audita

Add3

Security Audit - Customizable Web3 Products

January 2024

Prepared for: Add3 Inc. (*add3.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:

VestingModule.sol, StakingModuleDynamic.sol, StakingModuleStatic.sol

Network: EVM-compatibles

Programming language: Solidity

Method: Manual Audit by Solidity Experts

Client Website: https://www.add3.io/

Table of Contents

Executive Summary

Fixes

Refer to Findings Summary for detailed information on fixes introduced.

Manual Audit

During the manual audit conducted by our experts, we identified 1 Critical and 2 High severity vulnerabilities.

Additionally, we identified 2 Medium and 6 Low severity vulnerabilities.

18 Informational and 8 GAS issues were also indicated, relating to:

- Code Quality
- Gas Optimization

Overall Assessment

Severity	Count	Acknowledged	Fixed
Critical	1	Yes	Please refer to Findings Summary for detailed information on fixes introduced.
High	3	Yes	
Medium	2	Yes	
Low	6	Yes	
Informational	18	Yes	
GAS	8	Yes	

Documentation

Detailed public documentation will soon be available at https://www.add3.io/docs.

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:

Contract names
`VestingModule.sol`
`StakingModuleDynamic.sol`
`StakingModuleStatic.sol`

Fixes have been introduced. The codebase has been audited up to and including commit: 030231747fc78a3fadfdb001ca96f73ba1427a98

General

The rewards are contingent upon the implementation of the Controller contract, hence any issues there could impact the reward mechanism. Monitoring, mitigation and further security assessments are advised.

We believe a lot of aspects in the code can be optimized and simplified, such as checks and functionalities. We present the following for your consideration:

- Introducing a role-based approach, such as the OpenZeppelin Access contract, instead of multiple signatures.
- Consider decreasing complexity - add only a schedule per group for vestors, instead of adding many schedules/many groups.
- Make names of different functions more efficient, rename `getOutputFromInput` function.
- The code, and especially DynamicStaking contract is quite complex to follow. Consider rewriting it in a more simplified manner.

Findings

Summary - Staking Contracts

Code	Description	Severity	Fixes
[STATIC-01]	(getRewardsValue) Incorrect Rewards Calculation	Critical	**Fixed**
[S/D-02]	(getOutputFromInput) Rewards Front-Running Vulnerability	High	**Fixed**
[S/D-03]	Highly Permissive Access	High	**Fixed**
[STATIC-04]	Requirements violation	Medium	**Fixed**
[DYNAMIC-05]	DoS Vulnerability	Low	**Fixed**
[STATIC-06]	Switching max and min stake exists - improvement	Low	**Fixed**
[S/D-07]	(minLockDays) Variable name contradiction	Informational	**Fixed**
[S/D-08]	(_harvestAndWithdrawRewards) Function name contradiction	Informational	**Acknowledged**
[BASE-09]	Use the OpenZeppelin Reentrancy contract.	Informational	**Mitigated**
[BASE-10]	Split blackOrWhitelistAddress function to blackList and whiteList	Informational	**Fixed**
[BASE-11]	(transferAccidentalyLockedTokens) Redundant check	Informational	**Fixed**
[DYNAMIC-12]	(_previewApy) Redundant Calls to Controller	GAS	**Fixed**

[S/D-13]	(maxStake, minStake) Redundant Storage Variables Assignments	GAS	**Fixed**
[S/D-14]	(minLockDays) Redundant Access to a Storage Variable	GAS	**Fixed**
[STATIC-15]	(_calculateRewards) Redundant Variable Assignment	GAS	**Fixed**
[STATIC-16]	(initialize) Redundant Access to a Storage Variable	GAS	**Fixed**
[DYNAMIC-17]	(updateRatesHistory) Redundant Storage Variable Access	GAS	**Fixed**

Summary - Vesting Contracts

[VEST-01]	IF statement in VestingModule might be mistakenly placed	High	**Fixed**
[VEST-02]	Lack of mechanism to disable oracle in case of compromise	Low	**Acknowledged**
[VEST-03]	(feeSplitter) Redundant check if sendToAdd3 function has executed correctly	Low	**Fixed**
[VEST-04]	9000 and 10000 as variables	Low	**Fixed**
[VEST-05]	(airdropVestedTokens) Redundant Calls to getGroupGrantSchedule	GAS	**Fixed**
[VEST-06]	(initialize) Redundant checks if an address is a contract	GAS	**Fixed**
[VEST-07]	Missing addVestors._recipients and addVestors._amounts arrays lengths equality verification	Informational	**Fixed**

