# DeSpace

**NFT**

# Smart Contract Audit Report


**September 12, 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: DeSpaceAuction.sol, DesLinkRegistry.sol, DeSpaceNFT.sol
- Languages: Solidity(Smart contract)
- Github commit for initial audit:
- 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 | 1 | 2 | 6 |
| Closed | - | - | - |

# Contract Folder: NFT_Auction_Contracts

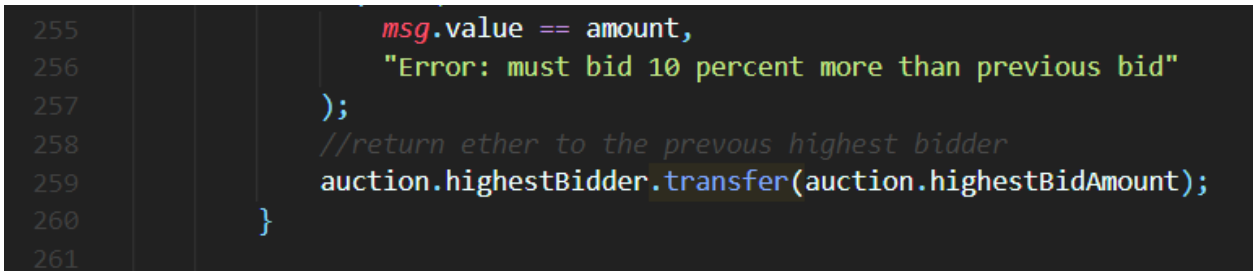
## Contract: DeSpaceAuction.sol

### High Severity Issues

1. **Sending Ether to External Entity with Transfer method could freeze the contract**
   **Line no - 259**

   **Explanation:**
   In **bidWithEther(uint)** method, an external call transferring ether to the highest bidder is being made. The function executes an external call before updating imperative state variables and this might lead to unwanted scenarios.?

   Since the `**transfer**` method used a non-adjustable 2300 amount of forwarded gas; the transaction could run out of gas, since there is no check that highestBidder is not an EOA (externally-owned-account) when being set in line 263, & contracts can be set as payable (see here) it could be a contract with a fallback function which could perform a reentrancy attack.
   Fallback functions requiring more gas than the stipend could otherwise freeze a contract.



```
255                 msg.value == amount,
256                 "Error: must bid 10 percent more than previous bid"
257             );
258             //return ether to the prevous highest bidder
259             auction.highestBidder.transfer(auction.highestBidAmount);
260         }
261
```

   **Recommendation:**
   Pull Over Push pattern can be used, which lets the users request payments instead of sending them, in order to avoid unexpected behavior. This can be done by storing the remaining user balances in a mapping (with the complaint token addresses) and adding another function for users to withdraw their funds.

## Medium severity issues

1. **Better error handling in case token is not complaint with registry**
   **Line no - 701**

   **Explanation:**
   In function **_nextBigAmountToken**, if this is the first bid, (i.e. current == 0) there is no check if the complaint token used is supported by the registry (i.e. its chainlink feed exists and is stored in the registry)
   This would result in `decimals` and `ethPerToken` being returned as **ZERO** and line 701 will trigger the INVALID_OPCODE (instead of revert since solidity 0.8 see here) because of **division by zero**.

```
696         if (current == 0) {
697             (,uint8 decimals) = registry.getProxy(_compliantToken);
698             uint ethPerToken = _getThePrice(_compliantToken);
699             return (
700             //get the equivilent based on chainlink oracle price and token decimal
701                 ((10 ** uint(decimals)) * initialBidAmount) / ethPerToken
702             );
```

   **Recommendation:**
   It is recommended to add a require statement after initializing ethPerToken on line 698, such as

   require(ethPerToken != 0, "Error: _complaintToken not supported");

   This ensures adequate error messages being forwarded to users.

2. **Return Value of an External Call is never used Effectively**
   **Line no - 325, 327, 345, 349**

   **Explanation:**
   The external calls made in the above-mentioned lines do return a boolean value that indicates whether or not the external call made was successful.

   These boolean return values can be used in the function as a check to ensure that the further execution of the function is only allowed if the external is successfully made.

   However, during the automated testing of the **Auction** contract, it was found that the protocol never uses these returns adequately.

