Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.05.05, the SlowMist security team received the Ethereum R1 team's security audit application for Simple Eth Donations, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

This is an audit of the Ethereum R1 ETH donation contract. Users can donate ETH to the contract before the donation period ends. Once the expected donation goal is reached, the owner can withdraw all ETH from the contract. If the expected donation goal is not met by the end of the donation period, users can reclaim the ETH they have donated.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Self-donation to meet donationsGoal conditions	Design Logic Audit	Medium	Fixed
N2	Missing Zero Value Checks in Constructor	Design Logic Audit	Suggestion	Acknowledged
N3	Inconsistent Donation Time Validation	Design Logic Audit	Low	Fixed

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/ethereum-r1/simple-eth-donations

commit: 07731e2ae8594f6bd490390845d67e97ed2227a2

Fixed Version:

https://github.com/ethereum-r1/simple-eth-donations

commit: b594423375bb8097147c982e4e99954fffae940e

Audit Scope:

```
./src
└── EthDonations.sol
```

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

EthDonations			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
donate	Public	Payable	-
returnDonation	External	Can Modify State	-
queueClaim	External	Can Modify State	onlyOwner
claim	External	Can Modify State	onlyOwner
addDonationsFor	External	Payable	onlyOwner
rescueToken	External	Can Modify State	onlyOwner
<Receive Ether>	External	Payable	-
<Fallback>	External	Payable	-

4.3 Vulnerability Summary

[N1] [Medium] Self-donation to meet donationsGoal conditions

Category: Design Logic Audit

Content

In the contract, when the user's ETH donations reach the `donationsGoal`, the owner can set the contract to a state where the donated funds can be claimed by calling the `queueClaim` function. However, if the current time has already passed `donationsEndTime` and the donation amount has not yet met `donationsGoal`, since the `queueClaim` function checks `address(this).balance`, a user could forcibly send ETH to the EthDonations contract via `selfdestruct`, causing the contract to meet the `donationsGoal` condition and enabling the funds to be claimed.

Code location: src/EthDonations.sol#L60

```
    function queueClaim() external onlyOwner {
        if (claimTimestamp > 0) revert DonationsAlreadyQueued();
        uint256 amount = address(this).balance;
        if (amount < donationsGoal) revert DonationsGoalNotReached();

        claimTimestamp = block.timestamp;
    }
```

Solution

If this is not the intended design, it is recommended to use a global variable to track the total amount donated by users, rather than relying on `address(this).balance` for the check.

Status

Fixed

[N2] [Suggestion] Missing Zero Value Checks in Constructor

Category: Design Logic Audit

Content

The constructor lacks crucial validation for zero address and zero value parameters, which could lead to contract deployment with invalid configurations.

Code location: src/EthDonations.sol#L29-L33

```
    constructor(uint256 _donationsGoal, uint256 _donationsEndTime, address _owner) {
        donationsGoal = _donationsGoal;
        donationsEndTime = _donationsEndTime;
        _initializeOwner(_owner);
    }
```

Solution

It is recommended to ensure that, during contract initialization, `_donationsGoal` must be greater than 0, `_donationsEndTime` must be later than the current time, and `_owner` must not be the zero address.

Status

Acknowledged

[N3] [Low] Inconsistent Donation Time Validation

Category: Design Logic Audit

Content

The contract contains an inconsistency in how donation end time validation is performed between the donate and addDonationsFor functions. While both functions should enforce the same time-based restrictions, they implement the checks differently, which could lead to confusion and potential manipulation. In the donate function, donations are rejected when block.timestamp >= donationsEndTime (greater than or equal to), while in the addDonationsFor function, donations are rejected only when block.timestamp > donationsEndTime (strictly greater than). This means the owner can still add donations exactly at the end time when regular users cannot.

Code location: src/EthDonations.sol#L35-L42, #L75-L91

```
    function donate() public payable {
        if (block.timestamp >= donationsEndTime) revert DonationsEnded();
        …
     }
    function addDonationsFor(address[] calldata donors, uint256[] calldata amounts)
  external payable onlyOwner {
        ...
        if (block.timestamp > donationsEndTime) revert DonationsEnded();
        …
     }
```

Solution

It’s recommended to standardize the time validation across both functions to ensure consistent behavior.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002505060001	SlowMist Security Team	2025.05.05 - 2025.05.06	Passed

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 medium risk, 1 low risk, and 1 suggestion. All the findings were fixed or acknowledged. The code was not deployed to the mainnet.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.