Niftify

NFT

Smart Contract Audit Report

September 02, 2021

IMMUNE BYTES

Introduction

1. About Niftify

Niftify is on a mission to create a strong community of NFT enthusiasts (content creators, traders, investors, etc.) wanting to buy, sell or trade NFTs, in a hassle-free environment.

There are already several NFT marketplaces, most of which need programming expertise to make successful use of them, which introduces more difficulty and the lack of accessibility. They are typically complex to use and difficult to understand and require in-depth trading knowledge and programming skills, which alienates new customers and proves to be time-consuming for the more experienced traders.
Users can only buy Non-fungible tokens (NFTs) with cryptocurrencies. While on a worldwide scale a high percentage of users do not have access to cryptocurrencies or simply do not want to use them, not to talk of using to purchase a simple NFT.
Another issue NFTs are facing is the fact that only traditional media types such as art, music, videos as well as gaming assets are on the market.

Visit https://niftify.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 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 Niftify team has provided the following doc for the purpose of audit:

1. https://niftify.io/lightpaper
2. https://docs.google.com/document/d/1s6r7jG7lWeOIctJnCSBneg1ZC0xH2AtcyYwjQnAEl4o/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: Niftify
- Contracts Name: NiftifyERC1155.sol, NiftifyERC721.sol, TokenStorage.sol, NiftifyNFTVoucher.sol
- Languages: Solidity(Smart contract), Javascript(Unit Testing)
- Github commit for initial audit: 19d63f7369e3b3c61a860ea147ac7bfb8c689ad7
- 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	-	2	2
Closed	-	-	-

Contract: TokenStorage

High Severity Issues

No issues were found.

Medium severity issues

1. **Multiplication is being performed on the result of Division**
   Line no - 83-94

   Explanation:
   During the automated testing of the **TokenStorage** contract, it was found that the contract includes a function that performs multiplication on the result of a Division.

   Integer Divisions in Solidity might truncate. Moreover, this performing division before multiplication might lead to a loss of precision.

   The following function involves division before multiplication in the mentioned lines:
   - **mulScale** at 83-94

   The automated Test result has been attached below:

```
TokenStorage.mulScale(uint256,uint256,uint32) (myFlats/flatTokenStorage.sol#532-543) performs a multiplication on the result of a division:
        -c = y / scale (myFlats/flatTokenStorage.sol#539)
        -a * c * scale + a * d + b * c + (b * d) / scale (myFlats/flatTokenStorage.sol#542)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
```

   Recommendation:
   Solidity doesn't encourage arithmetic operations that involve division before multiplication.
   Therefore it is highly recommended to write extensive test cases to ensure the function behavior is as per the expectation. The above-mentioned functions should be redesigned only if they do not lead to expected results.

2. **Operator Role has not been set up in the contract**
   Line no - 8

   Explanation:
   The **OPERATOR_ROLE** in the TokenStorage contract has not been assigned to any particular address within the contract.

Is this Intended?
While the **Minter** and **Redeemer** roles can be handled later via admin, the operator role might need an initial setup as functions like **setBlocked** uses Operator_Roles as a requirement, in the contract.

```
66
67      function setBlocked(uint256 _tokenId, bool _value) external onlyOperator {
68        _blocked[_tokenId] = _value;
69      }
70
```

Recommendation:
If the above-mentioned scenario is not intended, then the Operator Role should be assigned to a particular address during deployment.

Low Severity Issues

1. **Absence of Error messages in Require Statements**
   Line no - 47

   Explanation:
   The **TokenStorage** contract includes a **require statement** in the **_setRoyalty** function(at the above-mentioned line) that doesn't contain any error message within itself.
   While this makes it troublesome to detect the reason behind a particular function revert, it also reduces the readability of the code.

   Recommendation:
   Error Messages must be included in every require statement in the contract.

2. **External Visibility should be preferred**

   Explanation:
   Those functions that are never called throughout the contract should be marked as **external** instead of **public** visibility.
   This will effectively result in Gas Optimization as well.

