Lo-Fi Pepe NFT Security Review

Version 2.0

August 30, 2023

Conducted by:
Georgi Georgiev (Gogo), Independent Security Researcher

Audited through the **Hyacinth** platform.

Table of Contents

1 About Gogo

Gogo is an independent security researcher specializing in Solidity smart contracts auditing and bug hunting. Having conducted numerous solo and team smart contract security reviews, he always strives to deliver top-quality security auditing services. For security consulting, you can contact him on Twitter, Telegram, or Discord - *@gogotheauditor*.

2 Disclaimer

Audits are a time, resource and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to find as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

3 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that’s not so critical.

3.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

3.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

Overview

Project Name	Lo-Fi Pepe NFT
Repository	https://github.com/0xVins/lofipepestaking
Commit hash	970dcd8b1a07f869351c73425b51c10cf9916556
Resolution	862f9b9078a8287860ffaff12938e16f6f11a141
Methods	Manual review

Scope

contract/Staking.sol

Issues Found

Critical risk	0
High risk	2
Medium risk	2
Low risk	3
Informational	1

5 Findings

5.1 High severity

5.1.1 Rewards distribution timeline is incorrectly implemented

Severity: *High Risk*

Context: Staking.sol#L458

Description: As discussed with the client, the rewards should be emitted over a period of 90 days and should have a specific timestamp as a deadline, which is the same for all staking positions.

However, instead of having all rewards emissions stop after a certain timestamp, the deadline is currently set differently for each NFT staking. Instead of having all rewards stop after 90 days, for example, from the contract creation, rewards are distributed until 90 days pass since a position's `stakeTime` or until the reward balance becomes 0.

```
uint256 maxTimestamp = staking.stakeTime + 90 days;
uint256 curTimestamp = block.timestamp;

uint256 releaseTime = curTimestamp < maxTimestamp
    ? curTimestamp
    : maxTimestamp;

require(
    releaseTime > staking.releaseTime,
    "You already claimed all the reward for 90 days"
);

amount = rate / 90 days * (releaseTime - staking.releaseTime);
```

Consider the following scenario:

1. 500 users staked 1 NFT on January 1st, 2023.
2. All of them will accumulate rewards for the next 90 days unless they withdraw earlier.
3. 250 users withdraw their staked NFTs on day 45 and claim their rewards.
4. Now 250 spots are open and 250 new users come to stake their NFT.
5. The problem is that instead of accumulating rewards for the remaining 45 days, the new users will also accumulate rewards for 90 days starting from the moment each one executed the stake.
6. Therefore, after the 90th day has passed (March 31st), the accumulated rewards from that point onwards will exceed the planned resources.
7. If the late 250 stakers wait for 45 days more and claim their rewards before the other 250 stakers, the first 250 stakers will be left with no rewards, even if they have staked for `90 days + 45 days`.

Recommendation: Instead of using `staking.stakeTime + 90 days` as the end timestamp for each separate staking, consider setting explicit start and end timestamps for the reward distribution.

Resolution: Resolved in commit c13bbd4 by setting the start timestamp to September 10th 2023, 00:00 UTC and end timestamp to December 9th 2023, 00:00 UTC (90 days staking period).

5.1.2 Adversary can permanently block the NFT staking functionality

Severity: *High Risk*

Context: Staking.sol#L345-L349

Description: Users are allowed to maintain up to 20 active NFT stakings at once. To ensure that a user does not attempt to stake more than 20 NFTs, the following logic is implemented in the `callStakeToken` function:

```
uint256 cnt = 0;
for (uint256 index = 0; index < _stakingId; index++){
    if(_StakedItem[index].staker == msg.sender && _StakedItem[index].status ==
        StakingStatus.Active) {
        cnt++;
    }
}
require(cnt + _tokenID.length <= 20, "you can stake up to 20 tokens");
```

The problem arises when the for loop causes the transaction to revert due to reaching the block gas limit (30M on Ethereum) once the value of the `_stakingId` variable becomes too high. This situation can occur during the normal flow/execution by having an excessive number of stakes and unstakes, or by an adversary who repetitively stakes and unstakes the same NFTs multiple times to artificially increase the `_stakingId` shortly after the contract is deployed.

Recommendation: Consider tracking the number of active NFT stakings a user has, instead of iterating through all `_stakingId`s every time a user wants to stake. Alternatively, consider removing the cap of 20 NFTs.

Resolution: Resolved in commit c89c1a1 by tracking the number of active NFT stakings a user has.

5.2 Medium severity

5.2.1 Stakers will lose their accrued rewards if not claimed before withdrawal

Severity: *Medium Risk*

Context: Staking.sol#L453-L456, Staking.sol#L497

Description: Users can stake their NFTs for as long as they wish, but their stake will accrue rewards for a maximum of 90 days. Stakers are permitted to withdraw their NFTs and halt reward accumulation at any point.

However, if a user has staked their NFT for, for example, 90 days, and then decides to withdraw it using the `unStake` function, they will lose all of their rewards if they do not first call `claimReward`.

Recommendation: Consider either executing the `claimReward` function logic with each withdrawal or ensuring that the `staking.releaseTime` (the last time `claimReward` was called) falls within an acceptable time period before allowing a withdrawal.

Resolution: Resolved in commit c13bbd4 by claiming accumulated rewards when unstaking an NFT.

