# Protocol Audit Report

Version 1.0

*Ayoe.eth*

December 19, 2023

# Protocol Audit Report

Tss

December 19, 2023

Prepared by: Tss Lead Auditors: Ayo.eth

- Tss

## 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".

## Disclaimer

The Tss team 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 by the team 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 |
| Likelihood | High | H | H/M | M |
| | 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

**The findings described in this document correspond the following commit hash:**

Commit Hash

```
5b9554656d53baa2086ab7c74faf8bdeaf81a8b7
```

### Scope

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

### Roles

- `creator` - Deployer of the protocol, they are a trusted used who will receive the funds if a vote fails.
- `AllowedVoters` - A list of addresses that are allowed to vote on proposals.

## Executive Summary

- Add some not about the audit went, types of things you found, etc.*

*We spent X hours with Z auditors using Y tols. etc*

### Issues found

| Severity | Number of issues found |
|---|---|
| High | 1 |
| Medium | 0 |
| Low | 0 |
| Info | 0 |
| Total | 0 |

# Findings

## High

### [H-1] Incorrect Reward Distribution to 'For' Voters Locks Remaining Funds in the Contract

**Description:** Within the `VotingBooth::_distributeRewards` function, the calculation for total rewards fails to accurately distribute rewards among 'for' voters. The current approach involves deriving `rewardPerVoter` by dividing the `totalRewards` by the `totalVotes`, inadvertently considering votes against the proposal. Moreover, the calculation for the final voter to prevent to leaving dust funds results in an incomplete distribution, which actually leaves dust.

```
uint256 rewardPerVoter = totalRewards / totalVotes;
```

**Impact:** Misallocation of funds occurs, preventing correct allocation to 'for' voters. Consequently, the contract retains leftover funds indefinitely, rendering it a one-time use contract.

**Proof of Concept:** (Proof Of Code) Follow the steps below:

1. Import `StdInvariant` from `forge-std/StdInvariant.sol` into `VotingBoothTest.t.sol` contract for invariant testing.

```
import { StdInvariant } from 'forge-std/StdInvariant.sol'
```

2. Add the `StdInvariant` to the inheritance. Ensure that `StdInvariant` is the first in the inheritance chain to avoid `Linearization of inheritance graph impossible` error.

```
contract VotingBoothTest is StdInvariant, Test {}
```

3. Introduce the following invariant test in the contract.

Code

```
  function invariant_booth_balance_must_be_correct() public view {
        if (booth.isActive()) {
            assert(address(booth).balance == ETH_REWARD);
        } else {
            console.log("Voting Booth Balance: ", address(booth).
               balance);
            assert(address(booth).balance == 0);
        }
    }
```

4. The assertion will fail, demonstrating incomplete fund distribution to voters.

**Recommeded Mitigation:**

This are the recommended mitigation

1. Revise the denominator of `rewardPerVoter` to `totalVotesFor`:

```
uint256 rewardPerVoter = totalRewards / totalVotesFor;
```

2. Adjust the last voter's reward to avoid leftover funds:

```
if (i == totalVotesFor - 1) {
  rewardPerVoter = address(this).balance
}
```