Enzym - ZYM Token Vesting

Audit Report

Version 1.2

Zigtur

June 10, 2024

Enzym - ZYM Token Vesting - Audit Report

Zigtur

June 10, 2024

Prepared by: Zigtur & nisedo

Table of Contents

- Table of Contents
- Introduction
  - Disclaimer
  - About Zigtur
  - About ZYM Token Vesting
- Security Assessment Summary
  - Deployment chain
  - Scope
  - Risk Classification
- Issues
  - INFO-01 - ZYM Token Can Be Hardcoded if Vesting is Meant to Be Used Only with ZYM
  - INFO-02 - Vesting Storage is Not Aligned on 50 Slots
  - INFO-03 - `startTimestamp` Can Be in the Past
  - INFO-04 - `durationSeconds` Can Be Zero or Very Large
  - INFO-05 - Users Don't Control Their Token Release
  - INFO-06 - `VestingFactory::createVesting` Should Revert on Zero Address for Beneficiary to Avoid Loss of Funds
  - INFO-07 - `VestingFactory::claimable` May Revert with Out of Gas Error

  - INFO-08 - `VestingFactory::withdraw` Should Revert on Zero Amount Transfer and Zero Address Recipient
  - INFO-09 - `VestingWallet::receive` and `VestingWalletUpgradeable::receive` Functions Can Be Removed
  - INFO-10 - Number of Vesting Contracts per User is Not Bounded
- Appendix
  - INFO-10 - Proof of Concept

Introduction

Disclaimer

A smart contract security review cannot guarantee the complete absence of vulnerabilities. This effort, bound by time, resources, and expertise, aims to identify as many security issues as possible. However, there is no assurance of 100% security post-review, nor is there a guarantee that the review will uncover all potential problems in the smart contracts. It is highly recommended to conduct subsequent security reviews, implement bug bounty programs, and perform on-chain monitoring.

About Zigtur

Zigtur is an independent blockchain security researcher dedicated to enhancing the security of the blockchain ecosystem. With a history of identifying numerous security vulnerabilities across various protocols in public audit contests and private audits, **Zigtur** strives to contribute to the safety and reliability of blockchain projects through meticulous security research and reviews. Explore previous work here or reach out on X @zigtur.

About ZYM Token Vesting

The ZYM Token Vesting system is designed to manage the vesting and distribution of the ZYM ERC20 token. It includes contracts that handle the allocation, release, and management of vested tokens, ensuring that beneficiaries receive their tokens according to a predetermined schedule.

Security Assessment Summary

Review commit hash - 4336d811e77ba95c6cca01ef7d37e9eae664910e

Fixes review commit hash - N/A

Deployment chain

- Ethereum Mainnet

Scope

The following smart contracts are in scope of the review:

- contracts/reserve/Reserve.sol
- contracts/token/extensions/ERC1363.sol
- contracts/token/ZYM.sol
- contracts/vendor/@amxx/hre/contracts/Balances.sol
- contracts/vendor/@amxx/hre/contracts/Distributions.sol
- contracts/vesting/AllocatedVesting.sol
- contracts/vesting/Vesting.sol
- contracts/vesting/VestingFactory.sol

Risk Classification

	Impact: High	**Impact:** Medium	**Impact:** Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Issues

INFO-01 - ZYM Token Can Be Hardcoded if Vesting is Meant to Be Used Only with ZYM

If the vesting system is intended to be used exclusively with the ZYM token, the functions `allocate()` and `release()` could be hardcoded to use the ZYM token. This change would simplify the code and potentially reduce the risk of incorrect token allocations.

All functions concerned:

- `AllocatedVesting::allocated`
- `AllocatedVesting::totalAllocated`
- `AllocatedVesting::allocationRemaining`
- `AllocatedVesting::released`
- `AllocatedVesting::totalReleased`
- `AllocatedVesting::releasable`
- `AllocatedVesting::vestedAmount`
- `AllocatedVesting::allocate`
- `AllocatedVesting::release`

Resolution

Enzym team: Acknowledged. It's by-design, to allow the system to be used with other ERC20 tokens than the ZYM token.

INFO-02 - Vesting Storage is Not Aligned on 50 Slots

The `Vesting` contract, due to its inheritance from `VestingWalletUpgradeable` from OpenZeppelin, does not align storage on 50 slots. As a result, the first variable of `Vesting` will occupy slot `153`, which might be unexpected and can affect storage optimization.

Run `forge inspect Vesting storage --pretty` to see the storage layout.

Name	Type	Slot	Contract
_initialized	bool	0	Vesting
_initializing	bool	0	Vesting
__gap	uint256[50]	1	Vesting
_released	uint256	51	Vesting

Name	Type	Slot	Contract
_erc20Released	mapping(address => uint256)	52	Vesting
_beneficiary	address	53	Vesting
_start	uint64	53	Vesting
_duration	uint64	54	Vesting
__gap	uint256[48]	55	Vesting
_owner	address	103	Vesting
__gap	uint256[49]	104	Vesting
_start	uint64	**153**	Vesting
_duration	uint64	153	Vesting

Resolution

Enzym team: Acknowledged. No impact if upgrades are correctly done.

INFO-03 - `startTimestamp` Can Be in the Past

In both the `AllocatedVesting` and `Vesting` contracts, the `startTimestamp` for vesting can be set to a time in the past.

It is recommended to add a check to ensure that the `startTimestamp` is in the future and at a reasonable distance from the current time.

Resolution

Enzym team: Acknowledged. It's by design, to make it possible to apply a vesting plan after its legal start-up in case of oversight.

