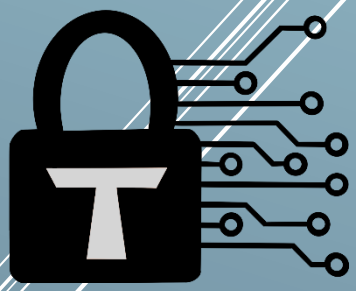
Trust Security

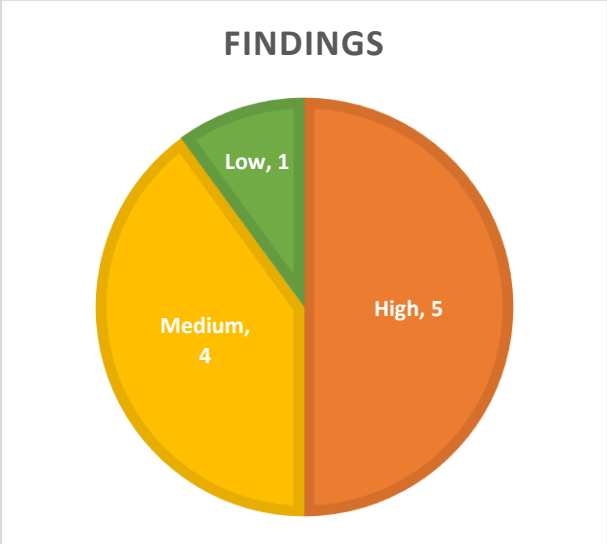


Smart Contract Audit

StakeDAO VoteMarket V2

10/09/24

Executive summary

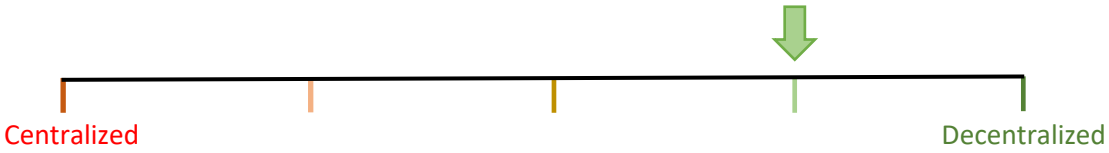


Category	Reward Management
Audited file count	10
Lines of Code	891
Auditor	cccz ether_sky
Time period	19/08/2024- 01/09/2024

Findings

Severity	Total	Fixed	Acknowledged
High	5	5	-
Medium	4	4	-
Low	1	-	1

Centralization score



Signature

Digitally signed by Trust
Date: 2024.09.10 13:40:41 +03'00'

Document properties

Versioning

Version	Date	Description
0.1	01/09/2024	Client report
0.2	10/09/2024	Mitigation review

Contact

Trust

trust@trust-security.xyz

Introduction

Trust Security has conducted an audit at the customer's request. The audit is focused on uncovering security issues and additional bugs contained in the code defined in scope. Some additional recommendations have also been given when appropriate.

Scope

The following files are in scope of the audit:

- ./Votemarket.sol
- ./verifiers/Verifier.sol
- ./oracle/Oracle.sol
- ./verifiers/RLPDecoder.sol
- ./interfaces/IVotemarket.sol
- ./oracle/OracleLens.sol
- ./interfaces/IOracle.sol
- ./interfaces/IHook.sol
- ./interfaces/IGaugeController.sol
- ./interfaces/IOracleLens.sol

Repository details

- **Repository URL:** https://github.com/stake-dao/votemarket-v2
- **Commit hash:** d7e963300dc9001c7202f34a53f5b667bcd83df5
- **Mitigation review hash #1:** 386d6a3066989826ef721d70913ee1651bd37ff1
- **Mitigation review hash #2:** 93df17035a1eaac22a1140bd561248c77d29a0c3

About Trust Security

Trust Security has been established by top-end blockchain security researcher Trust, in order to provide high quality auditing services. Trust is the leading auditor at competitive auditing service Code4rena, reported several critical issues to Immunefi bug bounty platform and is currently a Code4rena judge.

About the Auditors

A top competitor in audit contests, cccz has achieved superstar status in the security space. He is a Black Hat / DEFCON speaker with rich experience in both traditional and blockchain security.

ether_sky is a security researcher with a focus on blockchain security. He specializes in algorithms and data structures and has a solid background in IT development. He has placed at the top of audit contests in Code4rena and Sherlock recently.

Disclaimer

Smart contracts are an experimental technology with many known and unknown risks. Trust Security assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Furthermore, it is known to all parties that changes to the audited code, including fixes of issues highlighted in this report, may introduce new issues and require further auditing.

Methodology

In general, the primary methodology used is manual auditing. The entire in-scope code has been deeply looked at and considered from different adversarial perspectives. Any additional dependencies on external code have also been reviewed. Fuzz tests and unit tests have also been used as needed.

