The Endless Puzzle of Security

Protocol Audit Report

Version 1.0

Muhammad Kashif

October 14, 2025

Protocol Audit Report

Muhammad Kashif

October 14, 2025

Prepared by: Muhammad Kashif

Security Researcher: Muhammad Kashif

Table of Contents

Protocol Summary

This contract allows the creator to invite a select group of people to vote on something and provides an eth reward to the `for` voters if the proposal passes, otherwise refunds the reward to the creator. The creator of the contract is considered "Trusted".

This contract has been intentionally simplified to remove much of the extra complexity in order to help you find the particular bug without other distractions. Please read the comments carefully as they note specific findings that are excluded as the implementation has been purposefully kept simple to help you focus on finding the harder to find and more interesting bug.

This contract intentionally has no time-out period for the voting to complete; lack of a time-out period resulting in voting never completing is not a valid finding as this has been intentionally omitted to simplify the codebase.

Disclaimer

I makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

Github Codebase Link

```
git clone https://github.com/Cyfrin/2023-12-Voting-Booth
```

Commit Hash

```
5b9554656d53baa2086ab7c74faf8bdeaf81a8b7
```

Scope

```
./src/
#-- VotingBooth.sol
```

Issues found

Severity	No of issues found
High	1
Medium	0
Low	0
Info	0
Total	0

Findings

High

[H-1] Incorrect calculation of `rewardPerVoter` in the `_distributeRewards` function

Relevant Github Link

```
https://github.com/Cyfrin/2023-12-Voting-Booth/blob/00639857
    d2933a05bc21009852bc6f6c362a3dc5/src/VotingBooth.sol#L192
```

Description The incorrect calculation of `rewardPerVoter` causes a lower distribution of rewards to each voter. Since there is no additional withdraw function, the leftover funds in the contract will remain permanently locked.

In the function `_distributeRewards` uses `totalVotes` instead of `totalVotesFor` while calculating `rewardPerVoter` which gives an incorrect result when `totalVotesFor` ≠ `totalVotes` for a passed proposal as mentioned below code block.

```
else {
@>          uint256 rewardPerVoter = totalRewards / totalVotes;

            for (uint256 i; i < totalVotesFor; ++i) {
```

Impact The incorrect calculation leads to a lower or incorrect distribution of rewards. Additionally, since no withdrawal function is declared, the leftover funds remain permanently locked inside the contract.

Proof of Concepts The scenario is as follows: there are a total of five voters who are allowed to vote. User 1 votes true for the proposal. User 2 votes false for the proposal. User 3 votes true for the proposal.

After three users have voted, the quorum is reached and the reward distribution is initiated. The reward is supposed to be distributed equally among users who voted true, but the code distributes the rewards based on the total number of voters who voted, rather than only those who voted true.

Hence, the distribution is calculated as: `rewards / totalNumberOfVotes` instead of `rewards / totalNumberOfVotesWhoVotedTrue`.

Proof Of Code To test this, include the following code in the `VotingBoothTest.t.sol` file:

```
function testVotersReceivedTheActualAmountOfFunds() public {
        uint256 startingAmount = address(this).balance;

        vm.prank(address(0x1));
        booth.vote(true);

        vm.prank(address(0x2));
        booth.vote(false);

        vm.prank(address(0x3));
        booth.vote(true);

        uint256 totalNumberOfVotersVotedFOR = 2;
        uint256 expectedFundToReceive = ETH_REWARD/
            totalNumberOfVotersVotedFOR;

        assertEq(address(0x1).balance, expectedFundToReceive );
        assertEq(address(0x2).balance, expectedFundToReceive );
        assertEq(address(this).balance, 0 );
    }
```

Recommended mitigation In order to mitigate the incorrect distribution of rewards, change the divisor from `totalRewards` / `totalVotes` to `totalRewards` /`totalVotesFor` in the `_distributeRewards` function.

```diff
-    uint256 rewardPerVoter = totalRewards / totalVotes;
+    uint256 rewardPerVoter = totalRewards / totalVotesFor;
```