NFY Finance

Staking Platform - LPStaking Contract

Smart Contract Audit Report

IMMUNE BYTES

Audits

November 6, 2020

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 documentation for the purpose of conducting the audit.
Document link:

1. https://docs.google.com/document/d/1Js46U4A6zlHbQ0mwTRDcghhzfBsbEhRdtygiMpCmE_Y/edit

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-Staking-Platform/commit/7a8cc8994dc23442c3cd874f4e3dd8ccca93a844

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. **No mechanism to remove platform addresses in Ownable.sol.**
   In **Ownable** smart contract there is a mapping **platformAddress** which stores **truthy** value for NFY's internal system contracts which interacts with **LPStaking** contract. While there is an admin protected function to add a contract address in this mapping, there is no function to remove addresses from this mapping which can be an issue if any NFY's system contract gets redeployed/upgraded. In this case the old contract can still update the state of the LPStaking contract by calling any **onlyPlatform** protected function.

   Recommendation:
   Consider adding a function to remove an address from the **platformAddress** mapping.

2. **Admin rights.**
   The **LPStaking** contract has various admin protected functions which can be misused intentionally or unintentionally (in case admin's private key gets lost). Admin has the right to set **dailyReward** variable to any value which can range from **0** to **2**256 - 1**. Admin also has the right to add any address in **platformAddress** mapping, that address could be a smart contract or a EOA (user's wallet).

   Recommendation:
   Consider hardcoding a predefined range for **dailyReward** variable. Also there should be a check in **addPlatformAddress**() functions of **Ownable.sol** which verifies that the input address is not an EOA.
   Consider adding some governance for admin rights for **LPStaking** smart contract or atleast use a multi-sig wallet as the admin address.

Low severity issues

1. **Unchecked return value of ERC20 functions.**
   In **LPStaking** smart contract there are multiple instances (Line 137, 163, 168, etc) where the return value of **transfer**() and **transferFrom**() functions of **ERC20 tokens** 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.

2. **Precision loss errors.**
   The **getRewardPerBlock**() function of **LPStaking** smart contract performs a **division** before **multiplication** which may result in minute precision loss. To avoid the condition it is always recommended to perform **multiplication** before **division** which reduces the chances of precision errors.

   Recommendation:
   Consider performing multiplication before division in the mentioned function.

3. **Functions should be declared as *external.***
   Some of the functions in **LPStaking** smart contract are declared as **public** but are never accessed from within the contract and are only meant to be called **externally**. These functions include **getTotalBalance**(), **stakeLP**(), **claimAllRewards**(), **unstakeAll**() and **turnEmergencyWithdrawOn**(). These functions should be declared as **external** which is more gas optimal.

4. **No need to assign zero value to uint variables.**
   At **Line 60** of **LPStaking** the **uint** variable **accNfyPerShare** is assigned with a **zero(0)** value. In Solidity the default value for **uint** variable is **zero,** so there is no need for this statement.

Notes

1. Misleading comments
   The developer's comment above **unstakeLP**() function is misleading. It mentions a 5% unstaking fee from the staked amount of the users which is not the implemented as well as the intended business logic.

