QuillAudits

Audit Report
April, 2024

For

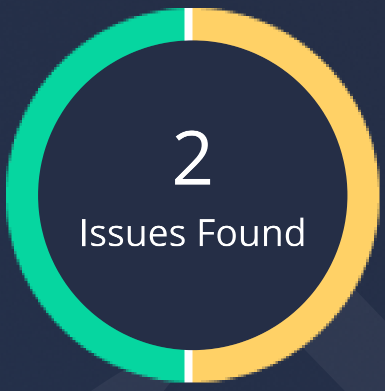
Table of Content

Executive Summary

Project Name	Olive Network
Overview	Olive Network's deposit repository contains contracts that allow users to deposit ETH or pre-approved tokens into the Strategy contracts. These deposited tokens count towards the point-based airdrop allocation and can be withdrawn at any time with no fees for deposits or withdrawals.
Timeline	15th March, 2024 - 19th April, 2024
Updated Code Received	29th April, 2024
Second Review	7539f362704bc1a6db477df1a6f49ae60fd1b7d5
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	The scope of this audit was to analyse the Olive Network codebase for quality, security, and correctness.
Source Code	*https://github.com/Olive-Network/deposits*
Contracts In-Scope	Strategy Folder: - contracts/strategy/IStrategyETH.sol - contracts/strategy/StrategyETH.sol - contracts/strategy/IStrategy.sol - contracts/strategy/Strategy.sol Administrator Folder: - contracts/administrator/Administrator.sol - contracts/administrator/IPausable.sol - contracts/administrator/IRole.sol - contracts/administrator/IBlackList.sol
Branch	main

Contracts Out of Scope	In-scope contracts have been audited by QuillAudits. However, these contracts inherit functionality from out-of-scope Smart contracts that were not audited. Vulnerabilities in unaudited contracts could impact in-scope Smart Contracts functionality. QuillAudits is not responsible for such vulnerabilities. Below are Out of Scope Contracts: - mock/Token.sol - OpenZeppelin contracts: Initializable.sol, ReentrancyGuardUpgradeable.sol, SafeERC20.sol
Fixed In	*https://github.com/Olive-Network/deposits/commit/7539f362704bc1a6db477df1a6f49ae60fd1b7d5*

Number of Security Issues per Severity



🟥 High 🟨 Medium

🟩 Low 🟪 Informational

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

Checked Vulnerabilities

✓ Re-entrancy	✓ Transfer forwards all gas
✓ Timestamp Dependence	✓ ERC20 API violation
✓ Gas Limit and Loops	✓ Compiler version not fixed
✓ DoS with Block Gas Limit	✓ Redundant fallback function
✓ Transaction-Ordering Dependence	✓ Send instead of transfer
✓ Use of tx.origin	✓ Style guide violation
✓ Exception disorder	✓ Unchecked external call
✓ Gasless send	✓ Unchecked math
✓ Balance equality	✓ Unsafe type inference
✓ Byte array	✓ Implicit visibility level

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's 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

Manual Review, Slither, Hardhat.

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

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

These are the issues identified in the initial audit and have been successfully fixed.

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

No issues were found.

Medium Severity Issues

1. Missing gap variables could cause storage collisions when upgrades occur

Path

StrategyETH.sol, Strategy.sol, Administrator.sol

Description

Upgradeable contracts allow for smart contracts to have extended functionality. The current codebase has the **harvest()** functions not implemented. They can be deployed as is, and upgraded at a later time to adjust the implementation. Within composable upgradeable contracts, the issue of storage collision exists which could alter the reading of storage for each of the contracts built atop each other.

To avoid issues when reading or writing to storage it is advisable to have a gap introduced in each contract to prevent the new storage variables added from crossing into existing storage locations. Here, OpenZeppelin describes in detail how to manage smart contract storage with gaps: Writing Upgradeable Contracts - OpenZeppelin Docs

Recommendation

Include storage gaps at the end of state variable declarations in each of the contracts affected.

Status

Resolved

Low Severity Issues

2. StrategyETH::rescue() can cause transactions to complete even with a silent uncaught failure

Path

StrategyETH.sol

Function

L115: StrategyETH::rescue()

Description

The vanilla ERC20 transfer implementation returns a boolean value that indicates success or failure of the operation. This return value is stored in memory and a require check is performed on the value to provide details of success or failure on the finality of the operation. **StrategyETH::rescue()** does not include this check on the transfer function, and as such allows the function to run to completion even if the returned bool is false.

Recommendation

Just like in Strategy.sol, use the SafeERC20 wrapper to handle token transfers for special case ERC20 tokens.

Status

Resolved

Informational Issues

No issues were found.

Functional Tests Cases

- ✓ Should rescue tokens that are not the StrategyToken
- ✓ Should withdraw deposited tokens
- ✓ Should prevent blacklisted addresses from depositing tokens

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 the Olive Network codebase. We performed our audit according to the procedure described above.

Some issues of Medium and Low severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.
In the End,Oliver Network Team,resolved all Issues.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Olive Network smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Olive Network smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Olive Network to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

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.

1000+
Audits Completed

$30B
Secured

1M
Lines of Code Audited

Follow Our Journey

Audit Report
April, 2024

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com