NFY Finance

Trading Platform

Smart Contract Audit Report

IMMUNE BYTES

Audits

December 19, 2020

IMMUNE BYTES

IMMUNE BYTES

Introduction

1. About NFY Finance

Non-Fungible Yearn is a DeFi platform whose goal is to utilize the full potential of Non-Fungible Tokens (NFTs) in the DeFi sector. As of now, DeFi is dominated by ERC-20 tokens, which are fungible tokens, meaning that all are the same and are not unique from one another. While ERC-20 tokens are surely needed and are not going to be going away anytime soon, there are many other different token standards that are rarely mentioned, ERC-721 being one of them. ERC-721 tokens are non-fungible tokens, meaning that each token is unique and no two are like, this is because of the unique token id that each token is given at the time of being minted. Currently, ERC-721 tokens are mostly used as collectibles, the Non-Fungible Yearn platform will re-imagine how these tokens have been used by creating a use case for them in DeFi.

Visit https://nfy.finance/ to know more.

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 15+ 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 NFY Finance team has provided write-up on telegram for the purpose of conducting the audit.

Audit Process & Methodology

ImmueBytes 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: NFY Finance
- Languages: Solidity(Smart contract), Javascript(Unit Testing)
- Github commit for audit: https://github.com/NFYFinance/NFY-Trading-Platform/commit/a5c653e59aca07751c1b7891b7564cf3c4676511

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	4
Closed	-	-	-

High severity issues

No Issues found

Medium severity issues

1. **Non handling of zero address can result in funds getting locked.**
   In **NFYTradingPlatform.sol**, there is no check for **zero (0x00)** address for **devAddress** and **communityFund** address variables in **setDevFeeAddress**() and **setCommunityFeeAddress**() functions respectively. These variables are used in **createLimitOrder**() for transfer of NFY Token. If any of these addresses are set to zero (**0x00**) then the user's transactions to create new orders will always revert. This can result in ambiguous contract state.

   Recommendation:
   Consider either adding a **require** statement in the mentioned functions that restricts setting of addresses to zero address or handle the zero address scenario in **createLimitOrder**() function explicitly.

2. **Same token can be added multiple times via addToken() function.**
   In **NFYTradingPlatform**.sol, the **addToken**() function is used by **admin** to register new tokens with the trading contract. The function lacks the check for duplicacy of a token, i.e. when the same token is added twice. This registration of a single token multiple times can result in inconsistent contract state.

   Recommendation:
   Consider adding a require statement which verifies that the token being added does not exist already.

Low severity issues

1. **Precision error during fee calculation.**
   In **createLimitOrder**() function of **NFYTradingPlatform.sol** precision error can occur at **line 200** and **201**. In these statements **division** is performed before **multiplication** which is not the best way for arithmetic calculations in Solidity. As Solidity only deals with integers (not decimal numbers), it is always recommended to perform **multiplications** before **division** to reduce the chances of precision loss.

2. **Unused variable - ETH**
   In **NFYTradingPlatform.sol** at **line 17** a **bytes32 constant** variable named as **ETH** is declared but is never used inside the **NFYTradingPlatform** contract. However there is a test case present in **nfyTradingPlatform.js (line 749)** which asserts the reverting of transaction when a user creates limit order with ETH ticker. This test case passes because ETH ticker wasn't added by admin in the **NFYTradingPlatform** contract, there is no restriction present for ETH ticker in the contract itself.
   Also the **nftAddress** variable (**line 38**) in **StakeToken** struct is also never used in the contract.

   Recommendation:
   Consider removing the unused variables and handle the ETH ticker explicitly in the NFYTradingPlatform contract.

3. **Redundant balance check in createLimitOrder() function.**
   At **line 198** there is a **require** statement which checks the NFY token balance of the caller to be greater than **platformFee**. This check seems redundant as the **transferFrom**() function of **ERC20** implicitly checks the sufficient balance condition before any token transfer. Every statement in Solidity smart contract increases the total gas cost of transactions. Removing unnecessary conditions can save gas.

4. **Unchecked return value of ERC20 functions.**
   In **NFYTradingPlatform** smart contract there are multiple instances (**line 205, 206 and 207**) where the return value of **transferFrom**() functions of ERC20 token is not checked. It is always suggested to explicitly check the return value of ERC20 functions.

   Recommendation:
   Consider wrapping the ERC20 function calls in a **require** statement.

Admin Privileges

The **admin/owner** of **NFYTradingPlatform** has some privileges over the smart contract. These privileges can be misused either intentionally or unintentionally (in case the admin's private key gets hacked). We assume that these extra rights will always be used appropriately.
Some of these admin rights are listed below.

1. The **setFee**() function in the **NFYTradingPlatform** contract can be called any number of times. Also, any **uint** value can be set as **platformFee**.

2. The **setCommunityFeeAddress**() in **NFYTradingPlatform** contract can be called any number of times by which any **address** can be set as **communityFund**.

Recommendation:
Consider hardcoding some predefined ranges/restrictions for variables. Also, consider adding some governance for admin rights for smart contracts or at least use a multi-sig wallet as admin/owner address.

Notes

1. **No mechanism to update rewardPool address.**
   The **rewardPool** address in **NFYTradingPlatform** contract is set at the time of contract deployment which collects fees on every new limit order. Currently there is no mechanism present in the contract to update this address's value.

   The adding or not adding of the function is the choice of the NFY Finance team.

