OpenZeppelin

Ocean Vesting Wallet Audit

OPENZEPPELIN SECURITY | OCTOBER 12, 2023

Security Audits

Table of Contents

Summary

Type	DeFi
Timeline	From 2023-09-19 To 2023-09-21
Languages	Solidity
Total Issues	12 (8 resolved, 1 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	5 (3 resolved)
Low Severity Issues	2 (1 resolved)
Notes & Additional Information	5 (4 resolved, 1 partially resolved)

Scope

We audited the oceanprotocl/vw-cli repository at commit 0397c74.

```
contracts/
├── Splitter.sol
└── VestingWalletHalving.sol
```

System Overview

from the initial multisig wallet to the `[illegible]` contracts and their respective stakeholders.

In a one-time transaction, Ocean tokens are intended to be transferred to a collection of vesting contracts with various vesting schedules. Over time, these vesting wallets will release funds into the payment splitter which will apportion the payment across the `DF Rewards` contracts.

The `VestingWalletHalving` contract is an implementation of the vesting wallet functionality of the protocol, based on a half-life formula. Its source code is substantially similar to the OpenZeppelin implementation of an abstract `VestingWallet`.

The `Splitter` contract implements the payment splitter functionality of the protocol. Its source code again hews closely to the OpenZeppelin implementation.

Security Model and Privileged Roles

When compared to their corresponding implementations by OpenZeppelin, both of these contracts have additional administrative functionalities accessible by their owners.

The owner of the `Splitter` contract can:

1. Add a payee at any point in time.
2. Remove a payee at any point in time.
3. Alter the shares of any payee at any point in time.

The owner of the `VestingWalletHalving` contract can:

1. Change the beneficiary address of the wallet at any point in time.
2. Remove all tokens from the contract at any point in time.

The administrative functionality creates substantial risk for the `payees` of the `Splitter` and `beneficiary` of the `VestingWalletHalving` contracts when compared to the immutable and irrevocable implementations.

The development team has indicated that these administrative features are intended as contingencies and that ownership of both is to be renounced at some future point in time. For this trusting counterparties. Cashflows into the `DF Rewards` contracts cannot, therefore, be considered trustless until ownership of the above contracts is renounced.

Medium Severity

Insufficient Input Validation for Vesting Schedule

The input arguments of the `constructor` are not sufficiently validated. Some problematic scenarios will arise from this, including:

- `startTimestamp` can be a timestamp that has already passed or is too far away in the future.
- `halfLife` can be zero, causing the `getAmount function` to always revert.
- `duration` can be zero, allowing a user to instantly release all allocated tokens and Ether.

Consider adding sensible lower and upper bounds for all arguments of the constructor to ensure none of the outlined scenarios occur.

***Update:** Resolved in pull request #42 at commit a009de2.*

Dust Is Paid to Last Payee in the `_payees` Array

Line 117 sets the `payment` variable to the remaining balance for the last address in the `_payees` array. While this keeps the rounded-down wei from remaining in the contract after a single payment, it also creates a fairness issue from the point of view of the other `_payees`.

Consider paying all payees `balance * _shares[payee] / _totalShares`.

***Update:** Resolved in pull request #44 at commit 822347b.*

Owner of the `Splitter` Contract Can Dilute Existing Shareholders

The `addPayee` function of the `Splitter` contract allows the owner of the contract to unilaterally mint new shares to a new address. When unclaimed funds are present within the splitter, the existing shareholders are diluted by the administrative action.

When unclaimed funds are present within the splitter, the same holds true for the `adjustShare` function.

administrative actions.

Update: *Acknowledged, not resolved. The Ocean Protocol team stated:*

> *The existing shareholders are diluted by the administrative action. This aligns with our expectations, we would like the alterations to the shares to effectively apply to the unclaimed amount.*

`VestingWalletHalving` Administrative Functions Ignore Previously Released Tokens

The `renounceVesting` function in `VestingWalletHalving.sol` allows the owner to renounce the `_beneficiary`'s vesting amount. This will transfer the entire balance for any token of the contract and would include those that haven't been released and transferred to the `_beneficiary`. In the case of Ocean Protocol, the `_beneficiary` will be the `splitter.sol`, which will then release tokens to the respective incentive programs.

For example, if a long time has passed and the `release` function has never been called, there would be a large amount of tokens owed to the incentive programs. However, if an owner were to call `renounceVesting`, the incentive programs would not get paid what they are owed at that point in time.

The same can be said for `changeBeneficiary`, since the prior incentive programs will not be paid what they are owed before it is changed.

Consider an architecture and deployment strategy where these administrative functions are not necessary. At a minimum, consider changing `renounceVesting` and `changeBeneificiary` to first transfer vested tokens to the incentive program and only the remaining as a refund to the owner.

