QuillAudits

AUDIT REPORT

April , 2025

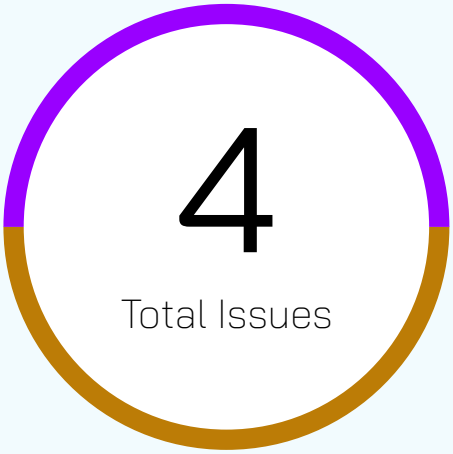
For

Table of Content

Executive Summary

Project name	Finger Tips
Audit Scope	The scope of this Audit was to analyze the Donation Dapp Smart Contracts for quality, security, and correctness.
Source Code link	https://github.com/valesh-gosats/donation-dapp-contracts
Contracts in Scope	DonationOpportunity.sol OpportunityFactory.sol
Branch	Main
Language	Solidity
Blockchain	EVM
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	11th April 2025 - 15th April 2025
Updated Code Received	17th April 2025
Review 2	17April 2025 - 18th April 2025
Fixed In	https://github.com/valesh-gosats/donation-dapp-contracts/blob/c338180aead3eefede26043168321b577e35e2c1/DonationOpportunity.sol#L46C42-L46C54

Number of Issues per Severity



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

Severity

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

Checked Vulnerabilities

- ✔ Access Management
- ✔ Arbitrary write to storage
- ✔ Centralization of control
- ✔ Ether theft
- ✔ Improper or missing events
- ✔ Logical issues and flaws
- ✔ Arithmetic Computations Correctness
- ✔ Race conditions/front running
- ✔ SWC Registry
- ✔ Re-entrancy
- ✔ Timestamp Dependence
- ✔ Gas Limit and Loops
- ✔ Exception Disorder
- ✔ Gasless Send
- ✔ Use of tx.origin
- ✔ Malicious libraries
- ✔ Compiler version not fixed
- ✔ Address hardcoded
- ✔ Divide before multiply
- ✔ Integer overflow/underflow
- ✔ ERC's conformance
- ✔ Dangerous strict equalities
- ✔ Tautology or contradiction
- ✔ Return values of low-level calls
- ✔ Missing Zero Address Validation
- ✔ Private modifier
- ✔ Revert/require functions
- ✔ Multiple Sends
- ✔ Using suicide
- ✔ Using delegatecall
- ✔ Upgradeable safety
- ✔ Using throw

- [x] Using inline assembly
- [x] Unsafe type inference
- [x] Style guide violation
- [x] 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 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.

Medium Severity Issues

Flawed Timelock logic allows owner to completely by-pass it.

Resolved

Path

DonationOpportunity.sol

Function

emergencyWithdraw

Description

The default value of lastWithdrawRequest is zero. According to the contract logic, the owner needs to first call requestEmergencyWithdraw changing the value of lastWithdrawRequest to current block.timestamp . Once that is done, emergencyWithdraw can be called after withdraw delay is over. Since timelock request does not check whether the withdraw request was actually made or not. Consider the following scenario:

•A DonationOpportunity contract is deployed and receives several donations
•After the contract has been deployed for more than 2 days (WITHDRAW_DELAY)
•The owner can directly call emergencyWithdraw() without first calling requestEmergencyWithdraw()
•The timelock check require(block.timestamp >= lastWithdrawRequest + WITHDRAW_DELAY) will pass because lastWithdrawRequest is 0
•All funds are immediately withdrawn, circumventing the intended 2-day warning period for donors

This undermines the entire purpose of the timelock mechanism, which was presumably designed to give donors and other stakeholders time to react before an emergency withdrawal occurs.

Recommendation

Modify the emergencyWithdraw() function to check if a withdrawal request was actually made by adding an additional state variable.

```
bool public withdrawalRequested;

function requestEmergencyWithdraw() external onlyOwner {
      lastWithdrawRequest = block.timestamp;
      withdrawalRequested = true;
}

function emergencyWithdraw() external onlyOwner nonReentrant {
      require(withdrawalRequested, "No withdrawal requested");
      require(block.timestamp >= lastWithdrawRequest + WITHDRAW_DELAY,
"Timelock active");

      // Reset the request flag
      withdrawalRequested = false;

      // Rest of the function remains the same
      // ...
}
```

Users can keep on donating even when the funding goal is reached

Resolved

Path

DonationOpportunity.sol

Function

emergencyWithdraw

Description

The DonationOpportunity contract does not implement any mechanism to automatically stop accepting donations once the funding goal has been reached. This allows the donation campaign to raise funds significantly beyond its stated target without requiring any action from the campaign creator.

Although there is no potential loss for the recipient for anybody, hyper-exceeding the decided funding amount might convey a sense of dishonesty since the meaning of funding goal becomes ambiguous.

Recommendation

Compare the incoming donations with the funding goal variable. For the edge case, make sure to only take partial amount and refund the excess amount back to the user in case donation amount exceeds funding goal.

Informational Severity Issues

Fee always goes to factory address

Resolved

Path

DonationOpportunity.sol

Function

-

Description

The contract has a feeRecipient variable and an updateFeeRecipient() function, but fees are always sent to the factory contract, not to feeRecipient.

This completely removes the usability of feeRecipient address.

Recommendation

Consider removing feeReciepient if it is not meant to receive fee.

Remove unused state variables

Resolved

Path

DonationOpportunity.sol

Function

-

Description

Many state variables are not being actively used in the contract. These variables waste gas during contract deployment and state updates, increase contract complexity, and can lead to confusion for developers and auditors.

Recommendation

Consider removing the state variables if they are not actively used for frontend keeping.

Functional Tests

Some of the tests performed are mentioned below:

- Pause and Upause functionality check
- Campaign Start/Stop Functionality
- Maximum Donors Limit Enforcement
- Emergency Withdrawal Timelock Enforcement.

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 Donation Dapp. We performed our audit according to the procedure described above.

Issues of Medium and Information severity were found.

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

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

Follow Our Journey

AUDIT REPORT

April , 2025

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com