DeSpace

Token

Smart Contract Audit Report

September 10, 2021

IMMUNE BYTES

Introduction

1. About DeSpace

DeSpace Protocol is a new breath of the DeFi industry supporting cross-chain multi-layer DeFi & NFT protocols, tokens, NFT cards, combined Governance (DAO), redesigned Yield Farming, and NFT Mining. The goal of the project is to create a stable, intuitive, user-friendly, and secure platform for every user, as well as to create value for our NFT cards in our ecosystem. NFTs without any ecosystem can only serve as speculative assets on the secondary markets.

DeSpace Protocol creates an ecosystem that brings value to the NFT cards. Users can increase their farming income via our NFTs. And each NFT has different power, this means that every new card will add more interest to your farming. The protocol works on the principle of layers analogous to layers of the solar system. Each new layer will complement the preceding layer, simultaneously revealing unknown and latent elements in the prior layers, thereby complementing each other. Each layer can be considered as a separate universe and functions independently.

Visit https://despace.io/ 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 DeSpace team has provided the following doc for the purpose of audit:

1. https://despace.io/space-paper/

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: DeSpace
- Contracts Name: BridgeAssistB.sol, BridgeAssistE.sol, DeSpaceToken.sol, TokenB.sol
- Languages: Solidity(Smart contract)
- Github commit for initial audit: d15fea9f51ee42fc56ea3c21d9372b7e5ef58f2d
- Platforms and Tools: Remix IDE, Truffle, Ganache, Solhint, 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	-	-	5
Closed	-	-	-

Contract TokenB.sol

High Severity Issues

No issues were found.

Medium severity issues

No issues were found.

Low Severity Issues

1. **Inadequate Error Handling found in transfer Functions**
   Line no - 61-66, 68-74

   Explanation:
   The transfer functions in the contract do not include proper **require** statements to ensure that the token to be transferred is smaller than or equal to the balance of the sender.
   As per the current design of the function, if the sender's balance is lower than the amount to be transferred the transaction simply reverts because of subtraction overflow without providing an appropriate error message.

```
61    function transfer(address _to, uint256 _amount) public returns (bool success) {
62        balanceOf[msg.sender] -= _amount;
63        balanceOf[_to] += _amount;
64        emit Transfer(msg.sender, _to, _amount);
65        return true;
66    }
67
```

   While this affects the code-readability, it also fails to provide an adequate reason behind a function revert, to the user.

   Recommendation:
   Require statements can be added accordingly to enhance code readability.

2. **Absence of Zero Address Validation**
   Line no- 55, 61, 68, 82

   Explanation:
   The **Token** Contract includes quite a few functions that update some of the imperative addresses in the contract.

However, during the automated testing of the contact it was found that no Zero Address Validation is implemented on the following functions while updating the state variables of the contract:

- **approve()**
- **transfer()**
- **transferFrom()**
- **setIssuerRights()**

Recommendation:
A require statement should be included in such functions to ensure no zero address is passed in the arguments.

3. **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 contract:

- **getOwner()**
- **mint()**
- **burn()**
- **burnFrom()**
- **approve()**
- **transfer()**
- **transferFrom()**
- **transferOwnerShip()**
- **setIssuerRights()**

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

Recommendations

1. **NatSpec Annotations must be included**

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

Recommendation:
Cover by NatSpec all Contract methods.

Contract BridgeAssistB.sol

High Severity Issues

No issues were found.

Medium severity issues

No issues were found.

Low Severity Issues

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 contract:
   - **collect()**
   - **dispense()**
   - **transferOwnership()**

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

Recommendations

1. **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.

