Paladin Security Review

Reviewers

Gerard Persoon, Lead Security Researcher

Jay Jonah8, Security Researcher

DefSec, Security Researcher

Devansh Batham, Apprentice

May 3, 2022

1 Executive Summary

Over the course of 9 days in total, Paladin engaged with Spearbit to review Warden-Quests.

We found a total of 48 issues with Paladin Warden-Quests.

Repository	Commit
Warden-Quest	3694c2963acdcadc190113f83c3f1492fe84cca9

Summary

Type of Project	Governance Lending Protocol, DeFi
Timeline	Apr 4 - Apr 13, 2022
Methods	Manual Review
Documentation	Medium
Testing Coverage	Medium-High

Total Issues

Critical Risk	0
High Risk	2
Medium Risk	7
Low Risk	11
Gas Optimizations	17
Informational	11

Contents

2 Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at Spearbit.com

3 Introduction

Paladin is a decentralized, non-custodial governance lending protocol where users can either loan the voting power in their governance token, or borrow some voting power. Depositors stake governance tokens or derivatives that grant voting power in exchange for yield, while borrowers can leverage their voting power to gain more influence temporarily.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Paladin according to the specific commit by a three person team. Any modifications to the code will require a new security review.

4 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

4.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies

4.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

4.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

5 Findings

5.1 High Risk

5.1.1 Verify user has indeed voted

Severity: *High Risk*

Context: MultiMerkleDistributor.sol

Description: If an error is made in the merkle trees (either by accident or on purpose) a user that did not vote (in that period for that gauge) might get rewards assigned to him.

Although the Paladin documentation says: "*the Curve DAO contract does not offer a mapping of votes for each Gauge for each Period*", it might still be possible to verify that a user has voted if the account, gauge and period are known.

Note: Set to high risk because the likelihood of this happening is medium, but the impact is high.

Recommendation: Check that a user has voted by interrogating the gauge contracts at reward retrieval time.

Paladin: We carefully considered this issue along the development cycle and the main argument against the recommendation is as follows:

If users want to pile up rewards in order to claim them all at once (e.g. because of gas fees), then the only vote we can fetch from the Curve Gauge Controller is the last vote from the user since the previous ones were removed, and no checkpoints of past votes exists in the Gauge Controller. That would mean user past claims would be locked, and never claimed. Because of that, we are trying to have the most trustworthy MerkleTree generation, so this type of issue does not appear.

Spearbit: Acknowledged, recommendation not implemented therefore risk still exists.

5.1.2 Tokens could be sent / withdrawn multiple times by accident

Severity: *High Risk*

Context: QuestBoard.sol#L678-L815

Description: Functions `closeQuestPeriod()` and `closePartOfQuestPeriod()` have similar functionality but interfere with each other.

1. Suppose you have closed the first quest of a period via `closePartOfQuestPeriod()`. Now you cannot use `closeQuestPeriod()` to close the rest of the periods, as `closeQuestPeriod()` checks the state of the first quest.
2. Suppose you have closed the second quest of a period via `closePartOfQuestPeriod()`, but `closeQuestPeriod()` continues to work. It will close the second quest again and send the rewards of the second quest to the distributor, again. Also, function `closeQuestPeriod()` sets the `withdrawableAmount` value one more time, so the creator can do `withdrawUnusedRewards()` once more.

Although both `closeQuestPeriod()` and `closePartOfQuestPeriod()` are authorized, the problems above could occur by accident.

Additionally there is a lot of code duplication between `closeQuestPeriod()` and `closePartOfQuestPeriod()`, with a high risk of issues with future code changes.

```
function closeQuestPeriod(uint256 period) external isAlive onlyAllowed nonReentrant {
...
// We use the 1st QuestPeriod of this period to check it was not Closed
uint256[] memory questsForPeriod = questsByPeriod[period];
require(
    periodsByQuest[questsForPeriod[0]][period].currentState == PeriodState.ACTIVE, // only checks first
↪   period
    "QuestBoard: Period already closed"
);
... // no further checks on currentState
_questPeriod.withdrawableAmount = ....    // sets withdrawableAmount (again)
IERC20(_quest.rewardToken).safeTransfer(distributor, toDistributeAmount); // sends tokens (again)
...
}
```

```
function closePartOfQuestPeriod(uint256 period, uint256[] calldata questIDs) external isAlive
↪   onlyAllowed nonReentrant {
...
_questPeriod.currentState = PeriodState.CLOSED;
...
_questPeriod.withdrawableAmount = _questPeriod.rewardAmountPerPeriod - toDistributeAmount;
IERC20(_quest.rewardToken).safeTransfer(distributor, toDistributeAmount);
...
}
```

Note: Set to high risk because the likelihood of this happening is medium, but the impact is high.

Recommendation: Let function `closeQuestPeriod()` call `closePartOfQuestPeriod()`. Make `closePartOfQuestPeriod()` more robust by:

- Checking the status of each period.
- Skipping the already closed periods.

Paladin: Implemented in #9.

Spearbit: Acknowledged.

5.2 Medium Risk

5.2.1 Limit possibilities of `recoverERC20()`

Severity: *Medium Risk*

Context: MultiMerkleDistributor.sol#L296-L300, QuestBoard.sol#L986-L991

Description: Function `recoverERC20()` in contract `MultiMerkleDistributor.sol` allows the retrieval of all ERC20 tokens from the `MultiMerkleDistributor.sol` whereas the comment indicates it is only meant to retrieve those tokens that have been sent by mistake.

Allowing to retrieve all tokens also enables the retrieval of legitimate ones. This way rewards cannot be collected anymore. It could be seen as allowing a rug pull by the project and should be avoided.

In contrast, function `recoverERC20()` in contract `QuestBoard.sol` does prevent whitelisted tokens from being retrieved.

Note: The project could also add a merkle tree that allows for the retrieval of legitimate tokens to their own addresses.

```
* @notice Recovers ERC20 tokens sent by mistake to the contract
contract MultiMerkleDistributor is Ownable {
    function recoverERC20(address token, uint256 amount) external onlyOwner returns(bool) {
        IERC20(token).safeTransfer(owner(), amount);
        return true;
    }
}
```

```
contract QuestBoard is Ownable, ReentrancyGuard {
    function recoverERC20(address token, uint256 amount) external onlyOwner returns(bool) {
        require(!whitelistedTokens[token], "QuestBoard: Cannot recover whitelisted token");
        IERC20(token).safeTransfer(owner(), amount);
        return true;
    }
}
```

Recommendation: Prevent the retrieval of legitimate tokens. Because it is not possible to enumerate `questRewardToken[]` to identify legitimate tokens, an extra data structure is needed. Also be aware of dual entry point tokens.

