IMMUNEBYTES
Audits

SMART CONTRACT AUDIT
FINAL REPORT

February 14, 2022

TOC

TABLE OF CONTENTS

Introduction

1. About The YellowHeart

YellowHeart's online marketplace, combined with its mobile and desktop browser applications, make it easy for any fan to buy and sell NFT tickets and collectibles. The credit-card enabled platform is built for everyone, no crypto experience required; it pairs ease of use with the robust performance required by top tier venues.

Visit https://yh.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 125+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-ups 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 Winkies team has provided the following doc for the purpose of audit:

1. https://docs.google.com/document/d/1J9rVVmXPKLDiAEAGb7tw1m0JSf3YnjPtNVGkOGM4RWk/edit

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: YellowHeart
- Contracts Name: HeartToken
- Languages: Solidity(Smart contract)
- Contract Link for the audit: https://goerli.etherscan.io/address/0x09Df9984CfFF401F2A0a9Ff13DD254490fA0c25c#code
- 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 Reference

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

Admin/Owner Privileges can be misused either intentionally or unintentionally.
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
Closed	-	-	-

Contract Name: HeartToken

High Severity Issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

1. **Lack of adequate input validations in the minter configuration functions**

   Explanation: 3460-3470, 3477-3482
   During the code review, it was found that the **configureMinter** and **removeMinter** function in the **BaseToken** contract doesn't include adequate input validation on the arguments being passed.

```
3460    function configureMinter(address minter, uint256 minterAllowedAmount)
3461        external
3462        whenNotPaused
3463        onlyMasterMinter
3464        returns (bool)
3465    {
3466        _s.minters[minter] = true;
3467        _s.minterAllowed[minter] = minterAllowedAmount;
3468        emit MinterConfigured(minter, minterAllowedAmount);
3469        return true;
3470    }
3471
```

   For instance, before initiating the actual function execution the configureMinter function doesn't validate if the minter address being passed as an argument is already marked as true.

   Moreover, the function also doesn't involve any check to ensure the right thresholds of the minterAllowedAmount argument that is being passed to the function. It is considered a better practice to pre-define the specific highest and lowest thresholds for an arithmetic value update.

   Recommendation:
   Unless the current contract design is intended, it is recommended to include adequate require statements in the above-mentioned functions to ensure the entry of only valid values as arguments to the function.

Recommendations/Informational

1. **External Visibility should be preferred**

   Explanation:
   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 HeartToken contract:

   - **initialize()**

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

2. **Coding Style Issues in the Contract**

   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.

```
Parameter BaseToken.mint(address,uint256)._to (contracts/HeartToken.sol#3329) is not in mixedCase
Parameter BaseToken.mint(address,uint256)._amount (contracts/HeartToken.sol#3329) is not in mixedCase
Parameter BaseToken.burn(uint256)._amount (contracts/HeartToken.sol#3490) is not in mixedCase
Parameter BaseToken.updateMasterMinter(address)._newMasterMinter (contracts/HeartToken.sol#3512) is not in mixedCase
Variable BaseToken._s (contracts/HeartToken.sol#3236) is not in mixedCase
Variable BaseToken._end (contracts/HeartToken.sol#3237) is not in mixedCase
```

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

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

Automated Audit Result

```
Compiled with solc
Number of lines: 3493 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 33 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 2
Number of informational issues: 63
Number of low issues: 1
Number of medium issues: 5
Number of high issues: 1
ERCs: ERC165, ERC20

+--------------------+-------------+--------------+--------------------+--------------+--------------------+
|        Name        | # functions |     ERCS     |     ERC20 info     | Complex code |      Features      |
+--------------------+-------------+--------------+--------------------+--------------+--------------------+
|    LibAppStorage   |      1      |              |                    |      No      |      Assembly      |
|      SafeMath      |      13     |              |                    |      No      |                    |
|     LibContext     |      1      |              |                    |      No      |      Assembly      |
|     LibOwnable     |      1      |              |                    |      No      |                    |
|  LibBlacklistable  |      2      |              |                    |      No      |                    |
|     LibPausable    |      2      |              |                    |      No      |                    |
|       Address      |      11     |              |                    |      No      |      Send ETH      |
|                    |             |              |                    |              |    Delegatecall    |
|                    |             |              |                    |              |      Assembly      |
|   IERC1363Receiver |      1      |              |                    |      No      |                    |
|   IERC1363Spender  |      1      |              |                    |      No      |                    |
|      LibERC20      |      5      |              |                    |      No      |                    |
|     LibERC1363     |      7      |              |                    |      No      |                    |
|    LibECRecover    |      1      |              |                    |      No      |      Ecrecover     |
|     LibEIP712      |      2      |              |                    |      No      |      Assembly      |
| LibGasAbstraction  |      9      |              |                    |      No      |                    |
|      SafeERC20     |      6      |              |                    |      No      |      Send ETH      |
|                    |             |              |                    |              | Tokens interaction |
|       IBeacon      |      1      |              |                    |      No      |                    |
|    StorageSlot     |      4      |              |                    |      No      |      Assembly      |
|     HeartToken     |      93     | ERC20,ERC165 |      Pausable      |      No      |    Receive ETH     |
|                    |             |              |      - Minting     |              |      Send ETH      |
|                    |             |              | Approve Race Cond. |              |    Delegatecall    |
|                    |             |              |                    |              |     Upgradeable    |
+--------------------+-------------+--------------+--------------------+--------------+--------------------+
contracts/HeartToken.sol analyzed (33 contracts)
```

Concluding Remarks

While conducting the audits of the YellowHeart smart contract, it was observed that the contracts contain only Low severity issues.

Our auditors suggest that 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 YellowHeart 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.