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

```
Parameter BridgeAssistB.collect(address,uint256)._sender (myFlats/FlatB_Bridge.sol#21) is not in mixedCase
Parameter BridgeAssistB.collect(address,uint256)._amount (myFlats/FlatB_Bridge.sol#21) is not in mixedCase
Parameter BridgeAssistB.dispense(address,uint256)._sender (myFlats/FlatB_Bridge.sol#28) is not in mixedCase
Parameter BridgeAssistB.dispense(address,uint256)._amount (myFlats/FlatB_Bridge.sol#28) is not in mixedCase
Parameter BridgeAssistB.transferOwnership(address)._newOwner (myFlats/FlatB_Bridge.sol#34) 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.

Contract BridgeAssistE.sol

High Severity Issues

No issues were found.

Medium severity issues

No issues were found.

Low Severity Issues

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 contract:
   - **collect()**
   - **dispense()**
   - **transferOwnership()**

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

Recommendations

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

   Explanation:
   Code readability of a Smart Contract is largely influenced by the Coding Style issues and its imperative to follow best the standard practice.

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

```
Parameter BridgeAssistE.collect(address,uint256)._sender (myFlats/FlatE_Bridge.sol#22) is not in mixedCase
Parameter BridgeAssistE.collect(address,uint256)._amount (myFlats/FlatE_Bridge.sol#22) is not in mixedCase
Parameter BridgeAssistE.dispense(address,uint256)._sender (myFlats/FlatE_Bridge.sol#28) is not in mixedCase
Parameter BridgeAssistE.dispense(address,uint256)._amount (myFlats/FlatE_Bridge.sol#28) is not in mixedCase
Parameter BridgeAssistE.transferOwnership(address)._newOwner (myFlats/FlatE_Bridge.sol#34) 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.

Contract DeSpaceToken.sol

High Severity Issues

No issues were found.

Medium severity issues

No issues were found.

Low Severity Issues

No issues were found.

Recommendations

--

Automated Test Results

1. TokenB.sol

```
Compiled with solc
Number of lines: 94 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 1 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 9
Number of informational issues: 17
Number of low issues: 0
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC20

+-------+-------------+-------+--------------------+--------------+----------+
|  Name | # functions |  ERCS |     ERC20 info     | Complex code | Features |
+-------+-------------+-------+--------------------+--------------+----------+
| Token |      10     | ERC20 |     ∞ Minting      |      No      |          |
|       |             |       | Approve Race Cond. |              |          |
|       |             |       |                    |              |          |
+-------+-------------+-------+--------------------+--------------+----------+
INFO:Slither:contracts/TokenB.sol analyzed (1 contracts)
```

2. BridgeAssistB.sol

```
Compiled with solc
Number of lines: 44 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 2 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 3
Number of informational issues: 8
Number of low issues: 2
Number of medium issues: 0
Number of high issues: 0


+---------------+-------------+------+------------+--------------+----------+
|      Name     | # functions | ERCS | ERC20 info | Complex code | Features |
+---------------+-------------+------+------------+--------------+----------+
|     IERC20    |      3      |      |            |      No      |          |
| BridgeAssistB |      4      |      |            |      No      |          |
+---------------+-------------+------+------------+--------------+----------+
INFO:Slither:contracts/BridgeAssistB.sol analyzed (2 contracts)
```

3. BridgeAssistE.sol

```
Compiled with solc
Number of lines: 42 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 2 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 3
Number of informational issues: 8
Number of low issues: 2
Number of medium issues: 0
Number of high issues: 0


+--------------+-------------+------+-----------+-------------+--------------------+
|     Name     | # functions | ERCS | ERC20 info | Complex code |      Features      |
+--------------+-------------+------+-----------+-------------+--------------------+
|    IERC20    |      2      |      |           |      No     |                    |
| BridgeAssistE |      4      |      |           |      No     | Tokens interaction |
+--------------+-------------+------+-----------+-------------+--------------------+
```

4. DeSpaceToken.sol

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

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


+------+-------------+------+-----------+-------------+----------+
| Name | # functions | ERCS | ERC20 info | Complex code | Features |
+------+-------------+------+-----------+-------------+----------+
+------+-------------+------+-----------+-------------+----------+
WARNING:Slither:No contract was analyzed
INFO:Slither:myFlats/Flat_DeSpaceToken.sol analyzed (0 contracts)
```

Concluding Remarks

While conducting the audits of the DeSpace smart contracts, it was observed that the contracts contained only Low severity issues with a few areas of recommendations. No High or Medium severity are found.

Our auditors suggest that Low severity issues and recommendations should be resolved by DeSpace 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 DeSpace 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