The following functions avoid the return values of external calls:

- **closeBid()**
- **bidWithToken()**

**Recommendation:**
Effective use of all the return values from external calls must be ensured within the contract.

## Low Severity Issues

1. **Violation of Check_Effects_Interaction Pattern**
   **Line no - 267, 359, 429, 451,**

   **Explanation:**
   As per the Check_Effects_Interaction Pattern in Solidity, external calls should be made at the very end of the function and event emission, as well as any state variable modification, must be done before the external call is made.

   However, the following functions in the DeSpaceAuction contract emit events as well as change imperative state variables after the external call has been made at the line number mentioned above:

   - **bidWithEther()**
   - **bidWithToken()**
   - **closeBid()**

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

2. **No minimum or maximum threshold is validated for _newFee**
   **Line no - 747-752**

   **Explanation:**
   The **_setFeePercentage** function in the protocol includes a validation that ensures that the _newFee argument passed must not be equal to the already existing **feePercentage**.

   However, during the manual review, it was found that the function doesn't include any validation to check the maximum or minimum threshold of the **_newFee** argument being passed. For instance, the **_newFee** can never be as low as zero.

```
747     function _setFeePercentage(
748         uint _newFee
749         ) private {
750         require(_newFee != feePercentage, "Error: already set");
751         feePercentage = _newFee;
752     }
```

Keeping in mind the fact that the **_feePercentage** state variable plays a significant role during fee calculation while closing a bid, it's imperative to validate the arguments passed adequately.

**Recommendation:**
Adequate input validations must be included before modifying imperative state variables.

3. **Reuse function result instead of evaluating again**
   **Line no - 283, 375**

**Explanation:**
In both **bidWithEther** and **bidWithToken** methods, while increasing the countdown to set the auction end period, timeLeft is used to store the return value of **_bidTimeRemaining**(uint) and is not modified.

However, the **_bidTimeRemaining** function is being redundantly called again while emitting the **AuctionUpdated** event at the lines mentioned above.

```
274         //increase countdown clock
275         uint timeLeft = _bidTimeRemaining(_tokenId);
276         if(timeLeft < 1 hours) {
277             timeLeft + 10 minutes <= 1 hours
278             ? auction.endPeriod += 10 minutes
279             : auction.endPeriod += 1 hours - timeLeft;
280
281             emit AuctionUpdated(
282                 _tokenId,
283                 block.timestamp + _bidTimeRemaining(_tokenId)
284             );
285         }
286
```

**Recommendation:**
It is common practice to use local variables inside functions as they don't require any storage slots, and since the **`timeLeft`** variable is unchanged, it can be used at the above-mentioned lines to save function execution gas cost.

4. **Prefer using constant on variables that aren't being reassigned**
   **Line no - 32, 34**

   **Explanation:**
   In DeSpaceAuction, the following variables: initialBidAmount, and DIVISOR are being set once in the constructor and never being reassigned.

```
32      uint public initialBidAmount;
33      uint public feePercentage; // 1% = 1000
34      uint private DIVISOR;
```

   Recommendation:
   They can be assigned their values using the `constant` keyword to save storage slots for this specific contract unless intended by design.

```
uint public constant initialBidAmount = 1 ether;
uint private constant DIVISOR = 100 * 1000;
```

## Recommendations

1. **Order of layout**

   **Explanation:**
   As per the Solidity Style Guide, the order of elements and statements should be according to the following layout:
   a. Pragma statements
   b. Import statements
   c. Interfaces
   d. Libraries
   e. Contracts

   Inside each contract, library or interface, use the following order:
   a. Type declarations
   b. State variables
   c. Events
   d. Functions

   The following documentation links can be used as a reference to understand the correct order: -

   https://solidity.readthedocs.io/en/v0.8.7/style-guide.html#order-of-layout
   https://solidity.readthedocs.io/en/v0.8.7/style-guide.html#order-of-functions

# Contract DesLinkRegistry.sol

## High Severity Issues

No issues were found.

## Medium severity issues

No issues were found.

