audita

NFPrompt

Security Audit - Token, Vesting, Airdrop

December 25, 2023

Prepared for: NFPrompt (nfprompt.*io*)

Presented By: Audita Security (*audita.io*)

Document

The contents of this document may include confidential information pertaining to the IT systems, intellectual property, and possible vulnerabilities along with methods of exploitation that the Client may possess. The report that contains this confidential information can be utilized internally by the Client, and can be made available to the public after all vulnerabilities are addressed, depending on the decision of the Client.

Approved by: Jose Saez Lopez, Reni Dimitrova @ Audita.io
Contracts: *NFPToken.sol, TokenVesting.sol, MerkleAirdrop.sol*
Network: Binance Smart Chain
Programming language: Solidity
Method: Manual Audit by Solidity Experts
Client Website: https://nfprompt.io/
Timeline: 21/12/2023 - 25/12/2023

Table of Contents

Document **1**

Executive Summary **3**

Fixes 3

Audita Vulnerability Classifications **5**

Scope **6**

Findings **7**

Summary - MerkleAirdrop.sol 7

Summary - Vesting.sol 8

Detailed Findings - MerkleAirdrop.sol 10

Detailed Findings - Vesting.sol 12

Recommendations **22**

Fixes **23**

Executive Summary

Manual Audit

During the manual audit conducted by our experts, we did not identify any Critical severity vulnerabilities.

We identified 0 High, 1 Medium and 13 Low severity vulnerabilities.

11 Informational and 3 GAS issues were indicated, relating to:

- Code Quality
- Gas Optimization

Fixes

Issues were acknowledged by NFPrompt's team. Fixes were introduced in the Vesting contracts.

Overall Assessment

Severity	Count	Acknowledged	Addressed
Critical	-	-	Refer to Findings Summary for Fixes already introduced in the Vesting contract.
High	-	-	
Medium	1	Yes	
Low	13	Yes	
Informational	11	Yes	
GAS	3	Yes	

Documentation

We recommend selected information from this report, as well as the contracts' README files to be included in protocol's official Documentation, as soon as code is deployed.

Test Coverage

This audit was performed under the assumption that there is a total test coverage of 0%.

Audita Vulnerability Classifications

Audita follows the most recent standards for vulnerability severities, taking into consideration both the possible impact and the likelihood of an attack occurring due to a certain vulnerability.

Severity	Description
Critical	Critical vulnerability is one where the attack is more straightforward to execute and can lead to exposure of users' data, with catastrophic financial consequences for clients and users of the smart contracts.
High	The vulnerability is of high importance and impact, as it has the potential to reveal the majority of users' sensitive information and can lead to significant financial consequences for clients and users of the smart contracts.
Medium	The issue at hand poses a potential risk to the sensitive information of a select group of individual users. If exploited, it has the potential to cause harm to the client's reputation and could result in unpleasant financial consequences.
Low	The vulnerability is relatively minor and not likely to be exploited repeatedly, or is a risk that the client has indicated is not impactful or significant, given their unique business situation.
Informational	The issue may not pose an immediate threat to ongoing operation or utilization, but it's essential to consider implementing security and software engineering best practices, or employing backup measures as a safety net.

Scope

The security assessment was scoped to the following smart contracts in token-generation repository:

Contract names
`NFPToken.sol`
`TokenVesting.sol`
`MerkleAirdrop.sol`

The code has been audited up to and including commit *515183ffcf4f491a3db84575548ab3b84dfba844*

Findings

Summary - MerkleAirdrop.sol

Risk:

The contract does not perform checks on its token balance, leading to a lack of assurance that users will receive their tokens. Without verification, there's a risk that the contract might attempt to distribute more tokens than it holds, resulting in failed transactions.