Paladin: Implemented in #16.

Spearbit: Acknowledged.

5.2.2 Updating `QuestBoard` in `MultiMerkleDistributor.sol` will not work

Severity: *Medium Risk*

Context: MultiMerkleDistributor.sol#L285-L287

Description: Updating `QuestManager`/ `QuestBoard` in `MultiMerkleDistributor.sol` will give the following issue:

If the `newQuestBoard` uses the current implementation of `QuestBoard.sol`, it will start with `questId == 0` again, thus attempting to overwrite previous quests.

```
function updateQuestManager(address newQuestBoard) external onlyOwner {
        questBoard = newQuestBoard;
    }
```

Recommendation: Confirm the usefulness of updating the `QuestManager`/ `QuestBoard`. If this functionality is relevant, adapt `QuestBoard.sol` to start with a higher value for `nextID`.

Paladin: The `QuestBoard` should be unique (only replaced if the GaugeController is replaced or in case of a bug requiring to kill the contract), while the Distributor could be replaced to propose new ways to redeem the reward tokens.

`QuestBoard` is now immutable, therefore this method is not needed anymore.

Implemented in #16.

Spearbit: Acknowledged.

5.2.3 Old quests can be extended via `increaseQuestDuration()`

Severity: *Medium Risk*

Context: QuestBoard.sol#L380-L438

Description: Function `increaseQuestDuration()` does not check if a quest is already in the past. Extending a quest from the past in duration is probably not useful. It also might require additional calls to `closePartOfQuestPeriod()`.

```
function increaseQuestDuration(...) ... {
    updatePeriod();
    ...
    uint256 lastPeriod = questPeriods[questID][questPeriods[questID].length - 1];
    ...
    uint256 periodIterator = ((lastPeriod + WEEK) / WEEK) * WEEK;
    ...
   for(uint256 i = 0; i < addedDuration;){
        ...
        periodsByQuest[questID][periodIterator]....= ...
        periodIterator = ((periodIterator + WEEK) / WEEK) * WEEK;
        unchecked{ ++i; }
    }
    ...
}
```

Recommendation: Determine what the actions should be when the quest is in the past.

Paladin: We check that when calling the function, the current period is either the last period of the Quest, or before that last period. If not, the Quest is over, and we revert. Implemented in #8.

Spearbit: Acknowledged.

5.2.4 Accidental call of `addQuest` could block contracts

Severity: *Medium Risk*

Context: MultiMerkleDistributor.sol#L240, QuestBoard.sol#L276-L369

Description: The `addQuest()` function uses an `onlyAllowed` access control modifier. This modifier checks if `msg.sender` is `questBoard` or `owner`. However, the `QuestBoard.sol` contract has a `QuestID` registration and a token whitelisting mechanism which should be used in combination with `addQuest()` function. If `owner` accidentally calls `addQuest()`, the `QuestBoard.sol` contract will not be able to call `addQuest()` for that `questID`. As soon as `createQuest()` tries to add that same `questID` the function will revert, becoming uncallable because `nextID` still maintains that same value.

```
function createQuest(...) ... {
    ...
    uint256 newQuestID = nextID;
    nextID += 1;
    ...
    require(MultiMerkleDistributor(distributor).addQuest(newQuestID, rewardToken), "QuestBoard: Fail
↪   add to Distributor");
    ...
}
```

```
function addQuest(uint256 questID, address token) external onlyAllowed returns(bool) {
    require(questRewardToken[questID] == address(0), "MultiMerkle: Quest already listed");
    require(token != address(0), "MultiMerkle: Incorrect reward token");
    // Add a new Quest using the QuestID, and list the reward token for that Quest
    questRewardToken[questID] = token;
    emit NewQuest(questID, token);
    return true;
}
```

Note: Set to medium risk because the likelihood of this happening is low, but the impact is high.

Recommendation: Replace the modifier on `addQuest()` to a modifier like `onlyQuestBoard()`:

```
modifier onlyQuestBoard(){
    require(msg.sender == questBoard, "MultiMerkle: Not allowed");
    _;
}
```

Paladin: Choice to put only a require instead of a 1 time use modifier. Implemented in #5.

Spearbit: Acknowledged.

5.2.5 Reduce impact of `emergencyUpdatequestPeriod()`

Severity: *Medium Risk*

Context: MultiMerkleDistributor.sol#L311-L322

Description: Function `emergencyUpdatequestPeriod()` allows the merkle tree to be updated.

The merkle tree contains an embedded index parameter which is used to prevent double claims. When the merkleRoot is updated, the layout of indexes in the merkle tree could become different.

Example:

```
Suppose the initial merkle tree contains information for:
- user A: index=1, account = 0x1234, amount=100
- user B: index=2, account = 0x5689, amount=200
Then user A claims => _setClaimed(..., 1) is set.

Now it turns out a mistake is made with the merkle tree, and it should contain:
- user B: index=1, account = 0x5689, amount=200
- user C: index=2, account = 0xabcd, amount=300

Now user B will not be able to claim because bit 1 has already been set.
```

Under this situation the following issues can occur:

- Someone who has already claimed might be able to claim again.
- Someone who has already claimed has too much.
- Someone who has already claimed has too little, and cannot longer claim the rest because `_setClaimed()` has already been set.
- someone who has not yet claimed might not be able to claim because `_setClaimed()` has already been set by another user.

Note: Set to medium risk because the likelihood of this happening is low, but the impact is high.

Recommendation: The following steps can be taken to reduce the impact:

- Only allow an update if no claims have been done yet, by checking no bits have been set yet. However this has limited use.

- Make sure the index number of a user always stays the same, although this doesn't help if the user is entitled to an higher amount.
- Exclude already claimed tokens from the new merkle tree. To prevent claiming in the mean time, it might be a good idea to pause the claiming to make sure no claims are done in the mean time. Reset all the bits after the update.
- Consider storing how much each account has claimed and allow the account to claim less on future claims, in case the account has claimed too much. However this requires some complicated logic.

Note: The contract needs to store the size of the merkle tree (e.g. the largest index) to be able to check/reset all the bits.

Paladin: In the case where a new amount of token is transferred to the contract to cover losses from wrong calculated claims, we might need to change that amount to allow them to claim. Added a parameter `addedRewards` to increase `questRewardsPerPeriod`, but never decrease it. Implemented in #16.

