RichBullsClub

RichBulls

Smart Contract Audit Report

December 02, 2021

Introduction

1. About RichBullsClub

The Rich Bulls Club is a collection of 9999 bullish NFTs—unique digital collectibles living on the Ethereum blockchain. With over 170+ hand-drawn traits, your exclusive NFT serves as a membership to an elite club with multiple members-only benefits.

Visit https://richbullsclub.io/ to know more about.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 105+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

Documentation Details

The Rich Bulls Club team has provided the following doc for the purpose of audit:

1. A short description of the project was over email.

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: Rich Bulls Club
- Contracts Name: RichBulls
- Languages: Solidity(Smart contract)
- Github commit/Smart Contract Address for audit: https://www.toptal.com/developers/hastebin/raw/ecijibalaq
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level References

Every issue in this report was assigned a severity level from the following:

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	1	-	6
Closed	-	-	-

High Severity Issues

1. **Absence of ETHER refund procedure in mint functionalities**
   Line no: 61, 105
   Description:
   The **mint()** and **presaleMint()** functions in the contract mints NFT for the specific amount of ether passed to the function.

   Both of the above-mentioned functions, however, allow the caller to pass **more ETH than required** but don't include any procedure in the function to refund back the extra **ETH** sent while calling the function.

   No adequate refund mechanism has been written in the function that ensures the transfer of the extra ETH back to the user. This might not be an ideal scenario.

   Recommendation:
   The function should be modified to effectively handle the above-mentioned cases.

Medium Severity Issues

No issues were found.

Low Severity Issues

1. **Loops are extremely costly**
   Line no - 127, 136, 145
   Description:

   The **RichBulls** contract includes some specific **for loops** in the contract that uses state variables like .length of a non-memory array, in the condition of the for loops.

```
126     function isWhitelisted(address _user) public view returns (bool) {
127         for (uint256 i = 0; i < whitelistedAddresses.length; i++) {
128             if (whitelistedAddresses[i] == _user) {
129                 return true;
130             }
131         }
132         return false;
133     }
```

As a result, these state variables consume a lot more extra gas for every iteration of the for a loop.
The following function includes such loops at the mentioned lines:

- **isWhitelisted() at Line 127**
- **isVip() at Line 136**
- **isWhitelistToken at Line 145**

Recommendation:
It's quite effective to use a local variable instead of a state variable like .length in a loop. This will be a significant step in optimizing gas usage.
For instance,

```
function isVip(address _user) public view returns (bool) {
      local_variable = vipAddresses.length;
       for (uint256 i = 0; i < local_variable; i++) {
          if (vipAddresses[i] == _user) {
             return true;
          }
       }
       return false;
   }
```

2. **External Visibility should be preferred**
   Description:
   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:
   - **mint()**
   - **presaleMint()**
   - **walletOfOwner()**
   - **airDropWhitelistTokens()**
   - **reveal()**
   - **setPublicCost()**
   - **setPresaleCost()**
   - **setBaseExtension()**
   - **setPaused()**
   - **toggleWhitelistMint()**
   - **toggleVipMint()**
   - **whitelistUsers()**
   - **vipUsers()**
   - **withdraw()**

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

3. **Constant declaration should be preferred**
   Line no: 17, 20, 22, 23
   Description:
   State variables that are not supposed to change throughout the contract should be declared as **constant**.

   Recommendation:
   The following state variables need to be declared as **constant** unless the current contract design is intended.
   - **maxSupply**
   - **maxPresaleSupply**
   - **maxVipNfts**
   - **maxWhitelistNfts**

4. **Redundant comparisons to boolean Constants**
   Line no: 64, 107, 178
   Description:
   Boolean constants can directly be used in conditional statements or require statements.
   Therefore, it's not considered a better practice to explicitly use **TRUE or FALSE** in the **require()** statements.

   Recommendation:
   The equality to boolean constants could be removed from the above-mentioned line.

5. **No Events emitted after imperative State Variable modification**
   Line no - 220-222, 224-226
   Description:
   Functions that update an imperative arithmetic state variable contract should emit an event after the update.
   The following functions modify some crucial arithmetic parameters like **publicCost**, **presaleCost**, etc in the contract but don't emit any event after that:
   - **setPublicCost()**
   - **setPresaleCost()**

   Since there is no event emitted on updating these variables, it might be difficult to track it off-chain.

Recommendation:
An event should be fired after changing crucial arithmetic state variables.

6. **Absence of adequate input validation in imperative functions**
   Line no - 220-222, 224-226
   Description:
   RichBulls contracts include some imperative state-modifying functions like **setPublicCost()** as well as **setPresaleCost()** that updates critical state variables of the contract.

   However, it was found that no adequate input validations have been included in the function to ensure that only valid arguments are passed to that function.
   Additionally, if there is any specific threshold for the public or presale cost value, that range could also be explicitly mentioned in the **require()** statements of the functions. This will effectively ensure that no value out of a particular range is passed as an argument for the above-mentioned functions.

   Recommendation:
   Adequate input validations must be included in critical state-modifying functions.

Recommendations

1. **Similar error messages in require statements should be avoided**
   Line no - 67-110, 82-87
   Description:
   More than one require statements in some specific function in the contract include similar error messages.

   While this reduces code readability, it also leads to an inadequate experience as more than one revert will lead to similar error message which makes it difficult to trace back to the actual reason behind a function revert.

   Recommendation:
   Its always a better practice to use unique error messages for a specific require statement.

