4th05

Ethos

Security Review Report

5 December 2024

Security Review Report

4th05

December 5, 2024

Table of Contents

- Table of Contents
- Protocol Summary
- Disclaimer
- Risk Classification
- Overview
- Scope
- Issues found
- Findings
  - [H1] Wrong assignment value to `marketFunds[profileId]` may cause the `ReputationMarket::withdrawGraduatedMarketFunds` to revert for not enough ETH

Protocol Summary

Ethos Network is an onchain social reputation platform. This contest focuses on the financial stakes: vouching for others, participating in reputation markets. This builds on existing Ethos Network social contracts.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

Risk Classification

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

Overview

Contest platform	Sherlock
LOC	1096
Language	Solidity
Commit	57c02df7c56f0b18c681a89ebccc28c86c72d8d8
Previous audits	Sherlock, LightChaser

Scope

- EthosVouch.sol
- ReputationMarket.sol
- ReputationMarketErrors.sol
- Common.sol

Issues found

Severity	Number of issues found
High	1
Medium	0
Low	0
Info	0

Findings

[H1] Wrong assignment value to `marketFunds[profileId]` may cause the `ReputationMarket::withdrawGraduatedMarketFunds` to revert for not enough ETH

Summary

When `ReputationMarket::withdrawGraduatedMarketFunds` is called, it may reverts because of insufficient funds in the contract. This issue is caused in turn by a wrong value assignment to `marketFunds[profileId]` which is made in `ReputationMarket::buyVotes`.

Relevant GitHub Links

https://github.com/sherlock-audit/2024-11-ethos-network-ii/blob/main/ethos/packages/contracts/contracts/Reputati
https://github.com/sherlock-audit/2024-11-ethos-network-ii/blob/main/ethos/packages/contracts/contracts/Reputati
https://github.com/sherlock-audit/2024-11-ethos-network-ii/blob/main/ethos/packages/contracts/contracts/Reputati

Root Cause

In the `ReputationMarket::buyVotes` function it is assigned `marketFunds[profileId] += fundsPaid`.

```
     // Apply fees first
    applyFees(protocolFee, donation, profileId);

    // Update market state
    markets[profileId].votes[isPositive ? TRUST : DISTRUST] +=
       votesBought;
    votesOwned[msg.sender][profileId].votes[isPositive ? TRUST :
       DISTRUST] += votesBought;

    // Add buyer to participants if not already a participant
    if (!isParticipant[profileId][msg.sender]) {
      participants[profileId].push(msg.sender);
      isParticipant[profileId][msg.sender] = true;
    }

    // Calculate and refund remaining funds
    uint256 refund = msg.value - fundsPaid;
    if (refund > 0) _sendEth(refund);

    // tally market funds
    marketFunds[profileId] += fundsPaid;
```

The value of fundsPaid is taken as output of ReputationMarket::_calculateBuy and it considers protocolFee and the donation being fundsPaid += protocolFee + donation .

```
    while (fundsAvailable >= votePrice) {
      fundsAvailable -= votePrice;
      fundsPaid += votePrice;
      votesBought++;

      market.votes[isPositive ? TRUST : DISTRUST] += 1;
      votePrice = _calcVotePrice(market, isPositive);
    }
    fundsPaid += protocolFee + donation;
```

The assignment to marketFunds[profileId] in ReputationMarket::buyVotes is done after that protocolFee amount has been sent through ReputationMarket::applyFees to the protocolFeeAddress (so it is not still in ReputationMarket balance).

Donations may be withdrawn by users (the recipient user) by calling a function ReputationMarket::withdrawDonations.

Then, when ReputationMarket::withdrawGraduatedMarketFunds is called it may reverts because of ReputationMarket running out of funds.

```
function applyFees(
    uint256 protocolFee,
    uint256 donation,
    uint256 marketOwnerProfileId
  ) private returns (uint256 fees) {
    donationEscrow[donationRecipient[marketOwnerProfileId]] += donation;
    if (protocolFee > 0) {
      (bool success, ) = protocolFeeAddress.call{ value: protocolFee }("");
      if (!success) revert FeeTransferFailed("Protocol fee deposit failed");
    }
    fees = protocolFee + donation;
  }
  function withdrawGraduatedMarketFunds(uint256 profileId) public whenNotPaused {
  address authorizedAddress = contractAddressManager.getContractAddressForName(
    "GRADUATION_WITHDRAWAL"
  );
  if (msg.sender != authorizedAddress) {
    revert UnauthorizedWithdrawal();
  }
  _checkMarketExists(profileId);
  if (!graduatedMarkets[profileId]) {
    revert MarketNotGraduated();
  }
  if (marketFunds[profileId] == 0) {
    revert InsufficientFunds();
  }

  _sendEth(marketFunds[profileId]);
  emit MarketFundsWithdrawn(profileId, msg.sender, marketFunds[profileId]);
  marketFunds[profileId] = 0;
}
```

Internal pre-conditions

At least a market has been created, at least 1 vote has been bought, and then it has been graduated. (with a `marketFunds[profileId] > 0`)

External pre-conditions

`Recipient address` withdraws donations of the market.

`authorizedAddress` wants to withdraw the market funds calling the `ReputationMarket::withdrawGraduatedMarketFunds`.

Attack Path

Market is created.

At least 1 vote has been bought, and the `ProtocolFee` is sent to the `protocolFeeAddress`.

Donations are withdrawn. (Not always necessary because in some cases just the sum of all protocolFees paid for the market votes bought could be enough to cause that `ReputationMarket::withdrawGraduatedMarketFunds` revert when called).

Market is graduated.

`ReputationMarket::withdrawGraduatedMarketFunds` is called by the `authorizedAddress` for the market (graduated market) and it reverts because of insufficient funds.

Impact

The `ReputationMarket.sol` could run out of funds, with these possible impacts:

The `authorizedAddress` could not be able to withdraw the funds of the market (graduated market), using `ReputationMarket::withdrawGraduatedMarketFunds` if the ETH balance of `ReputationMarket.sol` is < than the `marketFunds[profileId]` (which is the amount that should be withdrawn) because of `protocolFees` and donations that have already left the contract balance.

The `recipient address` could not be able to withdraw the `donations` (having that `donationEscrow[recipientAddress]>0`). This may happen if `authorizedAddress` withdraws `marketFunds[profileId]` first through the `ReputationMarket::withdrawGraduatedMarketFu` and the balance of the contract had enough ETH.

Mitigation

A possible solution could be this:

```
       // Apply fees first
      applyFees(protocolFee, donation, profileId);

      // Update market state
      markets[profileId].votes[isPositive ? TRUST : DISTRUST] +=
         votesBought;
      votesOwned[msg.sender][profileId].votes[isPositive ? TRUST :
         DISTRUST] += votesBought;

      // Add buyer to participants if not already a participant
      if (!isParticipant[profileId][msg.sender]) {
        participants[profileId].push(msg.sender);
        isParticipant[profileId][msg.sender] = true;
      }

      // Calculate and refund remaining funds
      uint256 refund = msg.value - fundsPaid;
      if (refund > 0) _sendEth(refund);

      // tally market funds
 -    marketFunds[profileId] += fundsPaid;
 +    marketFunds[profileId] += fundsPaid - protocolFee - donation;
```