Explanation for the emergency update procedure:

1. Block claims for the Quest period by using this method to set an incorrect MerkleRoot, where no proof matches the root.
2. Prepare a new Merkle Tree, taking into account previous user claims on that period and missing/overpaid rewards. a) For all new claims to be added, set them after the last index of the previous Merkle Tree. b) For users that did not claim, keep the same index and adjust the amount to be claimed if needed. c) For indexes that were claimed, place an empty node in the Merkle Tree (with an amount at `0` & the address `0xdead` as the account).
3. Update the Quest period with the correct MerkleRoot (no need to change the Bitmap, as the new MerkleTree will account for the indexes already claimed).

Spearbit: Acknowledged. Implemented in part technically and procedurally.

5.2.6 Verify the correct merkle tree is used

Severity: *Medium Risk*

Context: MultiMerkleDistributor.sol#L260-L275, balance-tree.ts#L30-L35, MultiMerkleDistributor.sol#L126-L144

Description: The `MultiMerkleDistributor.sol` contract does not verify that the merkle tree belongs to the right quest and period. If the wrong merkle tree is added then the wrong rewards can be claimed.

Note: Set to medium risk because the likelihood of this happening is low, but the impact is high.

Recommendation: A solution which does not cost any extra storage but requires a small amount of gas involves adding `questid` and `period` into the merkle tree nodes, assuring rewards can only be claimed in combination with the right `questid` and `period`.

```
balance-tree.ts:

// keccak256(abi.encode(index, account, amount))
public static toNode(index: number | BigNumber, account: string, amount: BigNumber): Buffer {
    return Buffer.from(
-       utils.solidityKeccak256(['uint256', 'address', 'uint256'], [index, account, amount]).substr(2),
+       utils.solidityKeccak256(['uint256', 'uint256', 'uint256', 'address', 'uint256'],
+                               [ questID, period, index, account, amount]).substr(2),
      'hex'
    )
}
```

```
MultiMerkleDistributor.sol:

function claim(uint256 questID, uint256 period, uint256 index, address account, uint256 amount, ...)
↪  public {
    ...
    // Check that the given parameters match the given Proof
-    bytes32 node = keccak256(abi.encodePacked(index, account, amount));
+    bytes32 node = keccak256(abi.encodePacked(questID, period, index, account, amount));
    ...
}
```

Paladin: Implemented in #12.

Spearbit: Acknowledged.

5.2.7 Prevent mixing rewards from different quests and periods

Severity: *Medium Risk*

Context: MultiMerkleDistributor.sol#L260-L275

Description: The `MultiMerkleDistributor.sol` contract does not verify that the sum of all amounts in the merkle tree are equal to the rewards allocated for that quest and for that period. This could happen if there is a bug in the merkle tree creation script.

If the sum of the amounts is too high, then tokens from other quests or other periods could be claimed, which will give problems later on, when claims are done for the other quest/periods.

Note: Set to medium risk because the likelihood of this happening is low, but the impact is high.

Recommendation: Consider making token buckets per quest (or even per period) in the `MultiMerkleDistributor` contract.

Paladin: Implemented in #16.

Spearbit: Acknowledged.

5.3 Low Risk

5.3.1 Nonexistent zero address check for newQuestBoard in updateQuestManager function

Severity: *Low Risk*

Context: MultiMerkleDistributor.sol#L285-L287

Description: Nonexistent zero address check for `newQuestBoard` in `updateQuestManager` function. Assigning `newQuestBoard` to a zero address may cause unintended behavior.

Recommendation: Introduce a check for a `0x` addresss.

```
function updateQuestManager(address newQuestBoard) external onlyOwner {
+   require(newQuestBoard != address(0), "newQuestBoard: Zero Address");
    questBoard = newQuestBoard;
}
```

Paladin: Implemented in #16.

Spearbit: Acknowledged.

5.3.2 Verify `period` is always a multiple of `week`

Severity: *Low Risk*

Context: QuestBoard.sol#L201-L203, QuestBoard.sol#L678-L742, QuestBoard.sol#L750-L815, QuestBoard.sol#L843-L845, QuestBoard.sol#L854-L863, MultiMerkleDistributor.sol#L126-L144, MultiMerkleDistributor.sol#L185-L216, MultiMerkleDistributor.sol#L260-L275, MultiMerkleDistributor.sol#L311-L322

Description: The calculations with `period` assume that `period` is a multiple of `WEEK`. However, `period` is often assigned as a parameter and not verified if it is a multiple of `WEEK`. This calculation may cause unexpected results.

Note: When it is verified that `period` is a multiple of `WEEK`, the following calculation can be simplified:

```
- int256 nextPeriod = ((period + WEEK) / WEEK) * WEEK;
+ int256 nextPeriod = period + WEEK;
```

The following function does not explicitly verify that `period` is a multiple of `WEEK`.

```
function closeQuestPeriod(uint256 period) external isAlive onlyAllowed nonReentrant {
    ...
    uint256 nextPeriod = ((period + WEEK) / WEEK) * WEEK;
    ...
}
function getQuestIdsForPeriod(uint256 period) external view returns(uint256[] memory) { ... }
function closePartOfQuestPeriod(uint256 period, uint256[] calldata questIDs) ... { ... }
function addMerkleRoot(uint256 questID, uint256 period, bytes32 merkleRoot) ... { ... }
function addMultipleMerkleRoot(..., uint256 period, ...) external isAlive onlyAllowed nonReentrant {
↪   ... }
function claim(..., uint256 period, ...) public { ... }
function updateQuestPeriod(uint256 questID, uint256 period, bytes32 merkleRoot) ... { ... }
function emergencyUpdatequestPeriod(uint256 questID, uint256 period, bytes32 merkleRoot) ... { ... }

function claimQuest(address account, uint256 questID, ClaimParams[] calldata claims) external {
    ... // also uses period as part of the claims array
    require(questMerkleRootPerPeriod[claims[i].questID][claims[i].period] != 0, "MultiMerkle: not
↪   updated yet");
    require(!isClaimed(questID, claims[i].period, claims[i].index), "MultiMerkle: already claimed");
    ...
    require(
        MerkleProof.verify(claims[i].merkleProof, questMerkleRootPerPeriod[questID][claims[i].period],
↪   node),
        "MultiMerkle: Invalid proof"
    );
    ...
    _setClaimed(questID, claims[i].period, claims[i].index);
    ...
    emit Claimed(questID, claims[i].period, claims[i].index, claims[i].amount, rewardToken, account);
    ...
}
```

Recommendation: Consider assuing that `period` is a multiple of `WEEK`. This can be achieved in the following way:

```
uint256 period = (period / WEEK) * WEEK;
```

If this is done everywhere then consider making this change:

```
- ((period + WEEK) / WEEK) * WEEK;
+  period + WEEK;
```

Paladin: Implemented in #13. Changes not made for the claim methods, as this type of issue where period is incorrect will be handled by changes made in #7.