Qualitative analysis

Metric	Rating	Comments
Code complexity	**Good**	Project kept code as simple as possible, despite implementing custom data structures for efficiency.
Documentation	**Moderate**	Project is still under active development and currently lacks documentation.
Best practices	**Good**	Project generally follows best practices.
Centralization risks	**Good**	The Project introduces several centralization issues.

Findings

High severity findings

TRST-H-1 Rewards sent to hook will be withdrawn again when closing the campaign

- **Category:** Logical flaws
- **Source:** Votemarket.sol
- **Status:** Fixed

Description

In *_closeCampaign()*, the **totalRewardAmount - totalClaimedByCampaignId[campaignId]** is used as the leftover rewards to send to the manager.

```
    function _closeCampaign(uint256 campaignId, uint256 totalRewardAmount, address
rewardToken, address receiver)
        internal
    {
        // 1. Calculate leftover rewards
        uint256 leftOver = totalRewardAmount - totalClaimedByCampaignId[campaignId];

        // 2. Transfer leftover rewards to the receiver
        SafeTransferLib.safeTransfer({token: rewardToken, to: receiver, amount:
leftOver});
```

totalClaimedByCampaignId[campaignId] is the rewards that have been claimed by users and will be updated when claiming.

However, in addition to the rewards being claimed by users, leftover rewards are also sent to hook in *_updateRewardPerVote()*, but these sent rewards are not counted in **totalClaimedByCampaignId[campaignId]**.

```
        if (rewardPerVote > campaign.maxRewardPerVote) {
            rewardPerVote = campaign.maxRewardPerVote;

            // 4. Calculate leftover rewards
            uint256 leftOver = period.rewardPerPeriod -
rewardPerVote.mulDiv(totalVotes, 1e18);

            // 5. Handle leftover rewards
            address hook = campaign.hook;
            if (hook != address(0)) {
                // Transfer leftover to hook contract
                SafeTransferLib.safeTransfer({token: campaign.rewardToken, to: hook,
amount: leftOver});
```

Consider that the campaign manager provides 10,000 reward tokens, 2,000 are claimed by users, 8,000 are sent to the hook, and the manager calls *closeCampaign()* and receives additional 8,000 reward tokens.

Recommended mitigation

It is recommended to count the rewards sent to hook in **totalClaimedByCampaignId[campaignId]**.

```
            if (hook != address(0)) {
                // Transfer leftover to hook contract
                SafeTransferLib.safeTransfer({token: campaign.rewardToken, to: hook,
amount: leftOver});
+               totalClaimedByCampaignId[campaignId] += leftOver;
                // Trigger the hook
                (bool success,) = hook.call(
                    abi.encodeWithSelector(
                        IHook.doSomething.selector,
                        campaignId,
                        campaign.chainId,
                        campaign.rewardToken,
                        epoch,
                        leftOver,
                        hookData
                    )
                );
                if (!success) {
                    /// If the hook reverts, delete the hook to trigger rollover in
the next epoch instead.
                    delete campaign.hook;
                }
            }
```

Team response

Fixed.

Mitigation Review

The fix implements the recommendation.

TRST-H-2 updateEpoch() can be called to steal rewards after the campaign is closed

- **Category:** Logical flaws
- **Source:** Votemarket.sol
- **Status:** Fixed

Description

If the campaign is closed before it starts, all rewards will be sent to the manager.

```
    function closeCampaign(uint256 campaignId) external nonReentrant
notClosed(campaignId) {
        // 1. Get campaign data and calculate time windows
        Campaign storage campaign = campaignById[campaignId];
        uint256 currentTime = block.timestamp;
        uint256 claimWindow = campaign.endTimestamp + CLAIM_WINDOW_LENGTH;
        uint256 closeWindow = claimWindow + CLOSE_WINDOW_LENGTH;
        address receiver = campaign.manager;

        // 2. Handle different closing scenarios
        if (currentTime < campaign.startTimestamp) {
            // 2a. Campaign hasn't started yet
            _isManagerOrRemote(campaignId);
            _checkForUpgrade(campaignId, campaign.startTimestamp);
```

Since *updateEpoch()* does not have the *notClosed* modifier, *updateEpoch()* can still be called after the campaign is closed.

```
    function updateEpoch(uint256 campaignId, uint256 epoch, bytes calldata hookData)
        external
        nonReentrant
        validEpoch(campaignId, epoch)
        returns (uint256 epoch_)
    {
        epoch_ = _updateEpoch(campaignId, epoch, hookData);
    }
```

In *_updateRewardPerVote()*, it will send **leftOver** amount of reward tokens to hook, attacker can set very small **maxRewardPerVote** to steal the reward tokens.

```
        if (rewardPerVote > campaign.maxRewardPerVote) {
            rewardPerVote = campaign.maxRewardPerVote;

            // 4. Calculate leftover rewards
            uint256 leftOver = period.rewardPerPeriod -
rewardPerVote.mulDiv(totalVotes, 1e18);

            // 5. Handle leftover rewards
            address hook = campaign.hook;
            if (hook != address(0)) {
                // Transfer leftover to hook contract
                SafeTransferLib.safeTransfer({token: campaign.rewardToken, to: hook,
amount: leftOver});
```

Recommended mitigation

It is recommended to add the *notClosed()* modifier to the *updateEpoch()* function.

