audita

NFPrompt

Security Audit - Token, Vesting, Airdrop

December 16, 2023

Prepared for: NFPrompt (nfprompt.*io*)

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

Document

Approved by: Jose Saez Lopez, Reni Dimitrova @ Audita.io
Contracts: *NFPToken.sol, TokenVesting.sol, Airdrop.sol*
Network: Binance Smart Chain
Programming language: Solidity
Method: Manual Audit by Solidity Experts
Client Website: https://nfprompt.io/
Timeline: 14/12/2023 - 21/12/2023

Table of Contents

Executive Summary

Fixes

TBD - Information on Fixes introduced will be included here in the **Final Audit Report**.

Manual Audit

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

We identified 2 High, 2 Medium and 16 Low severity vulnerabilities.

9 Informational and 4 GAS issues were indicated, relating to:

- Code Quality
- Gas Optimization

Overall Assessment

Severity	**Count**	**Acknowledged**	**Addressed**
Critical	0	-	Refer to Findings Summary for details on fixes and addressed issues
High	2	TBD	
Medium	2	TBD	
Low	16	TBD	
Informational	9	TBD	
GAS	4	TBD	

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`
`Airdrop.sol`

Findings

Summary - Airdrop.sol

Code	Description	Severity	Fixes
[AIR-01]	(totalAirdropAmount) Inconsistent data - Underflow risk	High	TDB
[AIR-02]	(register, registerBatch) Missing token management system to ensure enough airdrop tokens on balance when creating airdrops	High	TDB
[AIR-03]	Removing users' balances risk - Locked tokens	Medium	TDB
[AIR-04]	All variables with public access modifiers to benefit from default getters	Low	TDB
[AIR-05]	(remove function) Two IF statements	Low	TDB
[AIR-06]	(claimedBlock) claimedBlock usage	Low	TDB
[AIR-07]	(claimed) claimed usage	Low	TDB
[AIR-08]	Emitting events for state-changing functions	Low	TDB
[AIR-09]	Redundant Import - Imported Address library	Informational	TDB
[AIR-10]	Inability to edit airdrop amount	Informational	TDB
[AIR-11]	Pause/unpause functionality	Informational	TDB
[AIR-12]	(startTime) Mutability	Informational	TDB
[AIR-13]	(register) Register function in registerBatch	GAS	TDB

Summary - Vesting.sol

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

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

Detailed Findings - Airdrop.sol

[AIR-01]	(totalAirdropAmount) Inconsistent data - Underflow risk	High

Details:

Path: Airdrop: register, remove, claim

`totalAirdropAmount` value is not updated when registering the account via `Register` function. That means upon remove, if all the users are removed, the function will start reverting (underflow) as the totalAirdropAmount will be less than the real total amount.

If the actual total airdrop amount and the one stored are different, this may lead to inability to remove the user from the airdrop, as in the remove function the amount is deducted from the totalAirdropAmount.
Additionally, the totalAirdropAmount is not updated in the claim function, which may indicate that the requirements are violated.

Recommendation:

Ensure totalAirdropAmount is updated in the register, remove, and claim functions to accurately reflect the actual total airdrop amount. This will prevent discrepancies and maintain consistency in airdrop management.

[AIR-02]	(register, registerBatch) Missing token management system to ensure enough airdrop tokens on balance when creating airdrops	High

Details:

Path: Airdrop : register, registerBatch

The contract lacks a verification mechanism to ensure there are enough airdrop tokens on the contract balance when creating airdrops via `register` and `registerBatch` functions. This discrepancy between the stored airdrop amount and the actual balance may lead to insufficient funds errors.
Furthermore, due to the absence of a token management system, the contract owner must directly send tokens to the contract. This approach also implies that the owner cannot withdraw excess tokens, for instance, in scenarios where a beneficiary is removed from the airdrop using the remove function.

Recommendation:

Integrate a token management system to verify token availability before airdrop creation and enable extra token withdrawal.

[AIR-03]	Removing users' balances risk - Locked tokens	Medium

Details:

By removing users' balances the tokens will remain locked into the contract.

Recommendation:

Consider a mechanism to withdraw these tokens similar to the one you've implemented in the vesting contract.

[AIR-04]	All variables with public access modifiers to benefit from default getters	Low

Recommendation:

Make all the variables with public access modifiers to benefit from the default getters.

[AIR-05]	(remove function) Two IF statements	Low

Details:

There are 2 if statements in remove function.

As that is onlyOwner function, the worst scenario is for the Owner to spend gas to remove a user with a zero balance.

Recommendation:

They could be skipped to decrease gas cost and complexity. Reverts are typically used for exception handling. They should be used to not break the contract state, not to guard for misunderstanding.

[AIR-06]	(claimedBlock) claimedBlock usage	Low

Details:

claimedBlock is not used by the contract anywhere in terms of logic execution.

Recommendation:

It could be removed. In case it is needed for front-end purposes, then emit an event and store it off-chain. No need to add complexity to the contract.