Spearbit: Acknowledged.

5.3.3 Missing safety check to ensure array length does not underflow and revert

Severity: *Low Risk*

Context: QuestBoard.sol#L235-L242, QuestBoard.sol#L380-L438, QuestBoard.sol#L448-L505, QuestBoard.sol#L515-L574

Description: Several functions use `questPeriods[questID][questPeriods[questID].length - 1]`. The second value in the `questPeriods` mapping is `questPeriods[questID].length - 1`. It is possible for this function to revert if the case arises where `questPeriods[questID].length` is `0`. Looking at the code this is not likely to occur but it is a valid safety check that covers possible strange edge cases.

```
function _getRemainingDuration(uint256 questID) internal view returns(uint256) {
    ...
    uint256 lastPeriod = questPeriods[questID][questPeriods[questID].length - 1];
    ...
}

function increaseQuestDuration(...) ... {
    ...
    uint256 lastPeriod = questPeriods[questID][questPeriods[questID].length - 1];
    ...
}

 function increaseQuestReward(...) ... {
    ...
    uint256 lastPeriod = questPeriods[questID][questPeriods[questID].length - 1];
    ...
}

function increaseQuestObjective(... ) ... {
    ...
    uint256 lastPeriod = questPeriods[questID][questPeriods[questID].length - 1];
    ...
}
```

Recommendation: Add a require to check the length of `questPeriods[questID]`:

```
+   require(questPeriods[questID].length > 0, "Array Underflow");
    uint256 lastPeriod = questPeriods[questID][questPeriods[questID].length - 1];
```

Paladin: Implemented in #8.

Spearbit: Acknowledged.

5.3.4 Prevent dual entry point tokens

Severity: *Low Risk*

Context: QuestBoard.sol#L986-L991

Description: Function `recoverERC20()` in contract `QuestBoard.sol` only allows the retrieval of non whitelisted tokens. Recently an issue has been found to circumvent these checks, with so called dual entry point tokens. See a description here: compound-tusd-integration-issue-retrospective

```
function recoverERC20(address token, uint256 amount) external onlyOwner returns(bool) {
        require(!whitelistedTokens[token], "QuestBoard: Cannot recover whitelisted token");
        IERC20(token).safeTransfer(owner(), amount);
        return true;
    }
```

Recommendation: Make sure dual entry point tokens are not whitelisted in the protocol.

Paladin: It is noted in our methodology to check if an ERC20 can be whitelisted as a reward token in the protocol to avoid this type of issue.

Spearbit: Acknowledged.

5.3.5 Limit the creation of quests

Severity: *Low Risk*

Context: QuestBoard.sol#L201-L203, QuestBoard.sol#L276-L369, QuestBoard.sol#L870-L879

Description: The function `getQuestIdsForPeriod()` could run out of gas if someone creates an enormous amount of quests. See also: what-is-the-array-size-limit-of-a-returned-array.

Note: If this were to happen, the `QuestId`s can also be retrieved directly from the getter of `questsByPeriod()`.

Note: `closeQuestPeriod()` has the same problem, but `closePartOfQuestPeriod()` is a workaround for this.

Requiring a minimal amount of tokens to create a quest can limit the number of quests. The minimum number of tokens to pay is: `duration * minObjective * minRewardPerVotePerToken[]`. The values of `duration` and `minObjective` are least `1`, but `minRewardPerVotePerToken[]` could be `0` and even if `minRewardPerVotePerToken` is non zero but still low, the number of tokes required is neglectable when using tokens with 18 decimals.

Requiring a minimum amount of tokens also helps to prevent the creation of spam quests.

```
function getQuestIdsForPeriod(uint256 period) external view returns(uint256[] memory) {
     return questsByPeriod[period]; // could run out of gas
}

function createQuest(...) {
    ...
    require(duration > 0, "QuestBoard: Incorrect duration");
    require(objective >= minObjective, "QuestBoard: Objective too low");
    ...
    require(rewardPerVote >= minRewardPerVotePerToken[rewardToken], "QuestBoard: RewardPerVote too
↪   low");
    ...
    vars.rewardPerPeriod = (objective * rewardPerVote) / UNIT; // can be 0 ==> totalRewardAmount can be
↪   0
    require((totalRewardAmount * platformFee)/MAX_BPS == feeAmount, "QuestBoard: feeAmount incorrect");
↪   // feeAmount can be 0
    ...
    require((vars.rewardPerPeriod * duration) == totalRewardAmount, "QuestBoard: totalRewardAmount
↪   incorrect");
    ...
    IERC20(rewardToken).safeTransferFrom(vars.creator, address(this), totalRewardAmount);
    IERC20(rewardToken).safeTransferFrom(vars.creator, questChest, feeAmount);
    ...
}

constructor(address _gaugeController, address _chest){
    ...
    minObjective = 1000 * UNIT; // initial value, but can be overwritten
    ...
}

function updateMinObjective(uint256 newMinObjective) external onlyOwner {
    require(newMinObjective > 0, "QuestBoard: Null value");  // perhaps set higher
    minObjective = newMinObjective;
}
function whitelistToken(address newToken, uint256 minRewardPerVote) public onlyAllowed { // geen
↪   isAlive???
    ...
    minRewardPerVotePerToken[newToken] = minRewardPerVote; // no minimum value required
    ...
}
```

Recommendation: Consider setting (higher) minimal values for `minRewardPerVotePerToken[]` and `minObjective`.

Paladin: The parameters for minObjective and minRewardPerVote will be carefully calculated off-chain and can be increased. Fake Quests would be skipped by using `closePartOfQuestPeriod()`.

Spearbit: Acknowledged (solution based on procedures, not technical).

5.3.6 Non existing states are considered active

Severity: *Low Risk*

Context: QuestBoard.sol#L750-L815

Description: The function `closePartOfQuestPeriod()` verifies if the state of `periodsByQuest[questIDs[i]][period]` is active. However, if a state is checked of a non existing `questIDs[i]` or a `questID` that has no quest in that period, then `periodsByQuest[questIDs[i]][period]` is empty and `periodsByQuest[questIDs[i]][period].currentState == 0`.

As `PeriodState.ACTIVE ==0`, the stated is considered to be active and the `require()` doesn't trigger and processing continues.