Update: *Acknowledged, not resolved. The Ocean Protocol team stated:*

> *The contract is not obligated to account for unpaid incentive programs and is designed to promptly return all funds back to the owner, so this is ok.*

`renounceVesting` Does Not Return ETH

The contract also has a function called `renounceVesting` used by the owner to return a specific ERC-20 token and renounce vesting.

However, the function does not allow the owner to have any ETH returned upon renouncing. This would leave the ETH stuck in the contract until the end of the given schedule period.

Consider adding a variant implementation of renounceVesting that allows the owner to recover ETH.

***Update:** Resolved in pull request #40 at commit 543fb5c.*

Low Severity

Missing Docstrings

Throughout the codebase, there are several parts that do not have docstrings. For instance:

- Line 210 of `VestingWalletHalving.sol`
- Line 217 of `VestingWalletHalving.sol`

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

***Update:** Resolved in pull request #53 at commit 11bbd7a.*

The Caller of `release` Will Pay Gas Fees for All Payees

The release function of the `Splitter` contract iterates over the array of payees in order to disperse payments to each payee. This creates a fairness issue in the scenario where a `payee` is intended to call `release`.

Consider allowing each payee to release their own payment.

***Update:** Acknowledged, not resolved. The Ocean Protocol team stated:*

Non-Explicit Imports Are Used

The use of non-explicit imports in the codebase can decrease the clarity of the code, and may create naming conflicts between locally defined and imported variables. This is particularly relevant when multiple contracts exist within the same Solidity files or when inheritance chains are long.

Throughout the codebase, global imports are used:

- Line 5 of `VestingWalletHalving.sol`
- Line 6 of `VestingWalletHalving.sol`
- Line 7 of `VestingWalletHalving.sol`
- Line 8 of `VestingWalletHalving.sol`

Following the principle that clearer code is better code, consider using named import syntax (`import {A, B, C} from "X"`) to explicitly declare which contracts are being imported.

***Update:** Resolved in pull request #55 at commit 55a3e07.*

Lack of Indexed Event Parameters

Throughout the codebase, several events do not have their parameters indexed. For instance:

- Line 20 of `Splitter.sol`
- Line 27 of `Splitter.sol`
- Line 30 of `Splitter.sol`

Consider indexing event parameters to improve the ability of off-chain services to search and filter for specific events.

***Update:** Resolved in pull request #50 at commit c6d6f5a.*

`constructor` Is Payable But `receive` Raises an Error

Consider removing the `payable` modifier from the `constructor` and deleting the `receive` function.

Update: *Resolved in pull request #48 at commit 49a845b.*

Unnecessary `require` Checks in `release` Functions

The `release()` and `release(address)` functions of the `VestingWalletHalving` contract both contain a `require` check that asserts that the beneficiary of the wallet is not the zero address. These checks are unnecessary since both the `constructor` function and the setter that mutates the `_beneficiary` variable contain the check as well, ensuring that the variable can never be the zero address.

Consider removing the unnecessary `require` checks.

To avoid legal issues regarding copyright and follow best practices, consider adding SPDX license identifiers to files as suggested by the Solidity documentation.

Update: *Resolved in pull request #46 at commit 1695bbf.*

`VestingWalletHalving` Currently Does Not Support All ERC-20 Tokens

The current NatSpec of the `VestingWalletHalving.sol` contract states:

> *This contract handles the vesting of ETH and ERC-20 tokens for a given beneficiary.*

This implies that the vesting wallet supports all ERC-20 tokens. This is not the case for some weird ERC-20 tokens, which could cause some issues. Although the contract will only be used with ETH and Ocean tokens, the NatSpec should not state that it supports all ERC-20 tokens.

Consider changing the NatSpec to be more specific about the types of tokens it will be handling.

Update: *Resolved in pull request #58*

Conclusion

behavior when used with funds present in the contract, and turn what was originally an untrusted relationship between parties into a trusted one.

While the desire to de-risk large protocol changes by creating a pathway for token recovery is understandable, it signals uncertainty around the larger-scale architecture of the protocol and sits in direct conflict with the stated goal of moving towards decentralization. It is our view that, ultimately, this strategy does little to de-risk the transition.

Consider two deployment approaches: one in which funds are fully committed to a vesting contract without ownership features and another in which funds are sent to the vesting contracts but ownership control is retained until a future date. Once the future date is reached and the ownership is renounced, the second situation reduces to the first as funds are now irrevocably committed to the contract. Additionally, it could be argued that the time period in which ownership is retained places the protocol at greater risk by increasing the attack surface.

We therefore advise exploring alternative deployment approaches that may alleviate the perceived need for the administrative features of the contract.

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users'

OpenBrush Contracts Library Security Review

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-

Security Audits

Security Audits

Security Audits