Code	Description	Severity	Fixes
[M-AIR-01]	Withdrawals handling	Low	TBD
[M-AIR-02]	No events being emitted upon claim, changeMerkleRoot, changestartAt and withdraw functions	Low	TBD
[M-AIR-03]	Make all variables public	Informational	TBD
[M-AIR-04]	Error Handling	Informational	TBD
[M-AIR-05]	bytes.concat usage	Informational	TBD
[M-AIR-06]	Encode Usage	Informational	TBD
[M-AIR-07]	MerkleRoot as an immutable property	Informational	TBD
[M-AIR-08]	startAt as an immutable property	Informational	TBD

Summary - Vesting.sol

Code	Description	Severity	Fixes
[VEST-01]	Possible rounding issue	Medium	Partially Fixed
[VEST-02]	Front-running Risk: Vesting Revocation near cliff	Low	Acknowledged
[VEST-03]	Contradiction - Code does not revert if vesting does not exist	Low	Fixed
[VEST-04]	No revert reason for require statements	Low	Fixed
[VEST-05]	Emitting events for state-changing functions	Low	Acknowledged
[VEST-06]	Redundant getCurrentTime function	Low	Fixed
[VEST-07]	Redundant code - removal	Low	Fixed
[VEST-08]	(computeReleaseAmount) currentTime equal to cliff mechanism	Low	Fixed
[VEST-09]	Private/public variables; default getters	Low	Acknowledged
[VEST-10]	Missing check for beneficiary being non-zero	Low	Fixed
[TOKEN-11]	Missing check for admin address being non-zero	Low	Acknowledged
[ALL-12]	Floating Pragma (all contracts)	Low	Fixed
[VEST-13]	Incorrect code comment - Withdrawer role	Informational	Fixed
[VEST-14]	(withdraw) Remove provision of desired amount upon withdraw	Informational	Fixed

[VEST-15]	(release) Remove provision of desired amount upon release	Informational	Fixed
[VEST-16]	Release function optimization	Informational	Fixed
[VEST-17]	Commented code	Informational	Fixed
[VEST-18]	Gas Inefficiency in Assignment	GAS	Fixed
[VEST-19]	(VestingCount) Redundant variable assignment	GAS	Fixed
[VEST-20]	(computeReleasableAmount) Redundant validation	GAS	Fixed

Detailed Findings - MerkleAirdrop.sol

[M-AIR-01]	Withdrawals handling	Low

Details:

Withdrawal of a user's balance should result in the user not being able to claim.

Recommendation:

Introduce endTime after which it is clear that the users who have not claimed yet will lose their tokens as the owner could withdraw them.

[M-AIR-02]	No events being emitted upon claim, changeMerkleRoot, changestartAt and withdraw functions	Low

Recommendation:

Emit events upon `claim`, `changeMerkleRoot`, `changestartAt` and `withdraw` functions.

[M-AIR-03]	Make all variables public	Informational

Recommendation:

Make all variables public.

[M-AIR-04]	Error Handling	Informational

Recommendation:

It is recommended that you choose an error style. Either user “require” or “revert” as a best practice.

[M-AIR-05]	bytes.concat usage	Informational

Recommendation:

Simply use the result from keccak256.

[M-AIR-06]	Encode Usage	Informational

Recommendation:

Use `encodePacked` instead of `encode`. However, keep in mind that `encodePacked` uses more gas.

[M-AIR-07]	MerkleRoot as an immutable property	Informational

Recommendation:

`merkleRoot` should be an immutable property since the moment of deployment.

[M-AIR-08]	startAt as an immutable property	Informational

Recommendation:

`startAt` should be an immutable property since the moment of deployment.

Detailed Findings - Vesting.sol

[VEST-01]	Possible rounding issue	Medium	Partially Fixed

Details:

Path: TokenVesting: `_computeReleasableAmount`

In the `_computeReleasableAmount` function there's a computation to determine the vestedAmount of tokens based on the time elapsed (vestedSeconds) and the total amount of tokens allocated for vesting (vestingSchedule.amountTotal).
If `vestingSchedule.amountTotal` is very small and unvalidated, the calculation for vestedAmount might result in zero due to Solidity's integer division.
This could lock tokens until the end of the vesting period.

Recommendation:

When creating the vesting in the `createVestingSchedule` function ensure that `_amount * _slicePeriodSeconds / _duration > 0`.

Fixes:

Modification was introduced, but there is still a problem with rounding if **_amount * _slicePeriodSeconds** is bigger by a few Wei. In such a case the rounding could occur again.

[VEST-02]	Front-running Risk: Vesting Revocation near cliff	Low	Acknowledged

Details:

Path: TokenVesting: revoke

There's a risk in vesting contracts where the owner tries to revoke a beneficiary's vesting close to the vesting cliff. The beneficiary could potentially front-run the owner's transaction, allowing them to claim the tokens that become available at the cliff before the revocation takes effect.