Team response

Fixed.

Mitigation Review

The fix implements the recommendation.

TRST-H-3 The rewards added in the last epoch will be lost

- **Category:** Logical flaws
- **Source:** Votemarket.sol
- **Status:** Fixed

Description

The manager or anyone can increase the rewards for the campaign, even during the last epoch. Suppose the last epoch of the campaign is E, and the manager wants to increase the rewards and the number of periods in epoch E.

These upgrades will be stored in **campaignUpgradeById** for the next epoch, E + 1.

```
    function manageCampaign(
        uint256 campaignId,
```

```
          uint8 numberOfPeriods,
          uint256 totalRewardAmount,
          uint256 maxRewardPerVote
    ) external nonReentrant onlyManagerOrRemote(campaignId) notClosed(campaignId) {
          if (getRemainingPeriods(campaignId, currentEpoch()) == 0) revert
CAMPAIGN_ENDED();

          uint256 epoch = currentEpoch() + EPOCH_LENGTH;

          if (campaignUpgrade.totalRewardAmount != 0) {
              campaignUpgrade = CampaignUpgrade({
                    numberOfPeriods: campaignUpgrade.numberOfPeriods + numberOfPeriods,
                    totalRewardAmount: campaignUpgrade.totalRewardAmount +
totalRewardAmount,
                    maxRewardPerVote: maxRewardPerVote > 0 ? maxRewardPerVote :
campaignUpgrade.maxRewardPerVote,
                    endTimestamp: campaignUpgrade.endTimestamp + (numberOfPeriods *
EPOCH_LENGTH)
              });
          } else {
              campaignUpgrade = CampaignUpgrade({
                    numberOfPeriods: campaign.numberOfPeriods + numberOfPeriods,
                    totalRewardAmount: campaign.totalRewardAmount + totalRewardAmount,
                    maxRewardPerVote: maxRewardPerVote > 0 ? maxRewardPerVote :
campaign.maxRewardPerVote,
                    endTimestamp: campaign.endTimestamp + (numberOfPeriods *
EPOCH_LENGTH)
              });
          }

          campaignUpgradeById[campaignId][epoch] = campaignUpgrade;
}
```

The changes are applied in the *_checkForUpgrade*() function.

```
    function _checkForUpgrade(uint256 campaignId, uint256 epoch) internal {
          CampaignUpgrade memory campaignUpgrade =
campaignUpgradeById[campaignId][epoch];

          if (campaignUpgrade.totalRewardAmount != 0) {
              Campaign storage campaign = campaignById[campaignId];

              campaign.endTimestamp = campaignUpgrade.endTimestamp;
              campaign.numberOfPeriods = campaignUpgrade.numberOfPeriods;
              campaign.maxRewardPerVote = campaignUpgrade.maxRewardPerVote;
              campaign.totalRewardAmount = campaignUpgrade.totalRewardAmount;
          }
    }
```

To trigger this function for epoch E + 1, users need to update this epoch.

```
    function _updateEpoch(uint256 campaignId, uint256 epoch, bytes calldata hookData)
internal returns (uint256) {
        Period storage period = _getPeriod(campaignId, epoch);
        if (period.updated) return epoch;

        _checkForUpgrade(campaignId, epoch);
    }
```

However, the *validEpoch*() modifier will prevent updating epoch E + 1 because it is not a valid epoch.

```
    modifier validEpoch(uint256 campaignId, uint256 epoch) {
```

```
        if (
            epoch > block.timestamp || epoch < campaignById[campaignId].startTimestamp
                || epoch >= campaignById[campaignId].endTimestamp || epoch % EPOCH_LENGTH
!= 0
        ) revert EPOCH_NOT_VALID();
        _;
    }
```

As a result, the added rewards will be lost, and the increased number of periods won't be applied.

Recommended mitigation

It is recommended to prevent any changes to the campaign in the last epoch.

```
    function manageCampaign(
          uint256 campaignId,
          uint8 numberOfPeriods,
          uint256 totalRewardAmount,
          uint256 maxRewardPerVote
    ) external nonReentrant onlyManagerOrRemote(campaignId) notClosed(campaignId) {
-         if (getRemainingPeriods(campaignId, currentEpoch()) == 0) revert
CAMPAIGN_ENDED();

+         if (getRemainingPeriods(campaignId, currentEpoch()) <= 1) revert
CAMPAIGN_ENDED();
    }
```

The same applies to the *increaseTotalRewardAmount*() function.