[VEST-08]	Inconsistent naming in vesting contract	Informational	**Fixed**
[VEST-09]	Use OpenZeppelin Ownable contract.	Informational	**Acknowledged**
[VEST-10]	Move signatures and validations to a Vesting Auth library	Informational	**Fixed**
[VEST-11]	(getEthInputFee) Multiple blockchains logic consideration	Informational	**Acknowledged**
[VEST-12]	Rename groups in getVestorGroupAmountSum	Informational	**Fixed**

Summary - All Contracts

[ALL-01]	(feeReturner, feeSplitter) Non-fixed fees	Medium	**Fixed**
[ALL-02]	No functionality to revoke signatures	Low	**Acknowledged**
[ALL-03]	Redundant receive function	Informational	**Fixed**
[ALL-04]	Redundant events	Informational	**Fixed**
[ALL-05]	Redundant errors	Informational	**Fixed**
[ALL-06]	Commented code	Informational	**Fixed**
[ALL-07]	Use of Unspecified uint Sizes in Contracts	Informational	**Acknowledged**
[ALL-08]	Too many checks in customModifier	Informational	**Fixed**
[ALL-09]	Move all the events, all structs and all errors in the interface	Informational	**Acknowledged**

Detailed Findings - Staking Contracts

[STATIC-01]	(getRewardsValue) Incorrect Rewards Calculation	Critical

Details:

The documentation states that rewards should be calculated for the duration between the start of staking and the minLockDay. However, in the getRewardsValue function, the check is implemented as maturingBalance.timestamp >= stakeTimeUser + minLockDaysVar. Here, maturingBalance.timestamp represents the time of the last update of the user's position. If the last update was made within stakeTimeUser + minLockDaysVar, then maturingBalance.timestamp will always be less than stakeTimeUser + minLockDaysVar.

This leads to an insufficient rewards issue. For example, if a user stakes at 1 second and the minLockDaysVar is 5 seconds, then unstakes at 10 seconds, they would receive rewards for 10 seconds. This allows users to withdraw more rewards than they should, potentially leading to a shortage of rewards for other users. The contract checks if there are enough tokens to cover the rewards only for minLockTime, meaning that excessive withdrawals could result in a lack of rewards for others, causing transactions to fail and users not receiving their due rewards.

Recommendation:

Ensure that `getRewardsValue` function calculates the rewards for the `minLockDay` interval.

[S/D-02]	(getOutputFromInput) Rewards Front-Running Vulnerability	High

Details:

A vulnerability exists in the reward calculation process when rewards are in a different token than the staking tokens. The rewards, calculated based on the current token price (via

`getOutputFromInput` function), can be affected by price fluctuations. This exposes the process to front-running risks, where a user's transaction might be preempted by others, changing the token price and resulting in the user receiving less than the expected rewards.

Recommendation:

Implement a feature allowing users to set a minimum reward threshold when they stake, unstake, or harvest rewards. This would protect users from the effects of price volatility and transaction front-running, ensuring a more predictable and secure reward system.

[S/D-03]	Highly Permissive Access	`High`

Details:

`Unstake` and `_harvestAndWithdrawRewards` use customModifier(Controller.amlActive(), customTypes.ADAPTER_NOT_SET. Therefore, if the staking contract is removed from the controller, users are unable to withdraw their stakes and rewards.

Recommendations:

Allow users to withdraw their funds when the staking is removed from the controller.

[STATIC-04]	Requirements violation	`Medium`

Details:

It is stated in the documentation that “In case of missing to harvest the exactly expected amount of Rewards, the contract emits a Warning message reporting Expected and Actual Rewards.”, “In case of warnings the DApp will display users that contact is failing to pay rewards and disables interactions. Then it is up to the contract owner to : Give the Expected Reward - Actual Reward amount to the msg.sender”. However, such functionality

is not implemented, as the `_harvestRewardsFromVault` function never allows for giving less than expected rewards to users.

Recommendation:

Ensure that implementation matches the requirements described in the documentation.

[DYNAMIC-05]	DoS Vulnerability	Low

Details:

The `getLastOutOfLockAndUnstakedDuringLock` function's loop over an unbounded balanceArray could lead to a Denial of Service (DoS) due to excessive gas consumption. As the array size grows with more staking transactions, the function's gas requirement may exceed the transaction limit, causing it to fail. This may lead to the inability to withdraw stake during the minLockTime period.

Recommendation:

Redesign the implementation so that `getLastOutOfLockAndUnstakedDuringLock` does not loop over an unlimited array.

[STATIC-06]	(initialize) Switching max and min stake exists - improvement	Low

Details:

The chosen approach for switching max and min stake exists is a bit “hacky”.

Recommendations:

Simply revert if the check is not met.

[S/D-07]	(minLockDays) Variable name contradiction	`Informational`

Details:

The minLockDays variable stores the seconds, but its title indicates that it stores the days. This may lead to the misunderstanding and incorrect usage of the variable.

Recommendation:

Fix the variable name.

[S/D-08]	(_harvestAndWithdrawRewards) Function name contradiction	`Informational`

Details:

The `_harvestAndWithdrawRewards` variable does not withdraw the reward for the autocompounds staking. Therefore, the title may wrongly indicate the function's behavior.

Recommendations:

Fix the function name.

[BASE-09]	Use the OpenZeppelin Reentrancy contract.	`Informational`

Recommendations:

For improved safety, we recommend implementing OpenZeppelin's reentrancy guard.

[BASE-10]	Split blackOrWhitelistAddress function to blackList and whiteList	Informational

Recommendations:

For clarity and future convenience, we recommend to split the blackOrWhitelistAddress function to `blackList` and `whiteList`.

[BASE-11]	(transferAccidentalyLockedTokens) Redundant check	Informational

Recommendations:

There is a !token.isContract() check in the `transferAccidentalyLockedTokens`. Remove this as the owner is calling it and they are required to provide accurate params.

[DYNAMIC-12]	(_previewApy) Redundant Calls to Controller	GAS

Details:

The function `Controller.productReward` is called three times within the `_previewApy` function. This repetitive calling leads to unnecessary gas consumption and can be optimized.

Recommendation:

It is advisable to call `Controller.productReward` only once and reuse its result in the `_previewApy` function. This change would reduce gas costs and improve the efficiency of the contract.

[S/D-13]	(maxStake, minStake) Redundant Storage Variables Assignments	GAS

Details:

In the initialize function the `maxStake` and `minStake` variables are set twice, which increases the Gas usage.

Recommendation:

Set the `maxStake` and `minStake` variable once after the required verifications.

[S/D-14]	(minLockDays) Redundant Access to a Storage Variable	GAS

Details:

In the initialize function the minLockDays is used for the verification, however the `_minLockDays` can be used to save Gas.

Recommendation:

Set the minLockDays after the required verification of `_minLockDays`.

[STATIC-15]	(_calculateRewards) Redundant Variable Assignment	GAS

Details:

In the `_calculateRewards` function the maturingBalance.interestHistoryIndex is set to 0, this code is redundant and requires extra Gas spending.

Recommendation:

Remove the redundant code.

[STATIC-16]	(initialize) Redundant Access to a Storage Variable	GAS

Details:

In the initialize function the `penaltyRate` is set and validated, however, it is never used.

Therefore, the gas usage is increased.

Recommendation:

Remove the redundant code.

[DYNAMIC-17]	(updateRatesHistory) Redundant Storage Variable Access	GAS

Details:

In the `updateRatesHistory` function the `historyInterestRate`.length is accessed before it is cached to the `historyLength` variable. This leads to the redundant Gas consumption.

Recommendation:

Use the `historyLength` variable.

Detailed Findings - Vesting Contracts

[VEST-01]	IF statement in VestingModule might be mistakenly placed	High

Details:

When removing vestors from a group, there is the following IF statement

```
if (
groupSchedule[groupId][
      groupSchedule[groupId].length - 2
] > block.timestamp
)
```

How would that work in case the schedule stores only how long it is active?

Comparing timestamp with days duration?

Recommendation:

Please ensure this functions according to the requirements, as there may be a mistake in testing.

[VEST-02]	Lack of mechanism to disable oracle in case of compromise	Low

Details:

The current smart contract system does not have a mechanism to deactivate or stop the oracle service in case it becomes compromised. This leaves the system vulnerable to potential security breaches if the oracle provides incorrect or malicious data.

Recommendations:

It is advised to implement an oracle fallback mechanism to protect against oracle compromise.

[VEST-03]	(feeSplitter) Redundant check if sendToAdd3 function has executed correctly	Low

Details:

In the feeSplitter function there is a check if sendToAdd3 function has executed correctly:

```
if (!sentAdd3 || feeToAdd3 == 0) {
            revert Error_No_Fee_Sent_To_Add3();
        }
```

Recommendations:

Let the `sendToAdd3` function deal with that.

[VEST-04]	9000 and 10000 as variables	Low

Recommendations:

Make 9000 and 10000 to be constant variables.

[VEST-05]	(airdropVestedTokens) Redundant Calls to getGroupGrantSchedule	GAS

Details:

The function `getGroupGrantSchedule` is called three times within the `airdropVestedTokens` function. This repetitive calling leads to unnecessary gas consumption and can be optimized.

Recommendations:

It is advisable to call `getGroupGrantSchedule` only once and reuse its result in the airdropVestedTokens function. This change would reduce gas costs and improve the efficiency of the contract.

[VEST-06]	(initialize) Redundant checks if an address is a contract	GAS

Details:

`Initialize` function checks in multiple places if an address is a contract. However, even if the contract is a random one, with incorrect values - the checks will still pass.

Recommendations:

Remove all these checks to decrease code complexity and gas costs.

[VEST-07]	Missing addVestors._recipients and addVestors._amounts arrays lengths equality verification	Informational

Details:

It is not checked in the `onlyAuthorizedAddVestorsToGroup` modifier if `_recipients` and `_amounts` array lengths are equal. This will lead to the unexpected processing of the incorrect input data.

Recommendations:

Verify if `_recipients` and `add_amounts` arrays have the same length.

[VEST-08]	Inconsistent naming in vesting contract	`Informational`

Details:

Discrepancy between file and contract names (File: VestingModule, Contract: Vesting).

Recommendations:

Align the file name with the contract name for clarity and consistency, following best practices. Rename the file to Vesting.sol or the contract to VestingModule.

[VEST-09]	Use OpenZeppelin Ownable contract.	`Informational`

Recommendations:

We recommend you use the OpenZeppelin Ownable contract for implementing ownership in your contracts.

[VEST-10]	Move signatures and validations to a Vesting Auth library	`Informational`

Recommendations:

Contract readability is largely impacted by all signatures and their validations. We recommend moving all of them to a Vesting Auth library.

[VEST-11]	(getEthInputFee) Multiple blockchains logic consideration	`Informational`

Details:

`getEthInputFee` seems to care about multiple blockchains, which increases complexity further. Best practice is for contracts to be small and easy to read.

Recommendations:

Provide `dollarsFee` as an argument in the initialize function and simply use it based on the blockchain. That way you can skip all that unnecessary logic for the different chains.

[VEST-12]	Rename groups in getVestorGroupAmountSum	`Informational`

Recommendations:

Rename groups in `getVestorGroupAmountSum` to recipients, as the current naming can be confusing.

Detailed Findings - All Contracts

[ALL-01]	(feeReturner, feeSplitter) Non-fixed fees	Medium

Details:

The implementation of `feeReturner` and `feeSplitter` in contracts processes the entire balance present in the contract, not refunding excess amounts to users. This handling of non-fixed fees poses a security risk, as it doesn't ensure the return of surplus funds. Additionally, this behavior is not detailed in the existing documentation, leading to potential misunderstandings and misuse of the system.

Recommendations:

It's advised to revise the fee processing logic to ensure correct handling of fees and refunding of excess balances. Updating the documentation to clearly outline these processes is also essential for clarity and security.

[ALL-02]	No functionality to revoke signatures	Low

Details:

The Vesting, StakingModuleStatic, StakingModuleDynamic, and StakingBase contracts heavily rely on delegation logic where a user's signature is required. However, currently, there is no functionality to lock or invalidate a signature if it is no longer relevant or in cases of fraudulent activities. This oversight could potentially lead to unauthorized or unintended actions being carried out, posing a significant security risk.

Recommendations:

Introducing a mechanism to lock or revoke signatures in the delegation process is strongly recommended. This would enhance the security by allowing users to invalidate their signatures in case of any irregularities or when they are no longer applicable.

[ALL-03]	Redundant receive function	`Informational`

Details:

The contracts contain a receive function that automatically refunds any incoming transfers to users. Since all incoming transfers are returned, the receive function appears to be redundant.

Recommendations:

Consider removing the receive function to simplify the contract.

[ALL-04]	Redundant events	`Informational`

Details:

The contracts contain events that are never emitted. Redundant code decreases the code readability and increases the contracts' sizes.

Recommendations:

Remove the redundant code.

[ALL-05]	Redundant errors	Informational

Details:

The contracts contain errors that are never thrown. Redundant code decreases the code readability and increases the contracts' sizes.

Recommendations:

Remove the redundant code.

[ALL-06]	Commented code	Informational

Details:

The contracts contain commented code. Commented code decreases the code readability and may indicate that the code is not finalized.

Recommendations:

Remove the redundant comments.

[ALL-07]	Use of Unspecified uint Sizes in Contracts	Informational

Details:

The contracts use uint variables without specifying their exact size (e.g. uint256). This affects code readability and might lead to inefficient storage use, as the variable may occupy more space than necessary.

Recommendation:

Specify exact sizes for uint variables (e.g. uint256) to improve readability and optimize storage usage.

[ALL-08]	Too many checks in customModifier	`Informational`

Recommendation:

customModifer does too many checks. Split the checks done in customModifer, as currently there are too many.

[ALL-09]	Move all the events, all structs and all errors in the interface	`Informational`

Recommendations:

We recommend for all events, all structs and all errors to be moved in the interface.

Overall Assessment

Severity	Count	Acknowledged	Addressed
Critical	1	Yes	Please refer to Findings Summary for detailed information on fixes introduced.
High	3	Yes	
Medium	2	Yes	
Low	6	Yes	
Informational	18	Yes	
GAS	8	Yes	

Recommendations

Please refer to Findings Summary for detailed information on fixes introduced by Add3.

Audita has put forward the following recommendations for Add3 contracts, and Add3 have implemented the vast majority of them:

Staking Contracts:

- Ensure that `getRewardsValue` function calculates the rewards for the `minLockDay` interval.
- Implement a feature allowing users to set a minimum reward threshold when they stake, unstake, or harvest rewards.
- Allow users to withdraw their funds when the staking is removed from the controller.
- Ensure that implementation matches the requirements described in the documentation.
- Redesign the implementation so that `getLastOutOfLockAndUnstakedDuringLock` does not loop over an unlimited array.
- Revert if the check is not met for max and min stake.
- Fix minLockDays variable name.
- Fix the _harvestAndWithdrawRewards function name.
- For improved safety, we recommend implementing OpenZeppelin's reentrancy guard.
- For clarity and future convenience, we recommend to split the blackOrWhitelistAddress function to `blackList` and `whiteList`.
- There is a !token.isContract() check in the `transferAccidentalyLockedTokens`. Remove this as the owner is calling it and they are required to provide accurate params.

- It is advisable to call `Controller.productReward` only once and reuse its result in the `_previewApy` function.
- Set the `maxStake` and `minStake` variable once after the required verifications.
- Set the minLockDays after the required verification of `_minLockDays`.
- Remove redundant code In the `_calculateRewards` function.
- Remove the redundant code in the `penaltyRate` function.
- In the `updateRatesHistory` function, use the `historyLength` variable.

Vesting Contracts:

- Ensure IF statement in VestingModule functions according to the requirements.
- It is advised to implement an oracle fallback mechanism to protect against oracle compromise.
- Let the `sendToAdd3` function deal with the check in `feeSplitter`.
- Make 9000 and 10000 to be constant variables.
- It is advisable to call `getGroupGrantSchedule` only once and reuse its result in the airdropVestedTokens function.
- Remove checks if an address is a contract to decrease code complexity and gas costs.
- Verify if `_recipients` and `add_amounts` arrays have the same length.
- Align the file name with the contract name for clarity and consistency, following best practices. Rename the file to Vesting.sol or the contract to VestingModule.
- We recommend you use the OpenZeppelin Ownable contract for implementing ownership in your contracts.
- Contract readability is largely impacted by all signatures and their validations - We recommend moving all of them to a Vesting Auth library.
- Provide `dollarsFee` as an argument in the initialize function and simply use it based on the blockchain.

- Rename groups in `getVestorGroupAmountSum` to recipients, as the current naming can be confusing.

All Contracts:

- It's advised to revise the fee processing logic to ensure correct handling of fees and refunding of excess balances.
- Introducing a mechanism to lock or revoke signatures in the delegation process is strongly recommended.
- Consider removing the receive function to simplify the contract.
- Remove the redundant code for events that are never emitted.
- Remove the redundant code for errors that are never thrown.
- Remove the redundant comments in the code, as it may seem the code is unfinished.
- Specify exact sizes for uint variables (e.g. uint256) to improve readability and optimize storage usage.
- Split the checks done in customModifer, as currently there are too many.
- We recommend for all events, all structs and all errors to be moved in the interface.

Fixes

Please refer to Findings Summary for detailed information on fixes introduced by Add3.

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.