Recommendation:

The likelihood of this occurring is relatively low. However, as there is no straightforward smart contract solution to completely mitigate this risk, it is advisable for the owner to avoid initiating revocations near the vesting cliff period.

[VEST-03]	Contradiction - Code does not revert if vesting does not exist	Low	Fixed

Details:

Path: TokenVesting : `onlyIfVestingScheduleNotRevoked`

In the `onlyIfVestingScheduleNotRevoked` modifier comment it is stated: “Reverts if the vesting schedule does not exist or has been revoked.”, however, code does not revert if the schedule does not exist. Therefore, the non-existing schedule may be accessed and the requirements - violated.

Recommendation:

Ensure that schedule is created in the `onlyIfVestingScheduleNotRevoked` modifier.

[VEST-04]	No revert reason for require statements	Low	Fixed

Details:

There are “require” statements having no revert reason.

Recommendation:

It is a best practice to always have a reason for reverting.

[VEST-05]	Emitting events for state-changing functions	Low	Acknowledged

Recommendation:

It is a best practice to emit an event for every state-changing function. Such functions are createVestingSchedule/revoke/release.

[VEST-06]	Redundant getCurrentTime function	Low	Fixed

Details:

Path: TokenVesting : `getCurrentTime`

Using a separate function to obtain block.timestamp requires spending more Gas.

`getCurrentTime` is an internal function of a single line doing no logic operations.

Recommendation:

To save gas cost and contract size, we recommend using directly block.timestamp where needed.

[VEST-07]	Redundant code - removal	Low	Fixed

Recommendation:

Remove the following commented code:

```
// address payable beneficiaryPayable = payable(
//     vestingSchedule.beneficiary
// );
```

[VEST-08]	(computeReleaseAmount) currentTime equal to cliff mechanism	Low	Fixed

Details:

In case the currentTime is lower than the cliff the released amount is 0.

Does it make sense in case currentTime = cliff, the released amount to be 0 as well?

Recommendation:

Double-check if this is the desired functionality.

[VEST-09]	Private/public variables; default getters	Low	Acknowledged

Recommendation:

Instead of defining your getters, benefit by using the default ones. To do so, make all the private variables public ones. There is no reason for them to be private anyway.

[VEST-10]	Missing check for beneficiary being non-zero	Low	Fixed

Details:

Path: TokenVesting : createVestingSchedule

When creating the vesting there is no check if _beneficiary address is not equal to zero.

Recommendation:

Recommended to implement this check as this may lead to burning of tokens.

[TOKEN-11]	Missing check for admin address being non-zero	Low	Acknowledged

Details:

Path: NFPToken: constructor

When creating the NFPToken token in the constructor, it is not checked if the admin is not a zero address. This may result in not having an admin.

Recommendation:

Verify if admin is not a zero address.

[ALL-12]	Floating Pragma (all contracts)	Low	Fixed

Details:

Path: NFPToken, TokenVesting, Airdrop

The code uses a floating pragma (pragma solidity ^0.8.19;). This could lead to unintended behavior if the contract is compiled with a newer, potentially incompatible Solidity compiler version.

Recommendation:

Lock the pragma.

[VEST-13]	Incorrect code comment - Withdrawer role	Informational	**Fixed**

Details:

Path: TokenVesting: withdraw

The WITHDRAWER_ROLE is mentioned in the withdraw function. However, such a role does not exist in the contract. This may indicate that the requirements are violated.

Recommendation:

Ensure that comments match the implementation.

[VEST-14]	(withdraw) Remove provision of desired amount upon withdraw	Informational	**Fixed**

Details:

An admin can provide the amount to be taken out in the “withdraw” function. That only increases contract size & gas cost.

Recommendation:

We recommend withdrawing everything available instead of providing a desired amount.

[VEST-15]	(release) Remove provision of desired amount upon release	Informational	Fixed

Details:

A beneficiary can provide the desired amount to be released.

Recommendation:

To make the code more readable and gas-optimized as well, we recommend releasing everything available instead of providing the desired amount. In short, simply work with `_computeReleasableAmount` instead of the provided amount. On the other hand, if part of available tokens are stored for future vesting, withdrawing all of them will lead to the extra operation of transferring them back, so make sure you implement accordingly.