Team response

Fixed.

Mitigation Review

The fix implements the recommendation.

TRST-H-4 The rewards will be lost when increasing rewards for the campaign

- **Category:** Logical flaws
- **Source:** Votemarket.sol
- **Status:** Fixed

Description

Suppose the current total reward for the campaign is 10,000, and the manager wants to increase it by 5,000 during epoch E.

These 5,000 rewards will be added when the *_checkForUpgrade*() function is called for epoch E + 1.

```
    function _checkForUpgrade(uint256 campaignId, uint256 epoch) internal {
          CampaignUpgrade memory campaignUpgrade =
campaignUpgradeById[campaignId][epoch];

          if (campaignUpgrade.totalRewardAmount != 0) {
              Campaign storage campaign = campaignById[campaignId];
```

```
            campaign.endTimestamp = campaignUpgrade.endTimestamp;
            campaign.numberOfPeriods = campaignUpgrade.numberOfPeriods;
            campaign.maxRewardPerVote = campaignUpgrade.maxRewardPerVote;
            campaign.totalRewardAmount = campaignUpgrade.totalRewardAmount;
        }
    }
```

In epoch E + 1, the manager again wants to increase the rewards by 5,000.

However, there is no guarantee that epoch E + 1 has been updated.

If this epoch hasn't been updated, the **totalRewardAmount** of the campaign remains 10,000 instead of 15,000.

```
    function manageCampaign(
        uint256 campaignId,
        uint8 numberOfPeriods,
        uint256 totalRewardAmount,
        uint256 maxRewardPerVote
    ) external nonReentrant onlyManagerOrRemote(campaignId) notClosed(campaignId) {
        if (getRemainingPeriods(campaignId, currentEpoch()) == 0) revert
CAMPAIGN_ENDED();

        uint256 epoch = currentEpoch() + EPOCH_LENGTH;

        if (campaignUpgrade.totalRewardAmount != 0) {
            campaignUpgrade = CampaignUpgrade({
                numberOfPeriods: campaignUpgrade.numberOfPeriods + numberOfPeriods,
                totalRewardAmount: campaignUpgrade.totalRewardAmount +
totalRewardAmount,
                maxRewardPerVote: maxRewardPerVote > 0 ? maxRewardPerVote :
campaignUpgrade.maxRewardPerVote,
                endTimestamp: campaignUpgrade.endTimestamp + (numberOfPeriods *
EPOCH_LENGTH)
            });
        } else {
            campaignUpgrade = CampaignUpgrade({
                numberOfPeriods: campaign.numberOfPeriods + numberOfPeriods,
                totalRewardAmount: campaign.totalRewardAmount + totalRewardAmount,
                maxRewardPerVote: maxRewardPerVote > 0 ? maxRewardPerVote :
campaign.maxRewardPerVote,
                endTimestamp: campaign.endTimestamp + (numberOfPeriods *
EPOCH_LENGTH)
            });
        }

        campaignUpgradeById[campaignId][epoch] = campaignUpgrade;
    }
```

As a result, even after updating all epochs, the final **totalRewardAmount** will be 15,000, causing the 5,000 rewards to be lost.

Recommended mitigation

Allow changes only when the epoch has been updated.

```
    function manageCampaign(
        uint256 campaignId,
        uint8 numberOfPeriods,
        uint256 totalRewardAmount,
        uint256 maxRewardPerVote
    ) external nonReentrant onlyManagerOrRemote(campaignId) notClosed(campaignId) {
```

```
        if (getRemainingPeriods(campaignId, currentEpoch()) == 0) revert
CAMPAIGN_ENDED();

+       if (!periodByCampaignId[campaignId][currentEpoch()].updated) revert();
    }
```

The same applies to the *increaseTotalRewardAmount*() function.

Team response

Fixed.

Mitigation Review

The fix implements the recommendation and considers the case of managing the campaign before it starts. Note that *increaseTotalRewardAmount()* and *manageCampaign()* use different checks, and while they are both valid, it's better to unify them.

TRST-H-5 The reward distribution becomes incorrect when the number of periods is increased

- **Category:** Logical flaws
- **Source:** Votemarket.sol
- **Status:** Fixed

Description

Suppose a manager creates a campaign with 5 periods and a total reward of 5,000, the reward per period would be 1,000.