INFO-04 - `durationSeconds` Can Be Zero or Very Large

In both the `AllocatedVesting` and `Vesting` contracts, the `durationSeconds` for vesting can be set to 0 or a very large value (up to `type(uint64).max` == `18446744073709551615` == in 530 years).

It is recommended to add a check to ensure that the `durationSeconds` is greater than 0 and at a reasonable value.

Resolution

Enzym team: Acknowledged. “0” is a feature to release everything at once into the future (as an escrow). “Very large” is to be able not to choose an arbitrary value that could be either too short or too long.

INFO-05 - Users Don’t Control Their Token Release

The `VestingFactory::claimAll`, `AllocatedVesting::release` and `VestingWalletUpgradeable::release` functions can be called by any address, not just the beneficiary. While this is not a security issue, users might prefer to control when their tokens are released to manage their finances more effectively.

It is recommended to add a check to ensure that only the beneficiary can call the `claimAll` and `release` functions.

Resolution

Enzym team: Acknowledged. It’s a feature for users to use persistent addresses that can receive tokens but not call functions.

INFO-06 - `VestingFactory::createVesting` Should Revert on Zero Address for Beneficiary to Avoid Loss of Funds

The `createVesting` function in `VestingFactory` does not currently check if the beneficiary address is zero. This could lead to an irreversible loss of funds if tokens are vested to the zero address.

It is recommended to add a check to ensure that the beneficiary address is not zero before creating a new vesting contract.

Resolution

Enzym team: Acknowledged.

INFO-07 - `VestingFactory::claimable` May Revert with Out of Gas Error

The `claimable` function in `VestingFactory` iterates over all vesting instances for a user. If the user has a large number of vesting contracts, this function could potentially run out of gas. It is not a concern in the current implementation as the `claimable` function is a view function, but it could be if it’s ever used in a public state-changing context.

Resolution

Enzym team: Acknowledged.

INFO-08 - `VestingFactory::withdraw` Should Revert on Zero Amount Transfer and Zero Address Recipient

The `withdraw` function in `VestingFactory` does not currently check for zero amount transfers or zero address recipients.

Adding these checks would prevent potential misuses and ensure more robust handling of withdrawals.

Resolution

Enzym team: Acknowledged. We follow the underlying ERC20 token policy which authorizes or not 0 transfers.

INFO-09 - `VestingWallet::receive` and `VestingWalletUpgradeable::receive` Functions Can Be Removed

The `receive` functions in both `VestingWallet` and `VestingWalletUpgradeable` are not used since the system is meant to be used only with ZYM tokens and could lead to the contract receiving ETH by mistake.

Removing these functions would prevent accidental ETH deposits.

Resolution

Enzym team: Acknowledged. It's by-design, to allow the system to be used with other ERC20 as well as native tokens.

INFO-10 - Number of Vesting Contracts per User is Not Bounded

Description

There is no limit on the number of vesting contracts a single user can have. This could lead to the `VestingFactory::claimAll()` function to revert with an out-of-gas error if a user has a too large number of vesting contracts, thus preventing them from claiming their tokens through the factory.

According to the PoC provided in Appendix, the function would revert with an out-of-gas error if the user has more than 936 vesting contracts. This seems very unlikely to happen in practice, but it is recommended to cap the number of vesting contracts per user in the `VestingFactory` contract.

Resolution

Enzym team: Acknowledged. This is a helper function and not a critical path. A user with that many vestings can still retrieve their tokens by calling the `release` function on each vesting contract individually.

Appendix

INFO-10 - Proof of Concept

See Proof of Code

1. Add Foundry to the Hardhat project: https://book.getfoundry.sh/config/hardhat#adding-foundry-to-a-hardhat-project
2. Add the code below to a `test/VestingFactoryTest.t.sol` file
3. Run `forge test -vvvv`

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "forge-std/Test.sol";
import "contracts/vesting/VestingFactory.sol";
import "contracts/vesting/Vesting.sol";
import "@openzeppelin/contracts-4.5.0/token/ERC20/ERC20.sol";

contract MockERC20 is ERC20 {
    constructor() ERC20("Mock Token", "MTK") {
        _mint(msg.sender, type(uint256).max);
    }
}

contract VestingFactoryTest is Test {
    VestingFactory factory;
    MockERC20 token;
    address admin = address(1);
    address beneficiary = address(2);
    uint256 gasBlockLimit = 30_000_000;

    function setUp() public {
        token = new MockERC20();
        factory = new VestingFactory(IERC20(token), admin);
        token.transfer(address(factory), type(uint256).max);
    }

    function testClaimAll() public {
        vm.startPrank(admin);

        // Create vesting instances
        for (uint256 i = 0; i < 936; i++) { //@audit 935 is ok, 936 reverts with out-of-gas error
            factory.createVesting(
                0.000000000000000001 ether, // 1 wei
                beneficiary,
                uint64(block.timestamp),
                365 days,
```

```
38                  "Vesting[i]");
39          }
40
41          // Fast forward time to make some tokens claimable
42          vm.warp(block.timestamp + 366 days);
43
44          // Check initial balances
45          uint256 initialBalance = token.balanceOf(beneficiary);
46
47          // Claim all vested tokens
48          vm.startPrank(beneficiary);
49
50          uint256 gasUsed = gasleft();
51          factory.claimAll(newOwner);
52          gasUsed = gasUsed - gasleft();
53
54          // Check final balances
55          uint256 finalBalance = token.balanceOf(beneficiary);
56          assertGt(finalBalance, initialBalance, "Tokens were not claimed
                correctly");
57
58          assertLt(gasUsed, gasBlockLimit, "Gas usage exceeds block limit
                ");
59      }
60  }
```