state change in the protocol.

Using claimamount as a State Variable unncessarily uses extra Spaces and affects the Gas Usage in the contract.

***Is the USE of claimamount as a State Variable intended?***

***Recommendation:***
If the above-mentioned scenario is not intended, it is recommended to modify the ***claimamount*** variable as a local variable instead of a state variable.

## A.5 Violation of Check Effects Interaction pattern

***Explanation:***
The **AqarChain** contract includes a few functions that update some of the very imperative state variables of the contract after the external calls are made.

An external call within a function technically shifts the control flow of the contract to another contract for a particular period of time. Therefore, as per the Solidity Guidelines, any modification of the state variables in the base contract must be performed before executing the external call.

The following functions in the contract update the state variables  after making an external call at the lines mentioned below:

- ***seedusdt()*** *at Line* ***446***
- ***privateusdt()*** *at Line* ***486***
- ***publicusdt()*** *at Line* ***525***
- ***claim()*** *at Line* ***558***

```
554     function claim() external {
555         require(claimbool = true,"claiming amount should be true");
556 
557         claimamount = usermappublic[msg.sender].amount.add(usermapse
558         token.transfer(msg.sender,claimamount);
559         usermappublic[msg.sender].amount=0;
560         usermapprivate[msg.sender].amount=0;
561         usermapseed[msg.sender].amount=0;
562         claimamount=0;
```

***Recommendation:***
**Check Effects Interaction Pattern** must be followed while implementing external calls in a function.

## Low Severity Issues

### A.6 Redundant State Variable Update

**Line no: 393,396,409,410,411,412**

**Explanation**

The AquarChain Smart contract involves redundant updating of some of the State variables in the contract.

```
409         bool public seedrun = false;
410         bool public privaterun = false;
411         bool public publicrun = false;
412         bool public claimbool = false;
413 
```

A boolean variable is by-default initialized to FALSE whereas a uint256 is initialized to ZERO. Hence, such state variables do not need to be initialized explicitly.

***Recommendation:***
*Redundant initialization of state variables should be avoided.*

## A.7 Require statements can be used instead of IF and REVERT Statements

**Line no - 454, 475, 492, 512, 531,551**
**Explanation:**
The function at the above-mentioned lines uses IF-REVERT statements to ensure that users do not buy tokens more than the allowed token supply for each round.

However, this is a strict validation as the users should not be able to execute the function if this IF statement fails. Therefore, it is considered a better practise in Solidity Smart Contracts, to use **require statements for such validations.**

**Is this Function Design Intended?**

**Recommendation:**
The IF-REVERT statements can be modified as follows, unless the current function design is Intended.

```
if(seedamount.add(_amount.mul(seedprice))<=700000000000000000000000)
{
      // Logic
} else{
            revert("try reducing amount or seed round is finished")
}
```

The above-mentioned IF ELSE and Revert statement can be re-written as:

```
require(seedamount.add(_amount.mul(seedprice))<=700000000000000000000
0000,"try reducing amount or seed round is finished")
{
      // Logic
```

```
}
```

## A.8 Functions promise a return Value of uint256 but do not return anything.

**Line no: 569, 572, 575, 578**

**Explanation**

The functions at the above-mentioned lines indicate a uint256 return value at their function signature.

```
569     function toggleclaim() external onlyOwner returns (uint256) {
570         claimbool = !claimbool;
571     }
572     function toggleseed() external onlyOwner returns (uint256) {
573         seedrun = !seedrun;
574     }
575      function toggleprivate() external onlyOwner returns (uint256) {
576         privaterun = !privaterun;
577     }
578       function togglepublic() external onlyOwner returns (uint256) {
579         publicrun = !publicrun;
580     }
```

However, none of those functions actually return any uint256 value. If no uint value is not explicitly returned, the function will simply return a default return value for uint256, i.e., ZERO.

***Recommendation:***

*If the above-mentioned functions are not supposed to return any uint256 value, the function signatures should be modified accordingly.*

## A.9 External Visibility should be preferred

**Explanation**

Those functions that are never called throughout the contract should be marked as ***external*** visibility instead of ***public*** visibility.
This will effectively result in Gas Optimization as well.

