QuillAudits

AUDIT REPORT

February, 2025

For

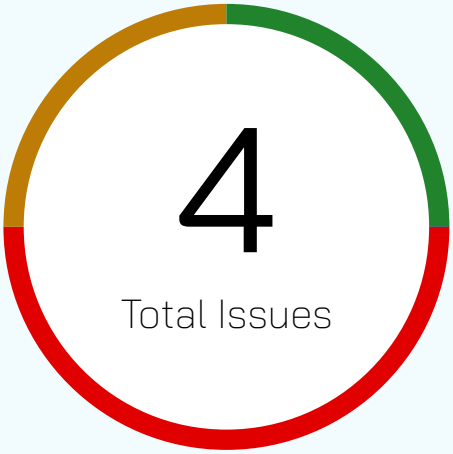
Table of Content

Executive Summary

Project name	Ozolio
Overview	Ozolio's contract manages vesting schedules for different wallets (Team, Advisory, Partner, Treasury), allocating tokens from an ERC20 token contract. It utilizes an exponential decay formula to calculate claimable amounts over time, factoring in lock-up periods and vesting durations. Wallets can claim their vested tokens, and the contract includes functionality for the owner to update wallet addresses and retrieve wallet information, with upgradeability and reentrancy protection.
Project URL	https://qa.ozoliotoken.io/
Audit Scope	The scope of this Audit was to analyze the Ozolio Smart Contracts for quality, security, and correctness. https://github.com/rahul-ray30/Ozolio-SmartContract/commit/e590a1fab0c54ba6233df131a511821f8c390847
Branch & Commit Hash	Branch: audit e590a1fab0c54ba6233df131a511821f8c390847
Contracts under scope	BasicMetaTransaction.sol OwnedUpgradeabilityProxy.sol Ozolio.sol VestingOzolio.sol
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Language	Solidity

Blockchain	Ethereum
Timeline	10th February 2025 - 17th February 2025
Updated Code Received	20th February 2025
Review 2	20th February 2025
Fixed in	Branch: audit 8acacce3f8cce70f18ad6770444dc7cd590c179c

Number of Issues per Severity



High	2 (50.00%)
Medium	1 (25.00%)
Low	1 (25.00%)
Informational	0 (0.00%)

Severity

Issues	High	Medium	Low	Informational
Open	0	0	0	0
Resolved	**2**	**1**	**1**	0
Acknowledged	0	0	0	0
Partially Resolved	0	0	0	0

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Use of tx.origin
- Transaction-Ordering Dependence
- Balance Equality
- Byte Array
- Compiler Version Not Fixed
- Send Instead of Transfer
- Style Guide Violation
- Transfer Forwards All Gas
- Upgradeable Safety
- Using Delegatecall
- Revert/Require Functions
- Unchecked External Call
- Return Values of Low-Level Calls
- Tautology or Contradiction
- ERC's Conformance
- Divide Before Multiply
- Malicious Libraries
- Race Conditions/Front Running
- Arithmetic Computations Correctness
- Logical Issues and Flaws
- Improper or Missing Events
- Centralization of Control
- Access Management
- Implicit Visibility Level
- Unchecked Math
- Missing Zero Address Validation

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools And Platforms Used For Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistical analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational Issues

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

Security vulnerabilities identified that must be resolved and are currently unresolved.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/ impact of the security issue, but are not completely resolved.

High Severity Issues

The nonce increment mechanism is improperly implemented

Resolved

Path

BasicMetaTransaction.sol

Function

executeMetaTransaction

Description

The nonce increment mechanism is improperly implemented, resulting in nonces never being incremented. This completely breaks the replay protection mechanism of the meta-transaction system.

The vulnerability stems from the misuse of the post-increment operator (++). In Solidity, when using post-increment, the following sequence occurs:

The current value is returned
The value is incremented
The returned value (pre-increment) is assigned back to the variable
Therefore:
If nonce starts at 0
nonces[userAddress]++ returns 0 and increments to 1
nonces[userAddress] = then assigns 0 back
The nonce effectively never changes

Recommendation

nonces[userAddress] = ++nonces[userAddress];