   The following function in the TokenStorage contract has been assigned **public visibility** but never called form within the contract:
   - **metadataHash() #Line 22**

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 **TokenStorage** contract had quite a few code-style issues.

```
INFO:Detectors:
Parameter TokenStorage.metadataHash(uint256)._tokenId (myFlats/flatTokenStorage.sol#471) is not in mixedCase
Parameter TokenStorage.creator(uint256)._tokenId (myFlats/flatTokenStorage.sol#483) is not in mixedCase
Parameter TokenStorage.royalty(uint256)._tokenId (myFlats/flatTokenStorage.sol#491) is not in mixedCase
Parameter TokenStorage.royaltyInfo(uint256,uint256)._tokenId (myFlats/flatTokenStorage.sol#500) is not in mixedCase
Parameter TokenStorage.royaltyInfo(uint256,uint256)._salePrice (myFlats/flatTokenStorage.sol#500) is not in mixedCase
Parameter TokenStorage.blocked(uint256)._tokenId (myFlats/flatTokenStorage.sol#512) is not in mixedCase
Parameter TokenStorage.setBlocked(uint256,bool)._tokenId (myFlats/flatTokenStorage.sol#516) is not in mixedCase
Parameter TokenStorage.setBlocked(uint256,bool)._value (myFlats/flatTokenStorage.sol#516) is not in mixedCase
Parameter TokenStorage.supportsInterface(bytes4)._interfaceId (myFlats/flatTokenStorage.sol#520) 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.

2. **NatSpec Annotations must be included**

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

   Recommendation:
   Cover by NatSpec all Contract methods.

Contract NiftifyERC721

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.

   During the automated testing, it was found that the **NiftifyERC721** contract had quite a few code-style issues.

```
Parameter NiftifyERC721.redeem(NiftifyNFTVouchers.ERC721Voucher)._voucher (myFlats/flaterc721.sol#2027) is not in mixedCase
Parameter NiftifyERC721.mint(NiftifyNFTVouchers.ERC721Voucher)._voucher (myFlats/flaterc721.sol#2044) is not in mixedCase
Parameter NiftifyERC721.mintAndTransfer(NiftifyNFTVouchers.ERC721Voucher,address)._voucher (myFlats/flaterc721.sol#2058) is not in mixedCase
Parameter NiftifyERC721.mintAndTransfer(NiftifyNFTVouchers.ERC721Voucher,address)._receiver (myFlats/flaterc721.sol#2058) is not in mixedCase
Parameter NiftifyERC721.updateMetadataHash(NiftifyNFTVouchers.UpdateMetadataHashVoucher)._voucher (myFlats/flaterc721.sol#2066) is not in mixedCase
Parameter NiftifyERC721.supportsInterface(bytes4)._interfaceId (myFlats/flaterc721.sol#2098) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

   Recommendation:
   Therefore, it is quite effective 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.

Contract NiftifyERC1155

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.