Luckily as all other values are also `0` (especially `_questPeriod.rewardAmountPerPeriod`), `toDistributeAmount` will be 0 and no tokens are sent. However slight future changes in the code might introduce unwanted effects.

```
enum PeriodState { ACTIVE, CLOSED, DISTRIBUTED } // ACTIVE == 0

function closePartOfQuestPeriod(uint256 period, uint256[] calldata questIDs) external isAlive
↪  onlyAllowed nonReentrant {
        ...
        for(uint256 i = 0; i < length;){
            ...
            require(
                periodsByQuest[questIDs[i]][period].currentState == PeriodState.ACTIVE, // doesn't work
↪  if questIDs[i] & period are empty
                "QuestBoard: Period already closed"
            );
```

Recommendation: Don't use a value of `0` as a valid state, for example in the following way:

```
- enum PeriodState { ACTIVE, CLOSED, DISTRIBUTED }
+ enum PeriodState { ZERO, ACTIVE, CLOSED, DISTRIBUTED }
```

Note: this also requires setting the `ACTIVE` state explicitly in functions `createQuest()` and `increaseQuestDuration()`. Alternatively use `periodsByQuest[ questIDs[i] ][period].periodStart !=0` to verify the struct is filled.

Paladin: Implemented in #10.

Spearbit: Acknowledged.

5.3.7 Critical changes should use two-step process

Severity: *Low Risk*

Context: Ownable.sol#L62, QuestBoard#L27, MultiMerkleDistributor#L24 and QuestTreasureChest#L23

Description: The `QuestBoard.sol`, `QuestTreasureChest.sol` and `QuestTreasureChest.sol` contracts inherit from OpenZeppelin's `Ownable` contract which enables the `onlyOwner` role to transfer ownership to another address. It's possible that the `onlyOwner` role mistakenly transfers ownership to the wrong address, resulting in a loss of the `onlyOwner` role. This is an unwanted situation because the owner role is neccesary for several methods.

Recommendation: Consider implementing a two step process where the owner nominates an account and the nominated account needs to call an `acceptOwnership()` function for the transfer of ownership to fully succeed. This ensures the nominated `EOA` account is a valid and active account.

Paladin: Implemented in #14.

Spearbit: Acknowledged.

5.3.8 Prevent accidental call of `emergencyUpdatequestPeriod()`

Severity: *Low Risk*

Context: MultiMerkleDistributor.sol#L260-L275, MultiMerkleDistributor.sol#L311-L322

Description: Functions `updateQuestPeriod()` and `emergencyUpdatequestPeriod()` are very similar. However, if function `emergencyUpdatequestPeriod()` is accidentally used instead of `updateQuestPeriod()`, then `period` isn't `push()`ed to the array `questClosedPeriods[]`. This means function `getClosedPeriodsByQuests()` will not be able to retreive all the closed periods.

```
function updateQuestPeriod(uint256 questID, uint256 period, bytes32 merkleRoot) external onlyAllowed
↪  returns(bool) {
    ...
    questClosedPeriods[questID].push(period);
    ...
    questMerkleRootPerPeriod[questID][period] = merkleRoot;
    ...
}
```

```
function emergencyUpdatequestPeriod(uint256 questID, uint256 period, bytes32 merkleRoot) external
↪  onlyOwner returns(bool) {
    ... // no push()
    questMerkleRootPerPeriod[questID][period] = merkleRoot;
    ...
}
```

Recommendation: Make sure that `emergencyUpdatequestPeriod()` can only be called after `updateQuestPeriod()` has been called by adding the following check to `emergencyUpdatequestPeriod()`:

```
require(questMerkleRootPerPeriod[questID][period] !=0, "MultiMerkle: Not closed yet");
```

Paladin: Implemented in #4.

Spearbit: Acknowledged.

5.3.9 Usage of deprecated `safeApprove`

Severity: *Low Risk*

Context: QuestTreasureChest.sol#L53

Description: OpenZeppelin `safeApprove` implementation is deprecated. Reference. Using this deprecated function can lead to unintended reverts and potential locking of funds. SafeERC20.safeApprove() Insecure Behaviour.

Recommendation: Consider replacing `safeApprove()` with `safeIncreaseAllowance()` or `safeDecreaseAllowance()` instead regarding Openzeppelin comment.

Paladin: Implemented in #3.

Spearbit: Acknowledged.

5.3.10 `questID` on the `NewQuest` event should be `indexed`

Severity: *Low Risk*

Context: File.sol#L129

Description: The `NewQuest` event currently does not have `questID` set to `indexed` which goes against the pattern set by the other events in the contract where `questID` is actually `indexed`.

Recommendation: Add `indexed` to `questID` on the `NewQuest` event to aline with all other events in the file in which `questID` is `indexed`.

```
    event NewQuest(
-       uint256 questID,
+       uint256 indexed questID
        address indexed creator,
        address indexed gauge,
        address rewardToken,
        uint48 duration,
        uint256 startPeriod,
        uint256 objectiveVotes,
        uint256 rewardPerVote
    );
```

Paladin: Implemented in #2.

Spearbit: Acknowledged.

5.3.11 Add validation checks on addresses

Severity: *Low Risk*

Context: QuestBoard.sol#L182, MultiMerkleDistributor.sol#L81, MultiMerkleDistributor.sol#L126, MultiMerkleDistributor.sol#L185

Description: Missing validation checks on addresses passed into the `constructor` functions. Adding these checks on `_gaugeController` and `_chest` can prevent costly errors the during deployment of the contract.

Also in function `claim()` and `claimQuest()` there is no zero check for for `account` argument.

Recommendation: Consider doing the following:

In the constructor of `QuestBoard`, ensure that `_gaugeController` and `_chest` are non zero addresses and also that they are unique from one another.

```
contract QuestBoard is Ownable, ReentrancyGuard {
    constructor(address _gaugeController, address _chest){
+       require(_gaugeController != address(0), "Zero Address");
+       require(_chest != address(0), "Zero Address");
+       require(_gaugeController != _chest, "Duplicate address");
        ...
    }
}
```

```
contract MultiMerkleDistributor is Ownable {
    constructor(address _questBoard){
+       require(questBoard != address(0), "Zero Address");
        questBoard = _questBoard;
    }
}
```

In functions `claim()` and `claimQuest()` add:

```
+   require(account!= address(0), "Zero Address");
```

Paladin: Implemented in #1 and #17.

Spearbit: Acknowledged.

5.4 Gas Optimization

5.4.1 Changing public `constant` variables to non-public can save gas

Severity: *Gas Optimization*

Context: QuestBoard.sol#L38 , QuestBoard.sol#L36, QuestBoard.sol#L34

Description: Several constants are public and thus have a getter function. It is unlikely for these values to be called from the outside, therefore it is not necessary to make them public.