## Low Severity Issues

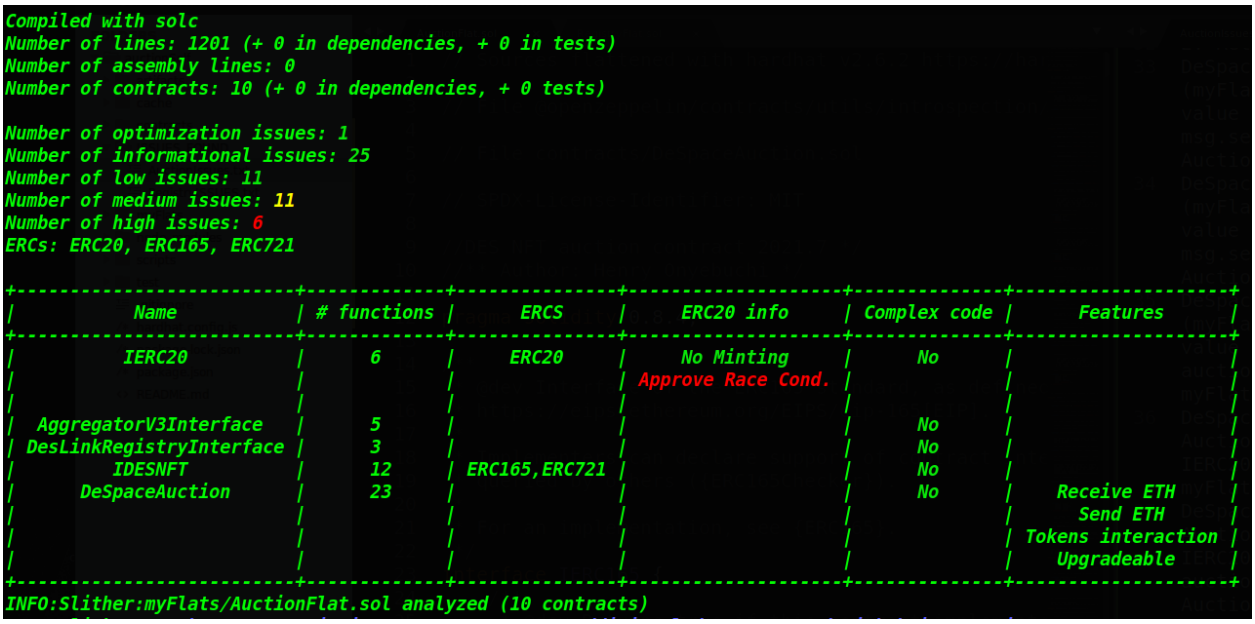
No issues were found.

## Recommendations

--

# Automated Test Results

1. **DeSpaceAuction.sol**

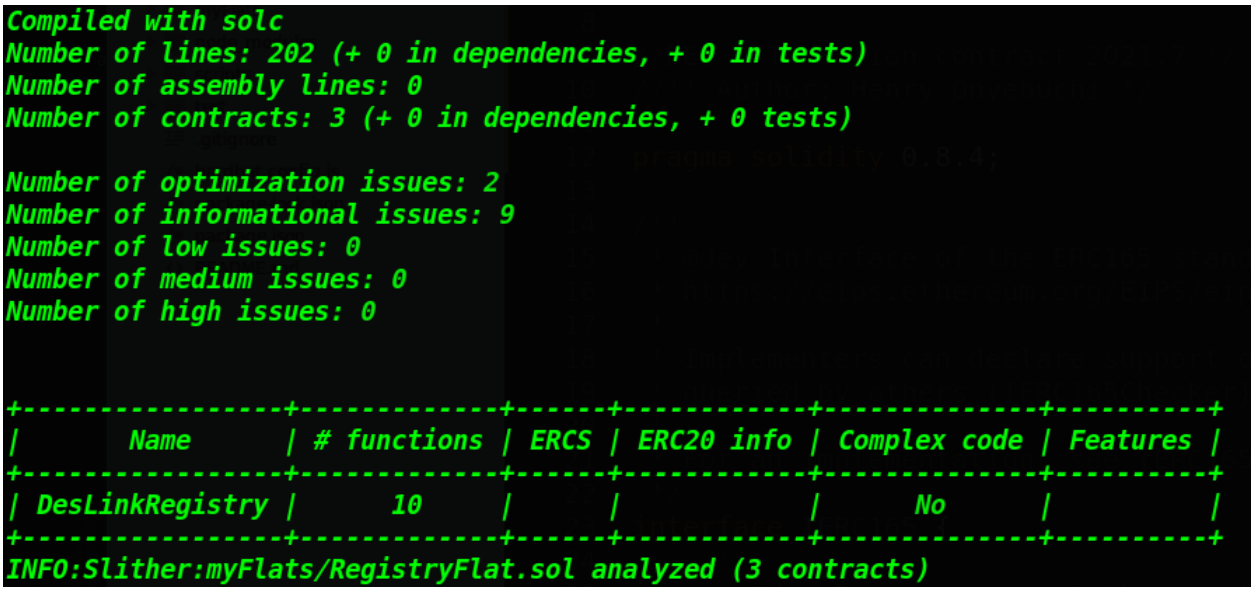


```
Compiled with solc
Number of lines: 1201 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 10 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 1
Number of informational issues: 25
Number of low issues: 11
Number of medium issues: 11
Number of high issues: 6
ERCs: ERC20, ERC165, ERC721

+-------------------------+------------+--------------+--------------------+--------------+--------------------+
|          Name           | # functions |     ERCS     |     ERC20 info     | Complex code |      Features      |
+-------------------------+------------+--------------+--------------------+--------------+--------------------+
|         IERC20          |     6      |    ERC20     |     No Minting     |      No      |                    |
|                         |            |              | Approve Race Cond. |              |                    |
|                         |            |              |                    |              |                    |
|  AggregatorV3Interface  |     5      |              |                    |      No      |                    |
| DesLinkRegistryInterface |     3      |              |                    |      No      |                    |
|         IDESNFT         |     12     | ERC165,ERC721 |                    |      No      |                    |
|      DeSpaceAuction     |     23     |              |                    |      No      |    Receive ETH     |
|                         |            |              |                    |              |      Send ETH      |
|                         |            |              |                    |              | Tokens interaction |
|                         |            |              |                    |              |    Upgradeable     |
+-------------------------+------------+--------------+--------------------+--------------+--------------------+
INFO:Slither:myFlats/AuctionFlat.sol analyzed (10 contracts)
```