Therefore, the following function must be marked as **external** within the contract:

- ***getBnbBalance()***

***Recommendation:***
If the PUBLIC visibility of the above-mentioned functions is not intended, then the EXTERNAL Visibility keyword should be preferred.

## A.10 Constant declaration should be preferred

**Line no - 399 to 401**

**Explanation**

State variables that are not supposed to change throughout the contract should be declared as **constant**.

***Recommendation:***

The following state variables could be declared as **constant**, unless the current contract design is intended.

- ***privateprice***
- ***publicprice***
- ***Seedprice***

## A.11 Too many Digits used

**Line no - 440-456, 458-477, 478-494, 495-514, 516-533, 534-553**

**Explanation**

The above-mentioned lines have a large number of digits that makes it difficult to review and reduces the readability of the code.

The following functions in the contract have this issue:

- **seedusdt()**
- **seedbnb()**
- **privateusdt**
- **privatebnb**
- **publicusdt**
- **publicbnb**

```
    }
    function seedusdt(string calldata _first,string calldata _last,string calldata _country
        require(_amount>=10000000000000000000 ,"Enter amount greater than 100 usd");
        require(seedamount<=700000000000000000000000,"seed round token sale completed");
        require(seedrun = true,"seed round is not started or over");
```

***Recommendation:***

Ether Suffix could be used to symbolize the 10^18 zeros.

For instance, the require statement at Line number 441,

```
require(_amount>=10000000000000000000 ,"Enter amount greater than
100 usd");
```

*Can be written as:*

```
require(_amount>=100 ether ,"Enter amount greater than 100 usd");
```

# Informational

## A.12 Contract includes Hardcoded address

**Line no: 418, 425, 429, 430**

**Explanation**

Keeping in mind the immutable nature of smart contracts, it is not considered a better practise to hardcode any address in the contract before deployment.

***Recommendation:***

Instead of including hardcoded addresses in the contract, initialize those addresses within the constructors at the time of deployment.

## A.13 Code Style Issues

**Explanation**

Code readability of a Smart Contract is largely influenced by the Coding Style issues and in some specific scenarios may lead to bugs in the future.

During the automated testing, it was found that the **AqarChain** contract had quite a few code style issues.

```
Struct aqarchain.seedUserInfo (contracts/Updated_AQR.sol#352-360) is not in CapWords
Struct aqarchain.privateUserInfo (contracts/Updated_AQR.sol#361-369) is not in CapWords
Struct aqarchain.publicUserInfo (contracts/Updated_AQR.sol#370-378) is not in CapWords
Parameter aqarchain.settoken(address)._token (contracts/Updated_AQR.sol#437) is not in mixedCase
Parameter aqarchain.seedusdt(string,string,string,string,uint256)._first (contracts/Updated_AQR.sol#440) is not in mixedCase
Parameter aqarchain.seedusdt(string,string,string,string,uint256)._last (contracts/Updated_AQR.sol#440) is not in mixedCase
Parameter aqarchain.seedusdt(string,string,string,string,uint256)._country (contracts/Updated_AQR.sol#440) is not in mixedCase
Parameter aqarchain.seedusdt(string,string,string,string,uint256)._id (contracts/Updated_AQR.sol#440) is not in mixedCase
Parameter aqarchain.seedusdt(string,string,string,string,uint256)._amount (contracts/Updated_AQR.sol#440) is not in mixedCase
Parameter aqarchain.seedbnb(string,string,string,string)._first (contracts/Updated_AQR.sol#458) is not in mixedCase
Parameter aqarchain.seedbnb(string,string,string,string)._last (contracts/Updated_AQR.sol#458) is not in mixedCase
Parameter aqarchain.seedbnb(string,string,string,string)._country (contracts/Updated_AQR.sol#458) is not in mixedCase
```

***Recommendation:***

Therefore, it is recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

## A.14 Commented codes must be wiped-out before deployment

**Line no: 236-276**

***Explanation***

The AqarChain contract includes quite a few commented codes regarding a **INonStandardERC20 interface** at the above-mentioned line.