Recommendation: Make constants that do not need to be accessible from the outside `private`:

```
-    uint256 public constant WEEK = 604800;
+    uint256 private constant WEEK = 604800;

-    uint256 public constant UNIT = 1e18;
+    uint256 private constant UNIT = 1e18;

-    uint256 public constant MAX_BPS = 10000;
+    uint256 private constant MAX_BPS = 10000;
```

Paladin: Implemented in #7.

Spearbit: Acknowledged.

5.4.2 Using `uint` instead of `bool` to optimize gas usage

Severity: *Gas Optimization*

Context: QuestTreasureChest.sol#L27

Description: A `bool` is more costly than `uint256`. Because each write action generates an additional SLOAD to read the contents of the slot, change the bits occupied by `bool` and finally write back.

```
contract BooleanTest {
    mapping(address => bool) approvedManagers;

    // Gas Cost : 44144
    function approveManager(address newManager) external{
        approvedManagers[newManager] = true;
    }

    mapping(address => uint256) approvedManagersWithoutBoolean;

    // Gas Cost : 44069
    function approveManagerWithoutBoolean(address newManager) external{
        approvedManagersWithoutBoolean[newManager] = 1;
    }
}
```

Recommendation: Consider changing `bool` definitions with `uint`.

Paladin: Acknowledged, but will not make the changes.

Spearbit: Acknowledged (not implemented).

5.4.3 Optimize `&&` operator usage

Severity: *Gas Optimization*

Context: QuestBoard.sol#L292, QuestBoard.sol#L297, QuestBoard.sol#L389, QuestBoard.sol#L457

Description: The check `&&` consumes more gas than using multiple require statements. Example test can be seen below:

```
   //Gas Cost: 22515
   function increaseQuestReward(uint256 newRewardPerVote, uint256 addedRewardAmount, uint256 feeAmount)
↪   public {
        require(newRewardPerVote != 0 && addedRewardAmount != 0 && feeAmount != 0, "QuestBoard: Null
↪   amount");

   }
  //Gas Cost: 22477
  function increaseQuestRewardTest(uint256 newRewardPerVote, uint256 addedRewardAmount, uint256
↪   feeAmount) public {
        require(newRewardPerVote != 0, "QuestBoard: Null amount");
        require(addedRewardAmount != 0, "QuestBoard: Null amount");
        require(feeAmount != 0, "QuestBoard: Null amount");
   }
```

Note : It costs more gas to deploy but it is worth it after X calls. Trade-offs should be considered.

Recommendation: Consider using multiple `require` statements instead of `&&` operator.

Paladin: Considering the gas cost diff, and the diff of gas for deploy, and the fact that those functions might not used too many times (mainly to adapt the Quest parameters, but in case the Quest has a small duration, recreating a new one would be easier for users), this change was not implemented.

Spearbit: Acknowledged (not implemented).

5.4.4 Unnecesary value set to `0`

Severity: *Gas Optimization*

Context: QuestBoard.sol#L713

Description: Since all default values in solidity are already `0` it is unnecessary to include `_questPeriod.rewardAmountDistributed = 0;` here as it should already be 0.

Recommendation: Consider removing `_questPeriod.rewardAmountDistributed = 0;` and adding a comment it is already `0`.

Paladin: Implemented in #9.

Spearbit: Acknowledged.

5.4.5 Optimize unsigned integer comparison

Severity: *Gas Optimization*

Context: QuestBoard.sol#L295, QuestBoard.sol#L390, QuestBoard.sol#L975

Description: Check `!= 0` costs less gas compared to `> 0` for unsigned integers in require statements with the optimizer enabled. While it may seem that `> 0` is cheaper than `!=0` this is only true without the optimizer being enabled and outside a require statement. If the optimizer is enabled at `10k` and it is in a require statement, it would be more gas efficient.

Recommendation: Change `> 0` comparison with `!= 0`.

Paladin: Implemented in #7.

Spearbit: Acknowledged.

5.4.6 Use memory instead of storage in `closeQuestPeriod()` and `closePartOfQuestPeriod()`

Severity: *Gas Optimization*

Context: QuestBoard.sol#L678-L815

Description: In functions `closeQuestPeriod()` and `closePartOfQuestPeriod()` a storage pointer `_quest` is set to `quests[questsForPeriod[i]]`. This is normally used when write access to the location is need. Nevertheless `_quest` is read only, to a copy of `quests[questsForPeriod[i]]` is also sufficient. This can save some gas.

```
function closeQuestPeriod(uint256 period) external isAlive onlyAllowed nonReentrant {
    ...
    Quest storage _quest = quests[questsForPeriod[i]];
    ...
    gaugeController.checkpoint_gauge(_quest.gauge);  // read only access of _quest
    ...
    uint256 periodBias = gaugeController.points_weight(_quest.gauge, nextPeriod).bias; // read only
↪   access of _quest
    ...
    IERC20(_quest.rewardToken).safeTransfer(distributor, toDistributeAmount); // read only access of
↪   _quest
    ...
}

function closePartOfQuestPeriod(uint256 period, uint256[] calldata questIDs) external isAlive
↪   onlyAllowed nonReentrant {
    ...
    Quest storage _quest = quests[questIDs[i]];
    ...
    gaugeController.checkpoint_gauge(_quest.gauge); // read only access of _quest
    ...
    uint256 periodBias = gaugeController.points_weight(_quest.gauge, nextPeriod).bias; // read only
↪   access of _quest
    ...
    IERC20(_quest.rewardToken).safeTransfer(distributor, toDistributeAmount); // read only access of
↪   _quest
    ...
}
```

Recommendation: Replace `storage` with `memory` in the definition of `_quest`.

```
-   Quest storage _quest = quests[questIDs[i]];
+   Quest memory _quest = quests[questIDs[i]];
```

Paladin: Implemented in #7.

Spearbit: Acknowledged.

5.4.7 Revert string size optimization

Severity: *Gas Optimization*

Context: QuestBoard.sol#L303, QuestBoard.sol#L987

Description: Shortening revert strings to fit in 32 bytes will decrease deploy time gas and will decrease runtime gas when the revert condition has been met. Revert strings using more than 32 bytes require at least one additional `mstore`, along with additional operations for computing memory offset.

Recommendation: Shorten the revert strings to fit in 32 bytes. Alternatively, the code could be modified to use custom errors, introduced in Solidity 0.8.4.

Paladin: Changed from revert strings to Custom Errors in #15.

Spearbit: Acknowledged.