2. **Coding Style Issues in the Contract**
   Description:
   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 RichBulls contract had quite a few code style issues.

```
Parameter RichBulls.presaleMint(uint256)._mintAmount (FlatContract.sol#1344) is not in mixedCase
Parameter RichBulls.mint(uint256)._mintAmount (FlatContract.sol#1388) is not in mixedCase
Parameter RichBulls.isWhitelisted(address)._user (FlatContract.sol#1409) is not in mixedCase
Parameter RichBulls.isVip(address)._user (FlatContract.sol#1418) is not in mixedCase
Parameter RichBulls.walletOfOwner(address)._owner (FlatContract.sol#1436) is not in mixedCase
Parameter RichBulls.setPublicCost(uint256)._newCost (FlatContract.sol#1503) is not in mixedCase
Parameter RichBulls.setPresaleCost(uint256)._newCost (FlatContract.sol#1507) is not in mixedCase
Parameter RichBulls.setBaseURI(string)._newBaseURI (FlatContract.sol#1511) is not in mixedCase
Parameter RichBulls.setBaseExtension(string)._newBaseExtension (FlatContract.sol#1515) is not in mixedCase
Parameter RichBulls.setNotRevealedURI(string)._notRevealedURI (FlatContract.sol#1522) is not in mixedCase
Parameter RichBulls.setWhitelistNotRevealedURI(string)._whitelistNotRevealedURI (FlatContract.sol#1526) is not in mixedCase
Parameter RichBulls.setPaused(bool)._state (FlatContract.sol#1533) is not in mixedCase
Parameter RichBulls.toggleWhitelistMint(bool)._state (FlatContract.sol#1537) is not in mixedCase
Parameter RichBulls.toggleVipMint(bool)._state (FlatContract.sol#1541) is not in mixedCase
Parameter RichBulls.whitelistUsers(address[])._users (FlatContract.sol#1545) is not in mixedCase
Parameter RichBulls.vipUsers(address[])._users (FlatContract.sol#1550) 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.

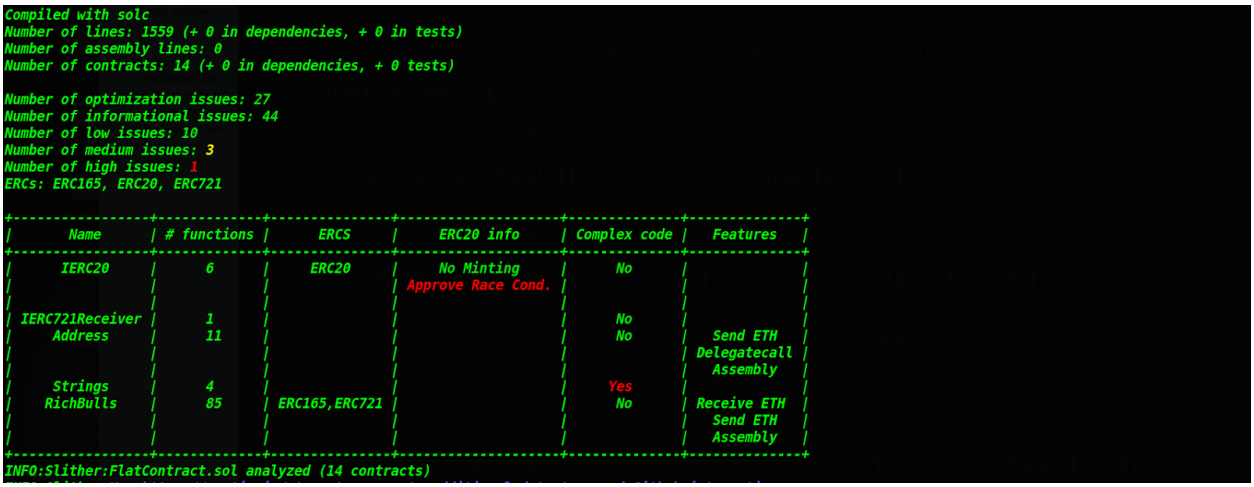
3. **NatSpec Annotations must be included**
   Description:
   The smart contracts do not include the NatSpec annotations adequately.

   Recommendation:
   Cover by NatSpec all Contract methods.

4. **No Test Cases were found**
   Description:
   No test cases were found during the audit procedure of the contract. Test cases effectively help in ensuring that the contract behaves in an intended manner.

   Recommendation:
   Test cases should be included in the contract to ensure all possible scenarios of the contract execution have been covered.

Automated Audit Result



```
Compiled with solc
Number of lines: 1559 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 14 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 27
Number of informational issues: 44
Number of low issues: 10
Number of medium issues: 3
Number of high issues: 1
ERCs: ERC165, ERC20, ERC721

+----------------+-------------+--------------+--------------------+--------------+--------------+
|      Name      | # functions |     ERCS     |     ERC20 info     | Complex code |   Features   |
+----------------+-------------+--------------+--------------------+--------------+--------------+
|     IERC20     |      6      |    ERC20     |     No Minting     |      No      |              |
|                |             |              | Approve Race Cond. |              |              |
|                |             |              |                    |              |              |
| IERC721Receiver |      1      |              |                    |      No      |              |
|     Address    |      11     |              |                    |      No      |   Send ETH   |
|                |             |              |                    |              | Delegatecall |
|                |             |              |                    |              |   Assembly   |
|     Strings    |      4      |              |                    |     Yes      |              |
|    RichBulls   |      85     | ERC165,ERC721 |                    |      No      | Receive ETH  |
|                |             |              |                    |              |   Send ETH   |
|                |             |              |                    |              |   Assembly   |
+----------------+-------------+--------------+--------------------+--------------+--------------+
INFO:Slither:FlatContract.sol analyzed (14 contracts)
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Concluding Remarks

While conducting the audits of the RichBullsClub smart contract, it was observed that the contracts contain High and Low severity issues.

Our auditors suggest that High and Low severity issues should be resolved by the developers. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the RichBullsClub platform or its product nor this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes