VestingTomo Contracts Audit

Khrystyna Tkachuk, Oleksandr Zadorozhnyi

October, 2024

Email: info@4irelabs.com

Website: 4irelabs.com

CONFIDENTIAL

This document and the Code Review it references is strictly private, confidential and personal to its recipients and should not be disclosed, copied, distributed, or reproduced in whole or in part, not passed to any third party.

1. **Introduction.** Tomo requested 4irelabs a review of their contracts.

   There is 1 contract source code in scope:

   - VestingTomo

2. **Warranty.** Audit is provided on an "as is" basis, without warranty of any kind, express or implied. The Auditor does not guarantee that the Code Review will identify all instances of security vulnerabilities or other related issues.

3. **Executive Summary.** The VestingTomo smart contract is designed to manage token vesting for contributors, allowing them to claim tokens according to a predefined vesting schedule. The contract includes functionalities such as pausing, emergency token withdrawal, and staking. The owner can add new vesting schedules and update existing ones. The vesting contract supports ERC20 tokens with different decimals, but the initial values in the vesting schedule should contain 18 decimals otherwise calculation of the vested amount will be broken and return 0. Besides the basic claim of the token (release() function), users can claim and stake a token into the AbsDepositTomoCat contract via the function depositOnlyManager() that is called in releaseAndStake() function of the VestingTomo contract. Function depositOnlyManager() was out of the scope of the current audit.

4. **Critical Bugs and Vulnerabilities.** 2 High issues have been found. More details in points 5.1.20 and 5.1.21.

5. **Line By Line Review.**

5.1. VestingTomo:

5.1.1. Info: Unused Library Imported

The **EnumerableSet** library is imported but not used in the contract.

Recommendation: Remove the unused import statement to reduce bytecode size and improve code readability.

5.1.2. Info: Unused variable

Delete unused variables to decrease the bytecode of the contract.

*Recommendation: Delete variable **stableCoinDecimals.***

5.1.3. Info: Incorrect order of layout

The layout contract elements in the VestingTomo contract are not logically grouped.

Inside the contract use the following order:

- Library declarations (using statements);
- Constant variables;
- Type declarations;
- State variables;
- Events;
- Errors
- Modifiers;
- Functions.

Functions should be grouped according to their visibility and ordered in the following way:

- constructor;
- receive function (if exists);
- fallback function (if exists);
- external;
- public;
- internal;
- private.

Recommendation: Consider changing the layout and functions order according to solidity style guide documentation.

5.1.4. Info: Functions visibility optimization

Instead of marking a function as **public**, consider marking it as **external** if it is not used internally.

Recommendation: Consider changing visibility from public to external: addDataToParticipation, emergencyWithdrawTokens, pause, unpause, startVesting, release, releaseAndStake, setAllowedAddress.

5.1.5. Info: Use error instead requires

The contract contains both errors and requires. Consider using errors that are more efficient in gas usage and to keep code consistent.

Recommendation: Consider changing all requires to dedicated custom errors.

5.1.6. Info: Unused error

There are some errors that are declared but never used.

Recommendation: Consider using the declared error or delete them to reduce bytecode of the contract: Disabled, VestingHasStarted, NotStartVesting.

5.1.7. Info: The wrong timestamp is emitted in the StartVesting() event or the event name is misleading

During execution of the startVesting() function contract calculates and stores the vestingStart variable; however, the event in this function emits the vestingTime variable. This could lead to confusion while reading event logs.

Recommendation: Consider changing the event name to one that would correspond better to the vestingTime variable or emit the vestingStart value instead.

5.1.8. Info: vestedAmount() function could revert without a meaningful message in case if user has no allocation

Internally called _vestingSchedule() function would try to calculate modulo by zero at L210 for a user that has no allocation (equal to zero) which would lead to revert without a correct error message.

Recommendation: Consider adding a check for zero allocation into the vestedAmount() function and reverting with an error message in such a case.

5.1.9 Low: Floating pragma

The current version of solc in contract VestingTomo is **^0.8.7** which allows the code to be compiled with any compiler version from **0.8.7** up to (but not including) **0.9.0**. This can lead to inconsistencies due to changes in the compiler's behavior in future versions.

Contracts should be deployed with the same compiler version and flags they have tested thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Recommendation: Lock pragma to a specific version.

5.1.10. Low: Unused Struct Field amount

In the **ContributionData** struct, the **amount** field is not used in any calculations or logic within the contract.

*Recommendation: Remove the unused **amount** field to simplify the struct.*

5.1.11. Low: State variable could be declared constant

There are variables that have not changed during deployment and further. Declare such variable constants to save the gas for reading such variables.

*Recommendation: Declare **EIGHTEEN_DECIMALS** variable constant.*

5.1.12. Low: State variable could be declared immutable

There are some variables that only changed in the constructor. Declare such variables immutable to the same gas usage.

Recommendation: Declare next variables immutable:

- *timeLockSeconds, vestingDurationSeconds, vestingWidthdrawInterval, token.*

Remove default initialization and declare the next variables immutable:

- *tgeReleasePercent, tokenReleaseDecimals.*

5.1.13. Low: Constructor optimization

Consider putting checks at the top of the constructor and using params variables for comparison that will save some gas usage.

Recommendation: Move checks to the top and avoid using storage variables to compare.

5.1.14. Low: Loop optimization

To decrease gas usage in the loop execution avoid initialization of the iterator since uint256 by default is 0, use pre-increment (++i) instead of post increment (i++), add length into separate memory function outside the loop, and use that in the loop.

Recommendation: Consider loop optimization in ***addDataToParticipation*** *function. The optimized loop example:*

```
uint256 length = contributionsUsers.length;
for (uint256 i; i < length; ) {
  ....
    unchecked { ++i; }
}
```

5.1.15. Low: Useless require statement in releaseAndStake

In function releaseAndStake there are two same requires that check if the contract has insufficient balance. They are duplicated. In fact, this case already checked in the safeTransfer function from Openzeppelin library.

Recommendation: Delete useless duplicated requires.

5.1.16. Low: Lack of zero address check

There is no check if the passed addresses aren't zero value In the constructor and the function setAllowedAddress() of VestingTomo contract.

Recommendation: Consider adding checks for zero address.

5.1.17. Low: Event is missing indexed fields

Index event fields make the field more quickly accessible to off-chain tools that parse events.

Recommendation: Consider adding an indexed field to each event.

5.1.18. Low: tgeReleasePercent variable allowed to be equal to 100% while should not

The constructor reverts if the TGE percent is greater than 100%; however, it allows it to be equal to this value, while the comment and contract logic stays opposite.

Recommendation: Consider updating the check at L102 to disallow 100% value.

5.1.19. Low: _vestingSchedule() function could return different values for the same timestamp and allocation if it's called before or after vestingTime timestamp due to check at L192

However, this function intends to return the vested amount based on the provided timestamp, regardless of whether it's future or past or what the current block timestamp is, so its result should not depend on the moment of execution.

Recommendation: Consider updating the _vestingSchedule() function logic so it always uses the provided timestamp value

instead of the current block timestamp. This way users would be able to calculate their vested amount for the future timestamps before vestingTime.

5.1.20. High: Owner can rewrite data of the existing vesting

Once new vesting data is added there is no check if the address exists. It may lead to the loss of the vested tokens by the end user since the owner of the contract can change that value any time.

Recommendation: Do not allow owner to rewrite existed vestings.

5.1.21. High: Owner can rewrite value of released token

Owner can rewrite the value of the resealed token for the existing vestings. It may lead to loss of the token by the user.

Released tokens should be written once the user claim them in the release and releaseAndStake functions.

Recommendation: Delete initializing released token from the function addDataToParticipation.