The start epoch is E, and in the third epoch E + 2, we update this epoch, so the reward for E + 2 is also 1,000. Now, the manager wants to increase the number of periods by 3. This upgrade information is stored in **campaignUpgradeById** for epoch E + 3. When epoch E + 3 is updated, the *_checkForUpgrade()* function adjusts the **campaign.endTimestamp** to E + 7, as the total number of periods is now 8.

```
    function _checkForUpgrade(uint256 campaignId, uint256 epoch) internal {
        CampaignUpgrade memory campaignUpgrade =
campaignUpgradeById[campaignId][epoch];

        if (campaignUpgrade.totalRewardAmount != 0) {
            Campaign storage campaign = campaignById[campaignId];

            campaign.endTimestamp = campaignUpgrade.endTimestamp;
            campaign.numberOfPeriods = campaignUpgrade.numberOfPeriods;
            campaign.maxRewardPerVote = campaignUpgrade.maxRewardPerVote;
            campaign.totalRewardAmount = campaignUpgrade.totalRewardAmount;
        }
    }
```

The *getRemainingPeriods*() function returns 5, so in the *_calculateTotalReward*() function, the total reward is calculated as 5 * 1,000 in the epoch E + 3, implying that each of the 8 epochs will have a reward of 1,000.

```
    function _calculateTotalReward(uint256 campaignId, uint256 epoch, uint256
remainingPeriods)
        internal
        view
        returns (uint256 totalReward)
    {
        Period storage previousPeriod = periodByCampaignId[campaignId][epoch -
EPOCH_LENGTH];
        totalReward =
            remainingPeriods > 0 ? previousPeriod.rewardPerPeriod * remainingPeriods :
previousPeriod.rewardPerPeriod;

        return previousPeriod.leftover + totalReward;
    }
```

However, since 3,000 rewards were already distributed over the first 3 epochs, only 2,000 rewards should be available for the remaining 5 epochs.

Recommended mitigation

To accurately calculate the remaining total rewards, it is recommended to sum up the rewards distributed in the previous periods and subtract this from the total rewards of the campaign.

Team response

Fixed.

Mitigation Review

The fix implements the recommendation, adds **totalDistributed** to track distributed rewards and uses **totalRewardAmount-totalDistributed** to calculate remaining rewards.

But the calculation is flawed, consider 5 epochs, the total reward is 5000.

epoch 1: **rewardPerPeriod** = 1000, **totalDistributed** = 1000, and **leftover** = 500.

epoch 2: **rewardPerPeriod** = 1125, **totalDistributed** = 2125, **leftover** = 0.

epoch 3: **rewardPerPeriod** = (5000 - 2125) / 3 = 958 (which should be (5000 - 2125 + 500) / 3 = 1125).

It is recommendated to subtract **leftover** from **totalDistributed** directly and no longer consider **leftover** in *_calculateTotalReward().*

```
function _updateEpoch(uint256 campaignId, uint256 epoch, bytes calldata hookData)
internal returns (uint256) {
    …
    // 7. Update the total distributed amount
-    campaignById[campaignId].totalDistributed += period.rewardPerPeriod;
+    campaignById[campaignId].totalDistributed += (period.rewardPerPeriod -
period.leftover);

    // 8. Mark the period as updated
    period.updated = true;
    return epoch;
}

…

function _calculateTotalReward(uint256 campaignId, uint256 epoch) internal view
returns (uint256 totalReward) {
```

```
    Period storage previousPeriod = periodByCampaignId[campaignId][epoch -
EPOCH_LENGTH];
    totalReward = campaignById[campaignId].totalRewardAmount -
campaignById[campaignId].totalDistributed;
-     return previousPeriod.leftover + totalReward;
+     return totalReward;
}
```

Team response #2

Fixed.

Mitigation Review #2

The fix implements the recommendation.

Medium severity findings

TRST-M-1 closeCampaign() validates the incorrect epoch

- **Category:** Logical flaws
- **Source:** Votemarket.sol
- **Status:** Fixed

Description

The *_validatePreviousState()* call in *closeCampaign()* validates that the second last epoch has been updated, whereas it should actually validate the first last epoch.

```
        } else if (currentTime < closeWindow) {
            // 2c. Within close window, only manager can close
            _isManagerOrRemote(campaignId);
            _validatePreviousState(campaignId, campaign.endTimestamp - EPOCH_LENGTH);
        } else {
            // 2d. After close window, anyone can close and funds go to fee collector
            _validatePreviousState(campaignId, campaign.endTimestamp - EPOCH_LENGTH);
            receiver = feeCollector;
        }
...
    function _validatePreviousState(uint256 campaignId, uint256 epoch) internal view {
        Period storage previousPeriod = periodByCampaignId[campaignId][epoch -
EPOCH_LENGTH];
        if (!previousPeriod.updated) {
            revert PREVIOUS_STATE_MISSING();
        }
    }
```

If the campaign has new rewards in the last epoch, when the manager closes the campaign, since the last epoch is not validated, the **campaignUpgrade** of the last epoch may not be merged into the **campaign**, which will cause the manager to lose the rewards in the **campaignUpgrade**.