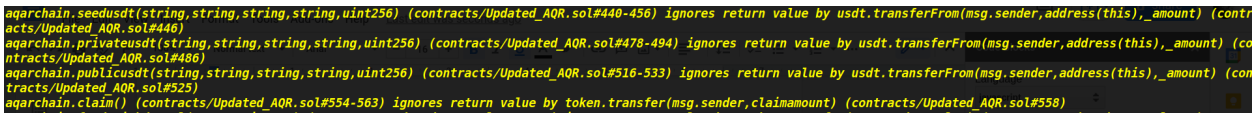
```
236  // interface INonStandardERC20 {
237  //     function totalSupply() external view returns (uint256);
238  //
239  //     function balanceOf(address owner) external view returns (uint256 balance);
240  //
241  //     ///
242  //     /// !!!!!!!!!!!!!!
243  //     /// !!! NOTICE !!! `transfer` does not return a value, in violation of the ERC-20 specification
244  //     /// !!!!!!!!!!!!!!
245  //     ///
246  //
247  //     function transfer(address dst, uint256 amount) external;
248  //
249  //     ///
250  //     /// !!!!!!!!!!!!!!
251  //     /// !!! NOTICE !!! `transferFrom` does not return a value, in violation of the ERC-20 specification
252  //     /// !!!!!!!!!!!!!!
```

This badly affects the readability of the code.