   During the automated testing, it was found that the **NiftifyERC1155** contract had quite a few code-style issues.

```
Parameter NiftifyERC1155.redeem(NiftifyNFTVouchers.ERC1155Voucher)._voucher (myFlats/flaterc1155.sol#2131) is not in mixedCase
Parameter NiftifyERC1155.mint(NiftifyNFTVouchers.ERC1155Voucher)._voucher (myFlats/flaterc1155.sol#2157) is not in mixedCase
Parameter NiftifyERC1155.mintAndTransfer(NiftifyNFTVouchers.ERC1155Voucher,address)._voucher (myFlats/flaterc1155.sol#2174) is not in mixedCase
Parameter NiftifyERC1155.mintAndTransfer(NiftifyNFTVouchers.ERC1155Voucher,address)._receiver (myFlats/flaterc1155.sol#2174) is not in mixedCase
Parameter NiftifyERC1155.updateMetadataHash(NiftifyNFTVouchers.UpdateMetadataHashVoucher)._voucher (myFlats/flaterc1155.sol#2188) is not in mixedCase
Parameter NiftifyERC1155.supportsInterface(bytes4)._interfaceId (myFlats/flaterc1155.sol#2225) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

   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.

Contract NiftifyNFTVouchers

High Severity Issues

No issues were found.

Medium severity issues

No issues were found.

Low Severity Issues

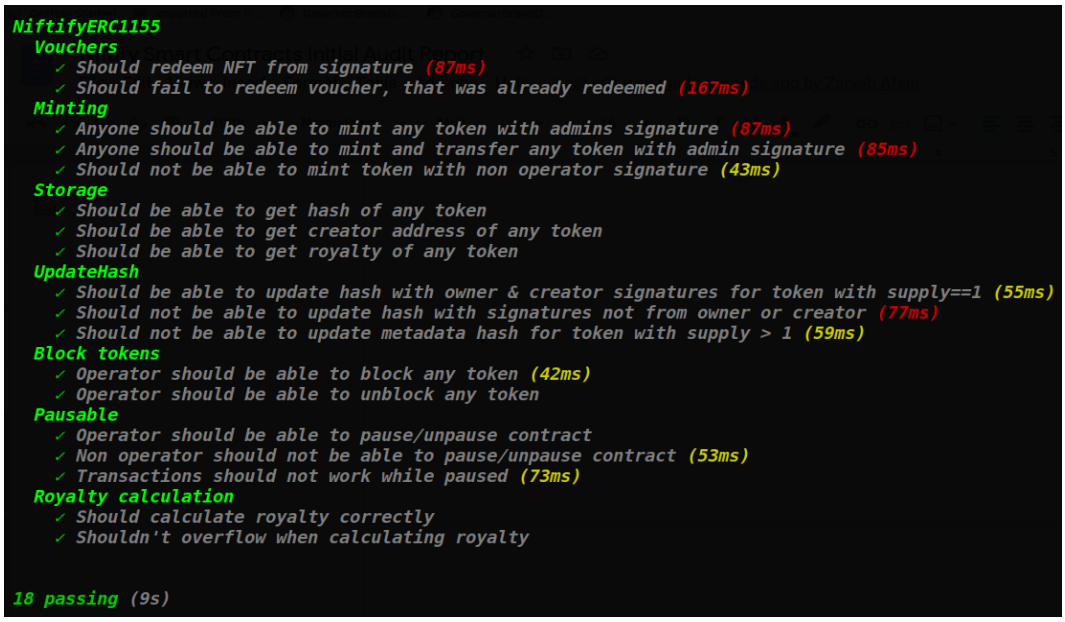
No issues were found.

Recommendations

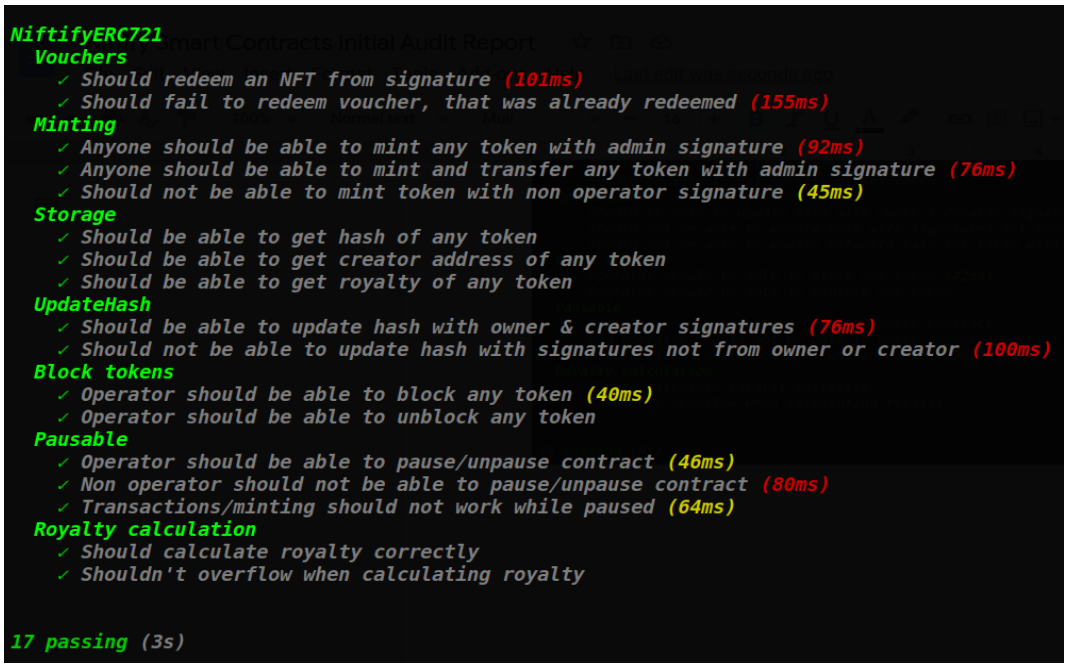
--

Unit Test

1. ERC1155



2. ERC721



Automated Test Results

1. TokenStorage