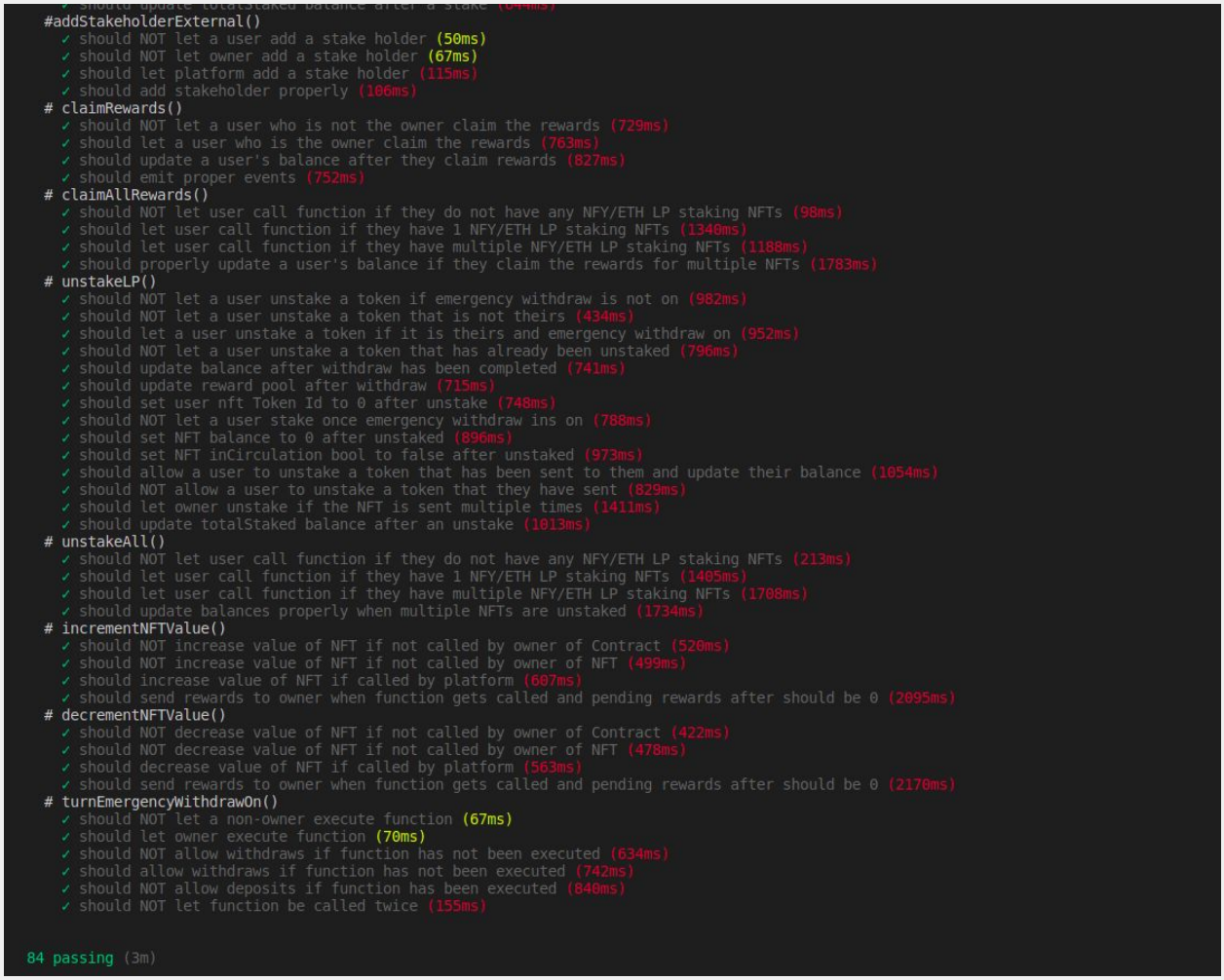
   Recommendation:
   Consider correcting the comment to avoid confusion.

   Amended: Comment has been removed by the NFY team and is no longer present.

2. No SPDX License identifier for Ownable.sol.
   As per Solidity documentation it is suggested to add a machine-readable SPDX license identifiers at the start of every source file. More details can be found here - https://solidity.readthedocs.io/en/v0.6.12/layout-of-source-files.html#spdx-license-identifier