[AIR-07]	(claimed) claimed usage	Low

Details:

claimed could be skipped as well by using only the balance of the user.

Recommendation:

If balance=0, then it can be considered already claimed. If this is needed for front-end purposes to know if someone has participated, it's better to use an off-chain service storing the registered accounts.

[AIR-08]	Emitting events for state-changing functions	Low

Details:

There are no events.

Recommendation:

There are no events. It is a best practice to emit an event in every state-change function. Such functions are register/registerBatch/claim/remove.

[AIR-09]	Redundant Import - Imported Address library	Informational

Details:

Path: Airdrop: Address

The Address library imported in the Airdrop contract is never used. Redundant code decreases the code readability and increases the contract size.

Recommendation:

Remove the redundant import.

[AIR-10]	Inability to edit airdrop amount	Informational

Details:

Path: Airdrop: `_addToInfo`

The `_addToInfo` function does not allow changing the info.balance value. The problem may arise in case the airdrop amount should be changed for the user.

Recommendation:

Ensure the inability to edit airdrop amount matches your requirements. If not, implement such functionality.

[AIR-11]	Pause/unpause functionality	Informational

Details:

The only thing that pause/unpause controls is the claim function.

Pausing is typically considered an emergency function. There is no mechanism to deal with emergencies implemented here.

Recommendation:

Having the above details in mind, pausing/unpausing the claim function could be skipped.

[AIR-12]	(startTime) Mutability	Informational

Recommendation:

We recommend `startTime` to be immutable. This way users will have more transparency and increased trust in the protocol.

[AIR-13]	(register) Register function in registerBatch	**GAS**

Recommendation:

Reuse register function in registerBatch function to optimize gas and complexity.

Detailed Findings - Vesting.sol

[VEST-01]	Possible rounding issue	Medium

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.

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

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

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

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

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

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

Recommendation:

Remove the following commented code:

```
// address payable beneficiaryPayable = payable(

//     vestingSchedule.beneficiary
// );
```

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

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

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

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

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

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

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`

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`

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

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

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

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

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

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	0	-	TBD
High	2	TBD	
Medium	2	TBD	
Low	16	TBD	
Informational	9	TBD	
GAS	4	TBD	

Recommendations

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

Airdrop.sol

- Ensure totalAirdropAmount is updated in the register, remove, and claim functions to accurately reflect the actual total airdrop amount.
- Integrate a token management system to verify token availability before airdrop creation and enable extra token withdrawal.
- Consider a mechanism to withdraw tokens upon user deletion, so that they don't end up locked in the contract.
- Make all the variables with public access modifiers to benefit from the default getters.
- Skip the two IF statements in `remove` function to decrease gas cost and complexity.
- `claimedBlock` is not used by the contract anywhere in terms of logic execution - it could be removed.
- `claimed` could be skipped - use only the balance of the user.
- Emit an event in every state-change function.
- Remove the redundant Address library import.
- Ensure the inability to edit airdrop amount matches your requirements. If not, implement such functionality.
- Pausing/unpausing the claim function could be skipped, as it controls only the claim function.
- Make `startTime` immutable. This way users will have more transparency and increased trust in the protocol.
- Reuse register function in registerBatch function to optimize gas and complexity.

Vesting.sol

- When creating the vesting in the createVestingSchedule function ensure that _amount * _slicePeriodSeconds / _duration > 0.
- Due to front-running risk, it is advisable for the owner to avoid initiating revocations near the vesting cliff period.
- Ensure that schedule is created in the onlyIfVestingScheduleNotRevoked modifier, so that the code matches the desired functionality and reverts if there is no schedule.
- There are "require" statements having no revert reason. It is a best practice to always have a reason for reverting.
- Emit an event for every state-changing function.
- To save gas cost and contract size, we recommend using block.timestamp instead of getCurrentTime.
- Remove the following commented code:

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

// );
```

- Double-check if whenever currentTime = cliff, the desired functionality is for the release amount to be 0.
- Instead of defining your getters, benefit by using the default ones. To do so, make all the private variables public ones.
- Recommended to check if _beneficiary address is not equal to zero.
- Ensure that comments match the implementation regarding the WITHDRAWER role.
- Upon withdraw, We recommend withdrawing everything available instead of providing a desired 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.

- To make the code more readable and gas-optimized, we recommend releasing everything available with `_computeReleasableAmount` instead of providing an amount.
- 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())
  ```
- If this is not going to create an issue, keep commented code the way it is.
- Switch to `vestingSchedulesTotalAmount += _amount`; for optimized gas usage, combining the operations into one.
- Optimize by using the ++ operator directly:
  `holdersVestingCount[_beneficiary]++;`
- The `_computeReleasableAmount` function is called in the functions which have the `onlyIfVestingScheduleNotRevoked` modifier, so this check can be removed to save on Gas spending.

Token.sol

- Recommended to check if admin is not a zero address.

All contracts

- Lock the pragma.

Fixes

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

Information on specific fixes will be included here in the **Final Audit Report**.

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.