5.2.2 Rewards are incorrectly distributed when the reward rate is changed

Severity: *Medium Risk*

Context: Staking.sol#L470, Staking.sol#L528-L530

Description: When users stake their NFTs, they receive rewards in Pepe tokens at a rate determined by the staking contract's owner.

The `rate` variable is used in the `claimReward` function to calculate the reward amount earned since the previous claim or since the stake was initiated.

```
uint256 maxTimestamp = staking.stakeTime + 90 days;
uint256 curTimestamp = block.timestamp;

uint256 releaseTime = curTimestamp < maxTimestamp
    ? curTimestamp
    : maxTimestamp;

require(
    releaseTime > staking.releaseTime,
    "You already claimed all the reward for 90 days"
);

amount = rate / 90 days * (releaseTime - staking.releaseTime);
```

The issue arises when the reward rate is updated by the contract owner. The accrued but unclaimed rewards for all stakers are miscalculated, as the new rate is applied to the entire period since the last claim. Consider the following scenario:

1. Day 1: Alice stakes her NFT when the reward `rate` is 10,000 tokens.
2. Day 45: The contract owner changes the reward `rate` to 5,000 tokens per 90 staking days.
3. Day 90: Alice claims her accrued rewards for the full 90-day staking period at the latter `rate`, accumulating a total of 5,000 tokens.

Alice effectively loses 2,500 reward tokens, as the accrued rewards for the initial 45-day period are not accurately calculated using the previous reward `rate`. If Alice claimed her rewards on day 45, just before the `rate` update, she would have received a total of `45/90 * 10,000 + 45/90 * 5,000 = 7,500` tokens instead of only 5,000.

Similarly, if the reward `rate` is increased from 10,000 to 20,000 tokens, Alice would receive more tokens than her actual accrual if she does not claim her rewards on day 45.

Recommendation: The fix with the current implementation is not trivial. Consider removing the `setRewardRate` function.

Resolution: Resolved in commit c13bbd4 by removing the described functionality.

5.3 Low severity

5.3.1 Incorrect NFT contract address used upon deployment

Severity: *Low Risk*

Context: Staking.sol#L280

Description: The NFT contract address hardcoded in the staking contract is incorrect as it does not belong to the intended Lo-Fi Pepe NFT contract:

```
// @audit Correct address is 0x0fcbd68251819928c8f6d182fc04be733fa94170.
address private NFTToken = 0xAA65c27Df3d2a998C4564015E85761aF4af0B2a9;
```

The severity is low as this issue will only block deposits initially as the contract owner can simply change the NFT contract address via the `setNFTAddress` function.

Recommendation: Ensure the accurate NFT contract address is used during deployment.

Resolution: Resolved in commit c89c1a1 by using the correct NFT contract address.

5.3.2 Default enum value will incorrectly mark non-existent staking positions as active

Severity: *Low Risk*

Context: Staking.sol#L291-L295

Description: The `StakingStatus` enum is used to determine whether an NFT stake is `Active` or `Cancelled`. A potential issue arises from the default value of the enum (the first element) being set to `Active`, implying that stakings which do not actually exist would be incorrectly considered `Active`. This could occur when attempting to read a staking from the `_StakedItem` mapping using a key equal to or higher than `_stakingId`. However, the code consistently verifies that `staking.staker` is equal to the `msg.sender`, which prevents the use of an invalid `_stakingId`.

Recommendation: Consider adding an additional element at the first position in the `StakingStatus` enum to indicate stakings that are `NotActive`.

Resolution: Resolved in commit c89c1a1 by adding an `InActive` enum property at index 0.

5.3.3 Missing input validation in admin setter functions

Severity: *Low Risk*

Context: Staking.sol#L514-L563

Description: The admin of the staking contract currently possesses the following privileges:

1. Changing the NFT contract address used for staking.
2. Changing the reward token contract address used for claiming rewards.
3. Updating the reward rate.
4. Withdrawing all reward tokens deposited for stakers to claim as rewards.
5. Transferring the ownership of the contract to another address.

Although the first two functionalities do not impact the normal execution of the contract, it is not recommended to reuse the same contract for allowing users to stake a different NFT or receive rewards in an different reward token. Changing the reward token is dangerous if the reward rate is not simultaneously updated in the same transaction. If issue 5.2.2 is resolved, being able to change the reward token could become a serious vulnerability.

Furthermore, the owner currently possesses the ability to set the reward `rate` to any arbitrary value without validation whether enough tokens have actually been deposited into the contract. The owner can also withdraw any reward tokens sent to the contract, potentially making stakers unable to claim their accrued rewards.

Recommendation: Consider implementing an appropriate upper limit for the reward `rate` variable. Additionally, consider using OpenZeppelin's Ownable or Ownable2Step contract for additional safety measures.

Resolution: Resolved in commit c13bbd4. The contract's owner no longer has the ability to change the NFT and reward token contract addresses nor the reward rate.

5.4 Informational

5.4.1 Lack of zero array length checks may cause unexpected behavior

Severity: *Informational*

Context: Staking.sol#L340, Staking.sol#L485

Description: The staking and unstaking functions accept and iterate through an array of `tokenId`s and `stakingId`s. However, a user can pass an empty array, potentially leading to unexpected behavior, as there is no check for this scenario.

Recommendation: Consider implementing a check to ensure that the length of input arrays is greater than 0.

Resolution: Resolved in commit c89c1a1.