Recommended mitigation

It is recommended to change as follows.

```
        } else if (currentTime < closeWindow) {
            // 2c. Within close window, only manager can close
            _isManagerOrRemote(campaignId);
-           _validatePreviousState(campaignId, campaign.endTimestamp - EPOCH_LENGTH);
+           _validatePreviousState(campaignId, campaign.endTimestamp);
        } else {
            // 2d. After close window, anyone can close and funds go to fee collector
-           _validatePreviousState(campaignId, campaign.endTimestamp - EPOCH_LENGTH);
+           _validatePreviousState(campaignId, campaign.endTimestamp);
            receiver = feeCollector;
        }
```

Team response

Fixed.

Mitigation Review

The fix implements the recommendation.

TRST-M-2 An attacker may be able to make hook.call() fail, thus maliciously deleting campaign.hook

- **Category:** Gas-related flaws
- **Source:** Votemarket.sol
- **Status:** Fixed

Description

According to EIP-150, external calls consume up to 63/64 of all available gas.

In *_updateRewardPerVote()*, if the *hook.call()* call fails, it will delete **campaign.hook**, thus preventing **campaign.hook** from receiving the leftover reward in subsequent epochs.

```
            uint256 leftOver = period.rewardPerPeriod -
rewardPerVote.mulDiv(totalVotes, 1e18);

            // 5. Handle leftover rewards
            address hook = campaign.hook;
            if (hook != address(0)) {
                // Transfer leftover to hook contract
                SafeTransferLib.safeTransfer({token: campaign.rewardToken, to: hook,
amount: leftOver});
                // Trigger the hook
                (bool success,) = hook.call(
                    abi.encodeWithSelector(
                        IHook.doSomething.selector,
                        campaignId,
                        campaign.chainId,
                        campaign.rewardToken,
                        epoch,
                        leftOver,
                        hookData
                    )
                );
                if (!success) {
                    /// If the hook reverts, delete the hook to trigger rollover in
the next epoch instead.
                    delete campaign.hook;
                }
```

If the implementation of *hook.doSomething()* is complex and consumes a lot of gas, the attacker can provide little gas to call *updateEpoch()*. In *updateEpoch()*, it needs to execute **delete campaign.hook; period.rewardPerVote = rewardPerVote; period.updated = true;** after *hook.call()* fails, so if 1/64 of the *gasleft()* before *hook.call()* meets these gas consumptions, and 63/64 does not meet the gas consumption in *doSomething()*, malicious hook deletion is feasible.

Recommended mitigation

It is recommended that when *hook.call()* failed, require *hook.call()* to consume less than 63/64 of the available gas, thus guaranteeing that *hook.call()* does not fail due to out-of-gas.

```
+               uint256 gasBeforeCall = gasleft();
                (bool success,) = hook.call(
                    abi.encodeWithSelector(
```

```
                        IHook.doSomething.selector,
                        campaignId,
                        campaign.chainId,
                        campaign.rewardToken,
                        epoch,
                        leftOver,
                        hookData
                    )
                );
                if (!success) {
+                   uint256 gasAfterCall = gasleft();
+                   require(gasBeforeCall - gasAfterCall < gasBeforeCall * 63 / 64,
"Too little gas provided");
                    /// If the hook reverts, delete the hook to trigger rollover in
the next epoch instead.
                    delete campaign.hook;
                }
```

Note that the provided example is not 100% accurate because the 63/64 rule is applied after all call costs are expended, so consider it only as a suggested template.

Team response

Fixed.

Mitigation Review

The fix no longer deletes **campaign.hook**.

TRST-M-3 The total votes for an epoch can be incorrect

- **Category:** Logical flaws
- **Source:** OracleLens.sol
- **Status:** Fixed

Description

Suppose there are 10 voters in an epoch, each with 100 votes. One of these users is blacklisted, so the valid total votes should be 900. During this epoch, the blacklisted user changes their vote weight to 800.