[VEST-16]	Release function optimization	Informational	Fixed

Details:

To optimize “release” function in terms of gas we recommend replacing the following lines:

```
bool isBeneficiary = msg.sender == vestingSchedule.beneficiary;
bool isReleasor = (msg.sender == owner());
```

with

```
if(msg.sender == vestingSchedule.beneficiary || msg.sender == owner())
```

[VEST-17]	Commented code	Informational	Fixed

Details:

Path: TokenVesting, NFPToken

The commented code presents in the contracts and may indicate that the code is not finalized.

Recommendation:

If this is not an issue, proceed with the commented code.

[VEST-18]	Gas Inefficiency in Assignment	GAS	Fixed

Details:

Path: TokenVesting : createVestingSchedule

Using `vestingSchedulesTotalAmount = vestingSchedulesTotalAmount + _amount`; is less gas-efficient due to separate addition and assignment operations.

Recommendation:

Switch to `vestingSchedulesTotalAmount += _amount`; for optimized gas usage, combining the operations into one.

[VEST-19]	(VestingCount) Redundant variable assignment	GAS	Fixed

Details:

Path: TokenVesting : createVestingSchedule

The code uint256 currentVestingCount = holdersVestingCount[_beneficiary]; followed by holdersVestingCount[_beneficiary] = currentVestingCount + 1; involves an unnecessary intermediate variable, leading to higher gas costs.

Recommendation:

Optimize by using the ++ operator directly:

`holdersVestingCount[_beneficiary]++;`

[VEST-20]	(computeReleasableAmount) Redundant validation	GAS	Fixed

Details:

Path: TokenVesting : `_computeReleasableAmount`

It is validated in the `_computeReleasableAmount` function if the vesting is not revoked. However, the `_computeReleasableAmount` function is called in the functions which have the `onlyIfVestingScheduleNotRevoked` modifier. Due to this, this check is repeated and requires redundant Gas spending.

Recommendation:

Remove the redundant check.

Overall Assessment

Severity	Count	Acknowledged	Addressed
Critical	-	-	Refer to Findings Summary for Fixes already introduced in the Vesting contract.
High	-	-	
Medium	1	Yes	
Low	13	Yes	
Informational	11	Yes	
GAS	3	Yes	

Recommendations

Audita has put forward the following recommendations for NFPrompt's contracts:

MerkleAirdrop.sol

- Consider the following risk:
  The contract does not perform checks on its token balance, leading to a lack of assurance that users will receive their tokens. Without verification, there's a risk that the contract might attempt to distribute more tokens than it holds, resulting in failed transactions.
- Introduce endTime upon withdrawals, so that it is clear that the users who have not claimed yet will lose their tokens as the owner could withdraw them.
- Emit events upon `claim`, `changeMerkleRoot`, `changestartAt` and `withdraw` functions.
- Make all variables public.
- It is recommended that you choose an error style. Either user "require" or "revert" as a best practice.
- Instead of using bytes.concat, simply use the result from keccak256.
- Use `encodePacked` instead of `encode`. However, keep in mind that `encodePacked` uses more gas.
- `merkleRoot` should be an immutable property since the moment of deployment.
- `startAt` should be an immutable property since the moment of deployment.

Fixes

NFPrompt's team are dedicated and responsive, cooperating to acknowledge and implement the above recommendations.

Fixes were introduced in the Vesting contracts, please refer to Findings Summary. MerkleAirdrop is also currently being finalized.

Disclaimer

This audit makes no statements or warranties on the security of the code. This report should not be considered a sufficient assessment on the safety of the code, quality status, or any other contract statements. **While we have conducted the analysis to our best abilities and produced this report in line with latest industry developments, it is important to not rely on this report only.** In order for contracts to be considered as safe as possible, the industry standard requires them to be checked by several independent auditing bodies. Those can be other audit firms or public bounty programs.

The contacts live on a blockchain (a smart contract platform) - Smart contract platforms, their programming languages, and other software components are not immune to vulnerabilities that can be exploited by hackers. As a result, although a smart contract audit can help identify potential security issues, it cannot provide an absolute guarantee of the audited smart contract's security.