2. **DesLinkRegistry.sol**



```
Compiled with solc
Number of lines: 202 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 3 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 2
Number of informational issues: 9
Number of low issues: 0
Number of medium issues: 0
Number of high issues: 0


+-----------------+-------------+------+------------+--------------+----------+
|       Name      | # functions | ERCS | ERC20 info | Complex code | Features |
+-----------------+-------------+------+------------+--------------+----------+
| DesLinkRegistry |      10     |      |            |      No      |          |
+-----------------+-------------+------+------------+--------------+----------+
INFO:Slither:myFlats/RegistryFlat.sol analyzed (3 contracts)
```

# Contract Folder: Des-NFT

## Contract DeSpaceNFT.sol

### High Severity Issues

No issues were found.

### Medium severity issues

No issues were found.

### Low Severity Issues

No issues were found.

### 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.

```
Variable ERC721BurnableUpgradeable.__gap (FlatNFT.sol#2109) is not in mixedCase
Parameter DeSpaceNFT.initialize(address)._defaultAdmin (FlatNFT.sol#2148) is not in mixedCase
Parameter DeSpaceNFT.createNFT(string,address)._tokenURI (FlatNFT.sol#2158) is not in mixedCase
Parameter DeSpaceNFT.addMarketplace(address)._marketplace (FlatNFT.sol#2193) is not in mixedCase
Parameter DeSpaceNFT.removeMarketplace(address)._marketplace (FlatNFT.sol#2220) is not in mixedCase
Parameter DeSpaceNFT.isMarketplace(address)._marketplace (FlatNFT.sol#2247) is not in mixedCase
```

   During the automated testing, it was found that the DeSpaceNFT 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.

2. **NatSpec Annotations must be included**

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

   **Recommendation:**
   Cover by NatSpec all Contract methods.

## Automated Test Results

```
Compiled with solc
Number of lines: 2377 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 21 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 11
Number of informational issues: 93
Number of low issues: 3
Number of medium issues: 5
Number of high issues: 6
ERCs: ERC721, ERC165
```

| Name | # functions | ERCS | ERC20 info | Complex code | Features |
|---|---|---|---|---|---|
| StringsUpgradeable | 4 | | | Yes | |
| EnumerableSetUpgradeable | 24 | | | No | Assembly |
| CountersUpgradeable | 4 | | | No | |
| IERC721ReceiverUpgradeable | 1 | | | No | |
| AddressUpgradeable | 9 | | | No | Send ETH, Assembly |
| DeSpaceNFT | 114 | ERC165,ERC721 | | No | Assembly, Upgradeable |

## Concluding Remarks

While conducting the audits of the DeSpace smart contracts, it was observed that the contracts contain High, Medium, and Low severity issues with a few areas of recommendations.

Our auditors suggest that High, Medium, and 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***