5.4.8 Optimize `withdrawUnusedRewards()` and `emergencyWithdraw()` with pointers

Severity: *Gas Optimization*

Context: QuestBoard.sol#L582-L667

Description: Functions `withdrawUnusedRewards()` and `emergencyWithdraw()` use `periodsByQuest[questID][_questPeriods[i]]` several times. It is possible to set a pointer to this record and use that pointer to read and update values. This will save gas and also make the code more readable.

```
function withdrawUnusedRewards(uint256 questID, address recipient) external isAlive nonReentrant {
    ...
    if(periodsByQuest[questID][_questPeriods[i]].currentState == PeriodState.ACTIVE) {
        ...
    }
    ...
    uint256 withdrawableForPeriod = periodsByQuest[questID][_questPeriods[i]].withdrawableAmount;
    ...
    if(withdrawableForPeriod > 0){
        ...
        periodsByQuest[questID][_questPeriods[i]].withdrawableAmount = 0;
    }
    ...
}

function emergencyWithdraw(uint256 questID, address recipient) external nonReentrant {
    ...
    if(periodsByQuest[questID][_questPeriods[i]].currentState != PeriodState.ACTIVE){
        uint256 withdrawableForPeriod = periodsByQuest[questID][_questPeriods[i]].withdrawableAmount;
        ...
        if(withdrawableForPeriod > 0){
            ...
            periodsByQuest[questID][_questPeriods[i]].withdrawableAmount = 0;
        }
    } else {
    ..
    totalWithdraw += periodsByQuest[questID][_questPeriods[i]].rewardAmountPerPeriod;
    periodsByQuest[questID][_questPeriods[i]].rewardAmountPerPeriod = 0;
    }
    ...
}
```

Recommendation: Create a pointer, for example `qp`:

```
QuestPeriod storage qp = periodsByQuest[questID][_questPeriods[i];
```

And replace `periodsByQuest[questID][_questPeriods[i]` with `qp`.

Paladin: Implemented in #7.

Spearbit: Acknowledged.

5.4.9 Needless to initialize variables with default values

Severity: *Gas Optimization*

Context: MultiMerkleDistributor.sol#L168, MultiMerkleDistributor.sol#L193, MultiMerkleDistributor.sol#L189, QuestBoard.sol#L224, QuestBoard.sol#L330, QuestBoard.sol#L417, QuestBoard.sol#L491, QuestBoard.sol#L560, QuestBoard.sol#L588, QuestBoard.sol#L592, QuestBoard.sol#L639, QuestBoard.sol#L698, QuestBoard.sol#L765

Description: `uint256` variables are initialized to a default value of `0` per Solidity docs. Setting a variable to the default value is unnecessary.

Recommendation: Remove explicit initialization for default values.

Paladin: Implemented in #7.

Spearbit: Acknowledged.

5.4.10 Optimize the calculation of the `currentPeriod`

Severity: *Gas Optimization*

Context: QuestBoard.sol#L251-L255

Description: The retrieval of `currentPeriod` is relatively gas expensive because it requires an `SLOAD` instruction (100 gas) every time. Calculating `(block.timestamp / WEEK) * WEEK;` is cheaper (`TIMESTAMP: 2 gas`, `MUL: 5 gas`, `DIV: 5 gas`). Refer to evm.codes for more information.

Additionally, there is a risk that the call to `updatePeriod()` is forgotten although it does not happen in the current code.

```
function updatePeriod() public {
    if (block.timestamp >= currentPeriod + WEEK) {
        currentPeriod = (block.timestamp / WEEK) * WEEK;
    }
}
```

Note: it is also possible to do all calculations with `(block.timestamp / WEEK)` instead of `(block.timestamp / WEEK) * WEEK`, but as the Paladin project has indicated:"" *This currentPeriod is a timestamp, showing the start date of the current period, and based from the Curve system (because we want the same timestamp they have in the GaugeController).*"

Recommendation: Remove `updatePeriod()` and replace the use of `currentPeriod` with a call to a function like `getPeriod()`:

```
function getPeriod() public view returns(uint256) {
    return  (block.timestamp / WEEK) * WEEK;
}
```

Paladin: Implemented in#16.

Spearbit: Acknowledged.

5.4.11 Change `memory` to `calldata`

Severity: *Gas Optimization*

Context: MerkleProof.sol#L38, MerkleProof.sol#L22

Description: For the function parameters, it is often more optimal to have the reference location to be `calldata` instead of `memory`. Changing `bytes` to `calldata` will decrease gas usage. OpenZeppelin Pull Request

Recommendation: Change `memory` definition with `calldata` in the related code section.

Paladin: We decided not to change the OZ dependencies used in this project. However, if the PR linked in this issue is correctly tested, approved and merged before the release of Quest, an update (with the correct tests and checks) of the `MerkleProof.sol` dependency could be done.

Spearbit: Acknowledged.

5.4.12 Caching array length at the beginning of function can save gas

Severity: *Gas Optimization*

Context: MultiMerkleDistributor.sol#L167, MultiMerkleDistributor.sol#L192, QuestBoard.sol#L890, QuestBoard.sol#L857, QuestBoard.sol#L764

Description: Caching array length at the beginning of the function can save gas in the several locations.

```
function multiClaim(address account, ClaimParams[] calldata claims) external {
    require(claims.length != 0, "MultiMerkle: empty parameters");
    uint256 length = claims.length;  // if this is done before the require, the require can use "length"
    ...
}
```

Recommendation: Consider introducing `uint256 length = newToken.length;` in the first line, then `require()` could use `length`.

Paladin: Implemented in #7.

Spearbit: Acknowledged.

5.4.13 Check `amount` is greater than `0` to avoid calling `safeTransfer()` unnecessarily

Severity: *Gas Optimization*

Context: QuestTreasureChest.sol#L63-L65

Description: A check should be added to make sure `amount` is greater than `0` to avoid calling `safeTransfer()` unnecessarily.

Recommendation:

```
function transferERC20(address token, address recipient, uint256 amount) external onlyAllowed
↪  nonReentrant {
+    require(amount > 0, "Amount cannot be 0");
....
}
```

Paladin: Implemented in #7.

Spearbit: Acknowledged.

5.4.14 `Unchecked{++i}` is more efficient than `i++`

Severity: *Gas Optimization*

Context: QuestBoard.sol#L224, QuestBoard.sol#L316

Description: The function `getAllQuestPeriodsForQuestId` uses `i++` which costs more gas than `++i`, especially in a loop. Also, the `createQuest` function uses `nextID += 1` which costs more gas than `++nextID`. Finally the initialization of `i = 0` can be skipped, as `0` is the default value.