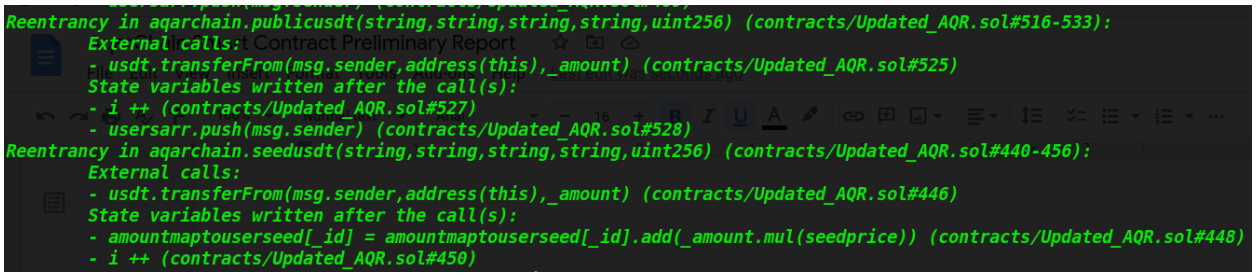
***Recommendation:***

Therefore, it is recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

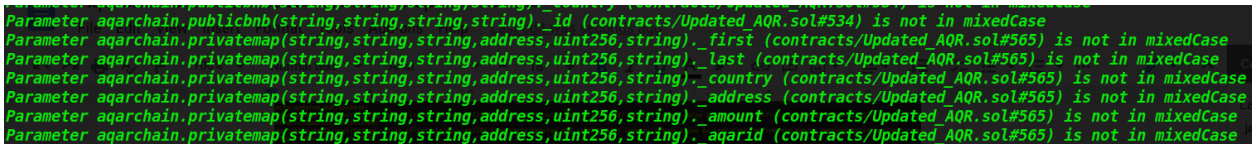
# Automated Test Results



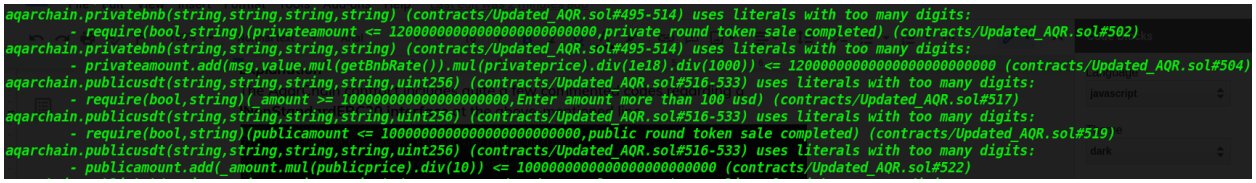
```
aqarchain.seedusdt(string,string,string,string,uint256) (contracts/Updated_AQR.sol#440-456) ignores return value by usdt.transferFrom(msg.sender,address(this),_amount) (contr
acts/Updated_AQR.sol#446)
aqarchain.privateusdt(string,string,string,string,uint256) (contracts/Updated_AQR.sol#478-494) ignores return value by usdt.transferFrom(msg.sender,address(this),_amount) (co
ntracts/Updated_AQR.sol#486)
aqarchain.publicusdt(string,string,string,string,uint256) (contracts/Updated_AQR.sol#516-533) ignores return value by usdt.transferFrom(msg.sender,address(this),_amount) (con
tracts/Updated_AQR.sol#525)
aqarchain.claim() (contracts/Updated_AQR.sol#554-563) ignores return value by token.transfer(msg.sender,claimamount) (contracts/Updated_AQR.sol#558)
```



```
Reentrancy in aqarchain.publicusdt(string,string,string,string,uint256) (contracts/Updated_AQR.sol#516-533):
        External calls:
        - usdt.transferFrom(msg.sender,address(this),_amount) (contracts/Updated_AQR.sol#525)
        State variables written after the call(s):
        - i ++ (contracts/Updated_AQR.sol#527)
        - usersarr.push(msg.sender) (contracts/Updated_AQR.sol#528)
Reentrancy in aqarchain.seedusdt(string,string,string,string,uint256) (contracts/Updated_AQR.sol#440-456):
        External calls:
        - usdt.transferFrom(msg.sender,address(this),_amount) (contracts/Updated_AQR.sol#446)
        State variables written after the call(s):
        - amountmaptouserseed[_id] = amountmaptouserseed[_id].add(_amount.mul(seedprice)) (contracts/Updated_AQR.sol#448)
        - i ++ (contracts/Updated_AQR.sol#450)
```



```
Parameter aqarchain.publicbnb(string,string,string,string)._id (contracts/Updated_AQR.sol#534) is not in mixedCase
Parameter aqarchain.privatemap(string,string,string,address,uint256,string)._first (contracts/Updated_AQR.sol#565) is not in mixedCase
Parameter aqarchain.privatemap(string,string,string,address,uint256,string)._last (contracts/Updated_AQR.sol#565) is not in mixedCase
Parameter aqarchain.privatemap(string,string,string,address,uint256,string)._country (contracts/Updated_AQR.sol#565) is not in mixedCase
Parameter aqarchain.privatemap(string,string,string,address,uint256,string)._address (contracts/Updated_AQR.sol#565) is not in mixedCase
Parameter aqarchain.privatemap(string,string,string,address,uint256,string)._amount (contracts/Updated_AQR.sol#565) is not in mixedCase
Parameter aqarchain.privatemap(string,string,string,address,uint256,string)._aqarid (contracts/Updated_AQR.sol#565) is not in mixedCase
```



```
aqarchain.privatebnb(string,string,string,string) (contracts/Updated_AQR.sol#495-514) uses literals with too many digits:
        - require(bool,string)(privateamount <= 120000000000000000000000000,private round token sale completed) (contracts/Updated_AQR.sol#502)
aqarchain.privatebnb(string,string,string,string) (contracts/Updated_AQR.sol#495-514) uses literals with too many digits:
        - privateamount.add(msg.value.mul(getBnbRate()).mul(privateprice).div(1e18).div(1000)) <= 120000000000000000000000000 (contracts/Updated_AQR.sol#504)
aqarchain.publicusdt(string,string,string,string,uint256) (contracts/Updated_AQR.sol#516-533) uses literals with too many digits:
        - require(bool,string)(_amount >= 1000000000000000000000,Enter amount more than 100 usd) (contracts/Updated_AQR.sol#517)
aqarchain.publicusdt(string,string,string,string,uint256) (contracts/Updated_AQR.sol#516-533) uses literals with too many digits:
        - require(bool,string)(publicamount <= 100000000000000000000000000,public round token sale completed) (contracts/Updated_AQR.sol#519)
aqarchain.publicusdt(string,string,string,string,uint256) (contracts/Updated_AQR.sol#516-533) uses literals with too many digits:
        - publicamount.add(_amount.mul(publicprice).div(10)) <= 100000000000000000000000000 (contracts/Updated_AQR.sol#522)
```