The *isVoteValid*() function will return **false** for this user because their **account_.lastVote** is greater than the current epoch.

```
    function isVoteValid(address account, address gauge, uint256 epoch) external view
returns (bool) {
        IOracle.VotedSlope memory account_ = IOracle(oracle).votedSlopeByEpoch(account,
gauge, epoch);
        if (account_.lastUpdate == 0) revert STATE_NOT_UPDATED();
        if (account_.slope == 0 || epoch >= account_.end || epoch <= account_.lastVote)
return false;

        return true;
    }
```

However, the *getAccountVotes*() function does not account for this and returns 800 as the vote weight.

```
    function getAccountVotes(address account, address gauge, uint256 epoch) external view
returns (uint256) {
        IOracle.VotedSlope memory account_ = IOracle(oracle).votedSlopeByEpoch(account,
gauge, epoch);
        if (account_.lastUpdate == 0) revert STATE_NOT_UPDATED();
        if (epoch >= account_.end) return 0;

        return account_.slope * (account_.end - epoch);
    }
```

As a result, the *_getAdjustedVote()* function for this epoch returns 200 and this is obviously incorrect.

```
    function _getAdjustedVote(uint256 campaignId, uint256 epoch) internal view returns
(uint256) {
        EnumerableSetLib.AddressSet storage addressesSet_ =
addressesByCampaignId[campaignId];

        uint256 totalVotes =
IOracleLens(ORACLE).getTotalVotes(campaignById[campaignId].gauge, epoch);

        uint256 addressesVotes;
        for (uint256 i = 0; i < addressesSet_.length(); i++) {
            addressesVotes +=
                IOracleLens(ORACLE).getAccountVotes(addressesSet_.at(i),
campaignById[campaignId].gauge, epoch);
        }

        if (whitelistOnly[campaignId]) {
            return addressesVotes;
        }

        return totalVotes - addressesVotes;
    }
```

The error causes the reward distribution for this epoch to be incorrect.

Recommended mitigation

It is recommended to modify the *getAccountVotes()* function.

```
    function getAccountVotes(address account, address gauge, uint256 epoch) external view
returns (uint256) {
        IOracle.VotedSlope memory account_ = IOracle(oracle).votedSlopeByEpoch(account,
gauge, epoch);
        if (account_.lastUpdate == 0) revert STATE_NOT_UPDATED();
        if (epoch >= account_.end) return 0;

    +    if (epoch <= account_.lastVote) return 0;

        return account_.slope * (account_.end - epoch);
    }
```

Team response

Fixed.

Mitigation Review

The fix implements the recommendation.

TRST-M-4 The rewardPerPeriod for the first epoch can be updated incorrectly

- **Category:** Logical flaws
- **Source:** Votemarket.sol
- **Status:** Fixed

Description

Suppose a manager creates a campaign with 1 period and a total reward of 1,000. The current time is before the campaign's start time and the manager decides to increase the number of periods to 10.

This upgrade information will be stored in the **campaignUpgradeById[id][startTimestamp]**. When the first epoch(i.e. **startTimestamp**) is updated, the **rewardPerPeriod** for this epoch is incorrectly calculated as 1,000 in the *_checkForUpgrade*() function.

```
    function _checkForUpgrade(uint256 campaignId, uint256 epoch) internal {
        CampaignUpgrade memory campaignUpgrade = campaignUpgradeById[campaignId][epoch];

        if (campaignUpgrade.totalRewardAmount != 0) {
            Campaign storage campaign = campaignById[campaignId];

            if (epoch == campaign.startTimestamp) {
                periodByCampaignId[campaignId][epoch].rewardPerPeriod =
                    campaignUpgrade.totalRewardAmount.mulDiv(1,
campaign.numberOfPeriods);
            } else {
                periodByCampaignId[campaignId][epoch - EPOCH_LENGTH].leftover +=
                    campaignUpgrade.totalRewardAmount - campaign.totalRewardAmount;
            }

            campaign.endTimestamp = campaignUpgrade.endTimestamp;
            campaign.numberOfPeriods = campaignUpgrade.numberOfPeriods;
            campaign.maxRewardPerVote = campaignUpgrade.maxRewardPerVote;
            campaign.totalRewardAmount = campaignUpgrade.totalRewardAmount;
        }
    }
```

However, the total rewards should be divided by the new 10 periods, not the original 1 period.

Recommended mitigation

The *_checkForUpgrade()* function should be modified as follows.

```
    function _checkForUpgrade(uint256 campaignId, uint256 epoch) internal {
        if (campaignUpgrade.totalRewardAmount != 0) {

            if (epoch == campaign.startTimestamp) {
    -            periodByCampaignId[campaignId][epoch].rewardPerPeriod =
                    campaignUpgrade.totalRewardAmount.mulDiv(1,
campaign.numberOfPeriods);

    +            periodByCampaignId[campaignId][epoch].rewardPerPeriod =
                    campaignUpgrade.totalRewardAmount.mulDiv(1,
campaignUpgrade.numberOfPeriods);

            } else {
                periodByCampaignId[campaignId][epoch - EPOCH_LENGTH].leftover +=
                    campaignUpgrade.totalRewardAmount - campaign.totalRewardAmount;
            }
        }
    }
```

Team response

Fixed.

Mitigation Review

The fix implements the recommendation.