Some funds are stuck due to division before multiplication

Resolved

Path

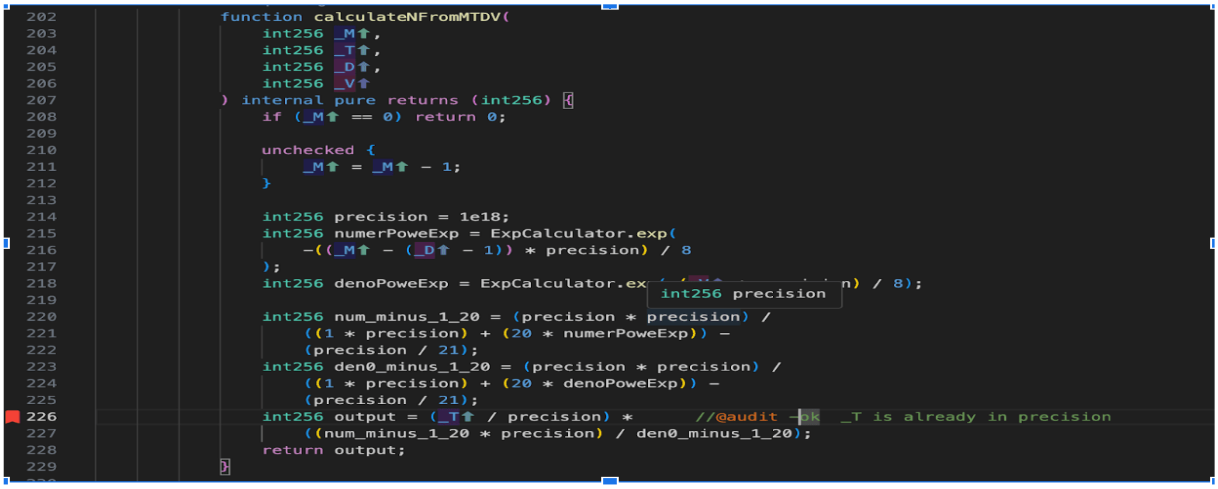
VestingOzolio.sol

Function

CalculateNFromMTDV

Description

The wallets tokenAllocation "_T" is first divided before being multiplied, this will lead to the allocation being rounded down before being multiplied



```
202         function calculateNFromMTDV(
203             int256 _M,
204             int256 _T,
205             int256 _D,
206             int256 _V
207         ) internal pure returns (int256) {
208             if (_M == 0) return 0;
209
210             unchecked {
211                 _M = _M - 1;
212             }
213
214             int256 precision = 1e18;
215             int256 numerPoweExp = ExpCalculator.exp(
216                 -((_M - (_D - 1)) * precision) / 8
217             );
218             int256 denoPoweExp = ExpCalculator.ex                    n) / 8);
219                                                   int256 precision
220             int256 num_minus_1_20 = (precision * precision) /
221                 ((1 * precision) + (20 * numerPoweExp)) -
222                 (precision / 21);
223             int256 den0_minus_1_20 = (precision * precision) /
224                 ((1 * precision) + (20 * denoPoweExp)) -
225                 (precision / 21);
226             int256 output = (_T / precision) *      //@audit ok  _T is already in precision
227                 ((num_minus_1_20 * precision) / den0_minus_1_20);
228             return output;
229         }
```

Recommendation

int256 output = (_T * ((num_minus_1_20 * precision) / den0_minus_1_20)) / precision;
return output;

Medium Severity Issues

Uses Ownable2StepUpgradeable instead of Ownable-Upgradeable

Resolved

Path

VestingOzolio.sol Ozolio.sol

Function

initialize()

Description

One issue is that the contract does not utilize the __Ownable2StepUpgradeable_init() function, designed to facilitate a two-step ownership transfer process. Without this mechanism, ownership transfers occur immediately, exposing the contract to potential risks. The two-step process enhances security by requiring a confirmation step before ownership is fully transferred, thereby reducing the likelihood of unauthorized access or control over the contract.

Recommendation

Use Ownable2StepUpgradeable library.

Low Severity Issues

Enhance Contract Robustness and Security

Resolved

Path

VestingOzolio.sol Ozolio.sol

Description

Add the sanity checks and best practices that make the contract more robust
SafeERC20: Replaced direct token.transfer() with token.safeTransfer() to prevent issues with tokens that don't return true on successful transfers.
Non-Negative N Check: Added a require(N >= 0, ...) check before converting N to a uint256 in claimableAmount() to prevent unexpected behavior if the vesting calculation results in a negative claimable amount.

Recommendation

Use SafeERC20 and,
Added a require(N >= 0, ...) check before converting N

Functional Tests

Some of the tests performed are mentioned below:

- Tested only the owner can mint Ozolio tokens.
- Tested Burning Mechanism.
- Tested only the authorized addresses can call claim().
- Tested Token totalSupply after the minting.

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of Ozolio. We performed our audit according to the procedure described above.

Some issues of High, medium and Low severity were found. Some suggestions, gas optimizations and best practices are also provided in order to improve the code quality and security posture. In the End Ozolio Team,Resolved all issues.

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem

QuillAudits

7+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

February, 2025

For

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com