Unit Test

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



```
  #addStakeholderExternal()
    ✓ should NOT let a user add a stake holder (50ms)
    ✓ should NOT let owner add a stake holder (67ms)
    ✓ should let platform add a stake holder (115ms)
    ✓ should add stakeholder properly (106ms)
  # claimRewards()
    ✓ should NOT let a user who is not the owner claim the rewards (729ms)
    ✓ should let a user who is the owner claim the rewards (763ms)
    ✓ should update a user's balance after they claim rewards (827ms)
    ✓ should emit proper events (752ms)
  # claimAllRewards()
    ✓ should NOT let user call function if they do not have any NFY/ETH LP staking NFTs (98ms)
    ✓ should let user call function if they have 1 NFY/ETH LP staking NFTs (1340ms)
    ✓ should let user call function if they have multiple NFY/ETH LP staking NFTs (1188ms)
    ✓ should properly update a user's balance if they claim the rewards for multiple NFTs (1783ms)
  # unstakeLP()
    ✓ should NOT let a user unstake a token if emergency withdraw is not on (982ms)
    ✓ should NOT let a user unstake a token that is not theirs (434ms)
    ✓ should let a user unstake a token if it is theirs and emergency withdraw on (952ms)
    ✓ should NOT let a user unstake a token that has already been unstaked (796ms)
    ✓ should update balance after withdraw has been completed (741ms)
    ✓ should update reward pool after withdraw (715ms)
    ✓ should set user nft Token Id to 0 after unstake (748ms)
    ✓ should NOT let a user stake once emergency withdraw ins on (788ms)
    ✓ should set NFT balance to 0 after unstaked (896ms)
    ✓ should set NFT inCirculation bool to false after unstaked (973ms)
    ✓ should allow a user to unstake a token that has been sent to them and update their balance (1054ms)
    ✓ should NOT allow a user to unstake a token that they have sent (829ms)
    ✓ should let owner unstake if the NFT is sent multiple times (1411ms)
    ✓ should update totalStaked balance after an unstake (1013ms)
  # unstakeAll()
    ✓ should NOT let user call function if they do not have any NFY/ETH LP staking NFTs (213ms)
    ✓ should let user call function if they have 1 NFY/ETH LP staking NFTs (1405ms)
    ✓ should let user call function if they have multiple NFY/ETH LP staking NFTs (1708ms)
    ✓ should update balances properly when multiple NFTs are unstaked (1734ms)
  # incrementNFTValue()
    ✓ should NOT increase value of NFT if not called by owner of Contract (520ms)
    ✓ should NOT increase value of NFT if not called by owner of NFT (499ms)
    ✓ should increase value of NFT if called by platform (607ms)
    ✓ should send rewards to owner when function gets called and pending rewards after should be 0 (2095ms)
  # decrementNFTValue()
    ✓ should NOT decrease value of NFT if not called by owner of Contract (422ms)
    ✓ should NOT decrease value of NFT if not called by owner of NFT (478ms)
    ✓ should decrease value of NFT if called by platform (563ms)
    ✓ should send rewards to owner when function gets called and pending rewards after should be 0 (2170ms)
  # turnEmergencyWithdrawOn()
    ✓ should NOT let a non-owner execute function (67ms)
    ✓ should let owner execute function (70ms)
    ✓ should NOT allow withdraws if function has not been executed (634ms)
    ✓ should allow withdraws if function has not been executed (742ms)
    ✓ should NOT allow deposits if function has been executed (840ms)
    ✓ should NOT let function be called twice (155ms)


  84 passing (3m)
```

Coverage Report

Test coverage of NFY LPStaking smart contract is 100% except the branch coverage which is 75%.

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

Slither Tool Result

```
LPStaking.claimRewards(uint256) (LPStaking.sol#192-208) compares to a boolean constant:
        -require(bool,string)(NFTDetails[_tokenId]._inCirculation == true,Stake has already been withdrawn)
(LPStaking.sol#194)
```

```
LPStaking.claimRewards(uint256) (LPStaking.sol#192-208) ignores return value by
NFYToken.transfer(_msgSender(),_pendingRewards) (LPStaking.sol#203)
```

```
stakeLP(uint256) should be declared external:
        - LPStaking.stakeLP(uint256) (LPStaking.sol#146-175)
```

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, 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.