Low severity findings

TRST-L-1 An incorrect endTimestamp can be used when claiming rewards

- **Category:** Logical flaws
- **Source:** Votemarket.sol
- **Status:** Acknowledged

Description

When users claim rewards, the validity of the claim is checked. If the claim is made outside the claim window, it is considered invalid. The **campaign.endTimestamp** is used to determine this validity.

```
    function _canClaim(ClaimData memory data) internal view returns (bool) {
        Campaign storage campaign = campaignById[data.campaignId];

        bool canClaimFromOracle = IOracleLens(ORACLE).isVoteValid(data.account,
campaign.gauge, data.epoch);

        bool withinClaimDeadline = campaign.endTimestamp + CLAIM_WINDOW_LENGTH >
block.timestamp;

        bool notClaimedOrNoReward =
totalClaimedByAccount[data.campaignId][data.epoch][data.account] == 0
            || periodByCampaignId[data.campaignId][data.epoch].rewardPerVote == 0;

        return canClaimFromOracle && withinClaimDeadline && notClaimedOrNoReward;
    }
```

However, if the manager increases the number of periods in some epochs and those epochs have not been updated, the **campaign.endTimestamp** will be incorrect.

Recommended mitigation

It is recommended to save the **endTimestamp** for campaigns particularly.

Team response

Acknowledged.

Additional recommendations

TRST-R-1 Redundant condition in _canClaim()

When **periodByCampaignId.rewardPerVote** is 0, *_canClaim()* is true. The comment says to allow users to claim new rewards. However, the current implementation does not change **rewardPerVote** once it is determined, so this condition is redundant.

```
        // 4. Check if the account has not claimed before or if there's a new reward
available.
        bool notClaimedOrNoReward =
totalClaimedByAccount[data.campaignId][data.epoch][data.account] == 0
-           || periodByCampaignId[data.campaignId][data.epoch].rewardPerVote == 0;
```

TRST-R-2 Leftover is not sent to hook when totalVotes is 0

When **totalVotes** is 0 the leftover reward is rolled over to the next epoch directly, however when **hook** is not **address(0)**, the leftover reward is sent to hook. This inconsistency may confuse users. It is recommended to document the behavior to inform users.

```
        // 2. If no votes, rollover the leftover to the next epoch.
        if (totalVotes == 0) {
            period.leftover = period.rewardPerPeriod;
            return;
        }

        Campaign storage campaign = campaignById[campaignId];

        // 2. Calculate reward per vote
        uint256 rewardPerVote = period.rewardPerPeriod.mulDiv(1e18, totalVotes);

        // 3. Cap reward per vote at max reward per vote
        if (rewardPerVote > campaign.maxRewardPerVote) {
            rewardPerVote = campaign.maxRewardPerVote;

            // 4. Calculate leftover rewards
            uint256 leftOver = period.rewardPerPeriod -
rewardPerVote.mulDiv(totalVotes, 1e18);

            // 5. Handle leftover rewards
            address hook = campaign.hook;
            if (hook != address(0)) {
```

TRST-R-3 The chain id is not considered in the Oracle

A campaign is associated with a specific gauge and chain ID.

```
    struct Campaign {
        uint256 chainId;
        address gauge;
    }
```

However, the Oracle does not take the chain ID into account.

Votes should be related to both the gauge and the chain ID.

TRST-R-4 Some tokens cannot transfer 0 amounts

Tokens like the **LEND** token cannot transfer 0 amounts.

In the *_transferTokens*() function, the **feeAmount** can be 0.

If this function reverts due to a zero transfer amount, users may be unable to claim their rewards.

```
    function _transferTokens(ClaimData memory data) internal {
        address rewardToken = campaignById[data.campaignId].rewardToken;

        SafeTransferLib.safeTransfer(rewardToken, data.receiver, data.amountToClaim);

        SafeTransferLib.safeTransfer(rewardToken, feeCollector, data.feeAmount);
    }
```

Centralization risks

TRST-CR-1 Platform requires privileged users to determine snapshot blocks

The protocol relies on privileged user to determine the block to snapshot for each epoch and call *Oracle.insertBlockNumber()*. If the privileged user does not work, users will not be able to claim rewards because they cannot insert data into the *Oracle* through *Verifier*.

Systemic risks

TRST-SR-1 Incompatibility with rebase and fee-on-transfer tokens

The reward token amount for a campaign is the token amount sent by the manager to the contract, this does not apply to rebase tokens and fee-on-transfer tokens. When these are used by campaigns, the protocol functionality will work incorrectly due to a token amount mismatch.

TRST-SR-2 The snapshot block should be close to the start time of the epoch in the Oracle

Privileged user snapshots may be of any block within the epoch.

If the user votes between the epoch start time and the snapshot block, the user's **lastVote** will be this time. In this case, *isVoteValid()* will return **false**, causing the user to be unable to claim the reward for this epoch.

Therefore, the snapshot block should be taken as close as possible the epoch start time.