Keep Safe Finance

Token

Smart Contract Audit Report

September 15, 2021

Introduction 3
About Keep Safe Finance 3
About ImmuneBytes 3
Documentation Details 3
Audit Process & Methodology 4
Audit Details 4
Audit Goals 5
Security Level References 5
High Severity Issues 6
Medium severity issues 6
Low Severity Issues 6
Recommendations 6
Automated Test Results 7
Concluding Remarks 8
Disclaimer 8

Introduction

1. About Keep Safe Finance

Keep safe finance is a trading platform that provides safety to the cryptocurrency environment and helps small traders acquire a soft loan that will boost their trading capital. By joining keep safe finance users will be eligible to have all features and will be guided into different major areas of cryptocurrency.

Keep safe finance token is a utility token of Keep safe finance exchange, a cryptocurrency built on Binance smart chain that is poised to replace banking with faster transactions, higher levels of security, and low fees.

Visit https://keepsafefinance.com/ to learn 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 75+ 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 Keep Safe team does not provide any documentation for the purpose of the audit.

Audit Process & Methodology

The 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: Keep Safe Finance
- Contracts Name: Keep Safe Finance
- Languages: Solidity(Smart contract)
- Smart Contract Address: 0xCAe905A3ab9304D681C73b32B6fCcBc69C9cEBb1
- 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
Closed	-	-	-

High Severity Issues

No issues were found.

Medium severity issues

No issues were found.

Low Severity Issues

1. **Absence of Zero-Address Validation found in the contract**

   Explanation:
   The CoinToken contract includes a constructor that accepts an address argument called **feeReciever_**.

   The "**msg.value**" amount passed while deploying the contract is transferred to this argument **feeReciever_** within the constructor body.

   However, during the manual code review of the contract, it was found that no **zero address** input validation has been performed on this argument before initiating the transfer of **BNB**.

   This might lead to an undesirable scenario where the **BNB** amount is transferred to an invalid address argument passed during the contract deployment.

```
326     ) payable ERC20(name_, symbol_,initialBalance_,decimals_,tokenOwner_) {
327         payable(feeReceiver_).transfer(msg.value);
328     }
329 }
```

   Recommendation:
   Adequate input validations must be implemented effectively in the contract to avoid the above-mentioned scenario.

Recommendations

1. **NatSpec Annotations must be included**

   Explanation:
   The smart contracts do not include the NatSpec annotations adequately.

   Recommendation:
   It is considered a better practice to cover your smart contracts with NatSpec annotations.

Automated Test Results

```
Compiled with solc
Number of lines: 329 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 5 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 11
Number of informational issues: 4
Number of low issues: 1
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC20

+-----------+-------------+-------+--------------------+--------------+-------------+
|    Name   | # functions |  ERCS |     ERC20 info     | Complex code |   Features  |
+-----------+-------------+-------+--------------------+--------------+-------------+
| CoinToken |      26     | ERC20 |     No Minting     |      No      | Receive ETH |
|           |             |       | Approve Race Cond. |              |   Send ETH  |
|           |             |       |                    |              |             |
+-----------+-------------+-------+--------------------+--------------+-------------+
INFO:Slither:contracts/CoinToken.sol analyzed (5 contracts)
```

Concluding Remarks

While conducting the audits of the Keep Safe Finance smart contract, it was observed that the contracts contained only Low severity issues. No High or Medium severity is found.

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 Keep Safe Finance 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