Unit Test

All unit tests provided by the NFY Finance team are passing without any issues.

```
    ✓ should add sell order (668ms)
    ✓ should add buy order (389ms)
    ✓ should get sell order details properly (602ms)
    ✓ should get buy order details properly (466ms)
    ✓ should arrange sell orders properly (2644ms)
    ✓ should arrange buy orders properly (1970ms)
  # getTokens()
    ✓ should start with 0 tokens (82ms)
    ✓ length should be 1 token after an add (142ms)
    ✓ length should be 2 tokens after 2 adds (269ms)
  # createLimitOrder()
    ✓ should revert if stakeNFT does not exist (182ms)
    ✓ should revert if user does not have stake deposited (240ms)
    ✓ should revert if user does not have enough NFY to cover fee (364ms)
    ✓ should update reward pool with fee when taken out (564ms)
    ✓ should update dev fund with portion of fee (591ms)
    ✓ should update community fund with portion of fee (561ms)
    ✓ should let a user create a sell order if conditions are met (802ms)
    ✓ should let a user create a buy order if conditions are met (502ms)
    ✓ should NOT let a user create an order with ticker ETH (458ms)
    ✓ should NOT let a user create a sell order and does not have enough (676ms)
    ✓ should add order to order book after a sell order has been created (488ms)
    ✓ should add order to order book after a buy order has been created (476ms)
    ✓ should have added proper information to orders when sell order has been created (575ms)
    ✓ should update filled if some of sell order is filled (898ms)
    ✓ should update filled if some of buy order is filled (1046ms)
    ✓ should create a new order if sell limit order is filled (1084ms)
    ✓ should create new limit order if nothing on the side of market order (1127ms)
    ✓ should NOT let a user create a sell order if more than they have deposited (495ms)
    ✓ should NOT let a user create a buy order if more eth than they have deposited (308ms)
    ✓ should properly fill limit order and take proper eth from user (1065ms)
    ✓ should update the eth balance of a seller when their sell order gets filled (1163ms)
    ✓ should properly fill the rest of a users order after the initial amount is filled (1484ms)
  # cancelOrder()
    ✓ should revert if stakeNFT does not exist (85ms)
    ✓ should allow user to cancel their sell order that is not filled at all (593ms)
    ✓ should allow user to cancel their buy order that is not filled at all (477ms)
    ✓ should NOT let a user cancel sell order if they have no sell orders (405ms)
    ✓ should NOT let a user cancel buy order if they have no buy orders (391ms)
    ✓ should update balance after unfilled sell order is cancelled (788ms)
    ✓ should update balance after unfilled buy order is cancelled (652ms)
    ✓ should remove sell order from array once cancelled (870ms)
    ✓ should remove buy order from array once cancelled (610ms)
    ✓ should cancel a partly filled sell order (918ms)
    ✓ should cancel a partly filled buy order (953ms)
    ✓ should cancel a partly filled sell order and update balances properly (1256ms)
    ✓ should cancel a partly filled buy order and update balances properly (1304ms)
    ✓ should cancel the last unfilled sell order if a user has multiple sell orders (1111ms)
    ✓ should cancel the last unfilled buy order if a user has multiple buy orders (978ms)
    ✓ should cancel the last partly filled sell order if a user has multiple sell orders (1532ms)
    ✓ should cancel the last partly filled buy order if a user has multiple buy orders (1946ms)
    ✓ should update properly when user cancels sell order before another user's sell orders which was partly filled was cancelled (1902ms)
    ✓ should update properly when user cancels sell after after another sell order which was partly filled was cancelled (1895ms)


  99 passing (12m)
```

Coverage Report

Test coverage of smart contracts is not 100%.

```
  99 passing (16m)

-------------------------|----------|----------|----------|----------|----------------|
File                     |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
-------------------------|----------|----------|----------|----------|----------------|
 contracts/              |    93.16 |    85.53 |     96.3 |    93.37 |                |
  NFYTradingPlatform.sol |    93.41 |    84.72 |      100 |    93.55 |... 376,377,458 |
  Ownable.sol            |     87.5 |      100 |       80 |       90 |             33 |
 contracts/mocks/        |    55.15 |    32.58 |     62.9 |    54.97 |                |
  LPStaking.sol          |    29.84 |    16.07 |    31.58 |    29.84 |... 309,310,311 |
  LPStakingNFT.sol       |    61.54 |       50 |       60 |    58.33 | 33,34,36,46,56 |
  Migrations.sol         |        0 |        0 |        0 |        0 |        9,13,17 |
  NFYStaking.sol         |    70.37 |     43.1 |       70 |    70.37 |... 284,285,286 |
  NFYStakingNFT.sol      |    92.31 |    66.67 |      100 |    91.67 |             46 |
  Ownable.sol            |      100 |       50 |      100 |      100 |                |
  RewardPool.sol         |      100 |      100 |      100 |      100 |                |
  demoLP.sol             |      100 |      100 |      100 |      100 |                |
  demoToken.sol          |      100 |      100 |      100 |      100 |                |
-------------------------|----------|----------|----------|----------|----------------|
All files                |    69.86 |    51.92 |    73.03 |    70.08 |                |
-------------------------|----------|----------|----------|----------|----------------|

> Istanbul reports written to ./coverage/ and ./coverage.json
> solidity-coverage cleaning up, shutting down ganache server
```

Recommendation:
We recommend 100% line and branch coverage for unit test cases.

Slither Tool Result

```
❌ NFYTradingPlatform.ETH (NFYTradingPlatform.sol:17) is never used in NFYTradingPlatform
(NFYTradingPlatform.sol#14-465)
```

Concluding Remarks

While conducting the audits of NFY Finance smart contract, it was observed that the contracts contain 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 NFY Finance developers. Resolving the areas of recommendations are up to NFY Finance's discretion. 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 NFY Finance platform or its product neither this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the one audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes Pvt Ltd.