```
Compiled with solc
Number of lines: 568 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 8 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 4
Number of informational issues: 19
Number of low issues: 0
Number of medium issues: 1
Number of high issues: 0

ERCs: ERC165

+--------------+-------------+--------+------------+--------------+----------+
|     Name     | # functions |  ERCS  | ERC20 info | Complex code | Features |
+--------------+-------------+--------+------------+--------------+----------+
|   Strings    |      4      |        |            |     Yes      |          |
| TokenStorage |      34     | ERC165 |            |      No      |          |
+--------------+-------------+--------+------------+--------------+----------+
INFO:Slither:myFlats/flatTokenStorage.sol analyzed (8 contracts)
```

2. NiftifyNFTVouchers

```
Compiled with solc
Number of lines: 445 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 3 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 0
Number of informational issues: 23
Number of low issues: 1
Number of medium issues: 0
Number of high issues: 0


+--------------------+-------------+------+------------+--------------+-----------+
|        Name        | # functions | ERCS | ERC20 info | Complex code |  Features |
+--------------------+-------------+------+------------+--------------+-----------+
|       ECDSA        |      9      |      |            |      No      | Ecrecover |
|                    |             |      |            |              |  Assembly |
| NiftifyNFTVouchers |      9      |      |            |      No      |           |
+--------------------+-------------+------+------------+--------------+-----------+
INFO:Slither:myFlats/flatVouchers.sol analyzed (3 contracts)
```

3. NiftifyERC1155

```
Compiled with solc
Number of lines: 2236 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 21 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 9
Number of informational issues: 59
Number of low issues: 11
Number of medium issues: 9
Number of high issues: 0

ERCs: ERC165

+-------------------+-------------+--------+------------+--------------+--------------+
|       Name        | # functions | ERCS   | ERC20 info | Complex code |   Features   |
+-------------------+-------------+--------+------------+--------------+--------------+
| IERC1155Receiver  |      3      | ERC165 |            |      No      |              |
|     Address       |      11     |        |            |      No      |   Send ETH   |
|                   |             |        |            |              | Delegatecall |
|                   |             |        |            |              |   Assembly   |
|      ECDSA        |      9      |        |            |      No      |  Ecrecover   |
|                   |             |        |            |              |   Assembly   |
|     Strings       |      4      |        |            |     Yes      |              |
|  NiftifyERC1155   |      93     | ERC165 |            |      No      |  Ecrecover   |
+-------------------+-------------+--------+------------+--------------+--------------+
INFO:Slither:myFlats/flaterc1155.sol analyzed (21 contracts)
```

4. NiftifyERC721

```
Compiled with solc
Number of lines: 2109 (+ 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: 56
Number of low issues: 4
Number of medium issues: 2
Number of high issues: 0

ERCs: ERC165, ERC721

+-----------------+-------------+---------------+------------+--------------+--------------+
|       Name      | # functions |      ERCS     | ERC20 info | Complex code |   Features   |
+-----------------+-------------+---------------+------------+--------------+--------------+
| IERC721Receiver |      1      |               |            |      No      |              |
|     Address     |      11     |               |            |      No      |   Send ETH   |
|                 |             |               |            |              | Delegatecall |
|                 |             |               |            |              |   Assembly   |
|     Strings     |      4      |               |            |     Yes      |              |
|      ECDSA      |      9      |               |            |      No      |  Ecrecover   |
|                 |             |               |            |              |   Assembly   |
|  NiftifyERC721  |      99     | ERC165,ERC721 |            |      No      |  Ecrecover   |
|                 |             |               |            |              |   Assembly   |
+-----------------+-------------+---------------+------------+--------------+--------------+
INFO:Slither:myFlats/flaterc721.sol analyzed (21 contracts)
```

Concluding Remarks

While conducting the audits of the Niftify smart contract, it was observed that the contract contained Medium and Low severity issues along with a few areas of recommendations.

Our auditors suggest that Medium and Low severity issues should be resolved by Niftify 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 Niftify 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