Recommendation: Use `++i` instead of `i++` to increment the value of an uint variable. Use `unchecked` where possible. Skip initialization to 0. Note: the `unchecked` pattern has been use elsewhere in the code too.

```
function getAllQuestPeriodsForQuestId(uint256 questId) external view returns(QuestPeriod[] memory) {
    ...
-   for(uint256 i = 0; i < nbPeriods; i++){
+   for(uint256 i; i < nbPeriods;){
        periods[i] = periodsByQuest[questId][questPeriods[questId][i]];
+       unchecked{ ++i; }
    }
    return periods;
 }
```

```
function createQuest(...) ... {
    ...
    uint256 newQuestID = nextID;
-   nextID += 1;
+   unchecked{ ++nextID; }
    ...
}
```

Paladin: Implemented in #7.

Spearbit: Acknowledged.

5.4.15 Could replace `claims[i].questID` with `questID`

Severity: *Gas Optimization*

Context: MultiMerkleDistributor.sol#L194-L195

Description: Could replace `claims[i].questID` with `questID` (as they are equal due to the check above)

Recommendation: Consider changing the code in the following way:

```
-    require(questMerkleRootPerPeriod[claims[i].questID][claims[i].period] != 0, "MultiMerkle: not
↪  updated yet");
+    require(questMerkleRootPerPeriod[questID][claims[i].period] != 0, "MultiMerkle: not updated yet");
```

Paladin: Implemented in #7.

Spearbit: Acknowledged.

5.4.16 Change function visibility from `public` to `external`

Severity: *Gas Optimization*

Context: QuestBoard.sol#L898

Description: The function `updateRewardToken` of the `QuestBoard` contract could be set external to save gas and improve code quality.

Recommendation: Consider changing the function in the following way:

```
- function updateRewardToken(address newToken, uint256 newMinRewardPerVote) public onlyAllowed
+ function updateRewardToken(address newToken, uint256 newMinRewardPerVote) external onlyAllowed
```

Paladin: Implemented in #7.

Spearbit: Acknowledged.

5.4.17 Functions `isClaimed()` and `_setClaimed()` can be optimized

Severity: *Gas Optimization*

Context: MultiMerkleDistributor.sol#L95-L113

Description: The functions `isClaimed()` and `_setClaimed()` of the contract `MultiMerkleDistributor` can be optimized to save gas. See OZ BitMaps for inspiration.

Recommendation: Consider changing the functions in the following way:

```
function isClaimed(uint256 questID, uint256 period, uint256 index) public view returns (bool) {
-     uint256 claimedWordIndex = index / 256;
-     uint256 claimedBitIndex = index % 256;
+     uint256 claimedWordIndex = index >> 8;
+     uint256 claimedBitIndex = index & 0xff;
      uint256 claimedWord = questPeriodClaimedBitMap[questID][period][claimedWordIndex];
      uint256 mask = (1 << claimedBitIndex);
-     return claimedWord & mask == mask;
+     return claimedWord & mask != 0;
}

function _setClaimed(uint256 questID, uint256 period, uint256 index) private {
-     uint256 claimedWordIndex = index / 256;
-     uint256 claimedBitIndex = index % 256;
+     uint256 claimedWordIndex = index >> 8;
+     uint256 claimedBitIndex = index & 0xff;

-     questPeriodClaimedBitMap[questID][period][claimedWordIndex] =
↪   questPeriodClaimedBitMap[questID][period][claimedWordIndex] | (1 << claimedBitIndex);
+     questPeriodClaimedBitMap[questID][period][claimedWordIndex] |= (1 << claimedBitIndex);
}
```

Paladin: Implemented in #7.

Spearbit: Acknowledged.

5.5 Informational

5.5.1 Missing events for owner only functions

Severity: *Informational*

Context: QuestBoard.sol#L914, QuestBoard.sol#L924, QuestBoard.sol#L934, QuestBoard.sol#L944, QuestBoard.sol#L954, QuestBoard.sol#L964, QuestBoard.sol#L974

Description: Several key actions are defined without event declarations. Owner only functions that change critical parameters can emit events to record these changes on-chain for off-chain monitors/tools/interfaces.

Recommendation: Add events to all owner functions that change critical parameters.

Paladin: Implemented in #6.

Spearbit: Acknowledged.

5.5.2 Use `nonReentrant` modifier in a consistent way

Severity: *Informational*

Context: MultiMerkleDistributor.sol#L126-L144, MultiMerkleDistributor.sol#L185-L216, MultiMerkleDistributor.sol#L296-L300

Description: The functions `claim()`, `claimQuest()` and `recoverERC20()` of contract `MultiMerkleDistributor` send tokens but don't have a `nonReentrant` modifier. All other functions that send tokens do have this modifier. Note: as the checks & effects patterns is used this is not really necessary.

```
 function claim(...) public {
    ...
    IERC20(rewardToken).safeTransfer(account, amount);
}

function claimQuest(...) external {
    ...
    IERC20(rewardToken).safeTransfer(account, totalClaimAmount);
}

function recoverERC20(address token, uint256 amount) external onlyOwner returns(bool) {
    IERC20(token).safeTransfer(owner(), amount);
    return true;
}
```

Recommendation: Consider adding a `nonReentrant` modifier to `claim()`, `claimQuest()` and `recoverERC20()` to be more consistent with the rest of the code.

Paladin: Implemented in #6.

Spearbit: Acknowledged.

5.5.3 Place struct definition at the beginning of the contract

Severity: *Informational* **Context:** QuestBoard.sol#L258, MultiMerkleDistributor.sol#L148

Description: Regarding Solidity Style Guide, the struct definition can move to the beginning of the contract.

Recommendation: Consider moving struct definition at the beginning of the contract.

Paladin: For CreateVars, as this struct is only used inside the createQuest method (to avoid the StackTooDeep error), the choice was made to put that Struct here for faster understanding. And for ClaimParams, as it is used as parameter for the next methods, the choice to place the Struct here was also made for better clarity in the variables layout.

Spearbit: Acknowledged.

5.5.4 Improve checks for past quests in `increaseQuestReward()` and `increaseQuestObjective()`

Severity: *Informational*

Context: QuestBoard.sol#L448-L574, QuestBoard.sol#L235-L242

Description: The functions `increaseQuestReward()` and `increaseQuestObjective()` check: `newRewardPerVote > periodsByQuest[questID][currentPeriod].rewardPerVote`. This is true when the quest is in the past (e.g. `currentPeriod` is outside of the quest range), because all the values will be `0`. Luckily execution is stopped at `_getRemainingDuration(questID)`, however it would be more logical to put this check near the start of the function.

```
function increaseQuestReward(...) ... {
    updatePeriod();
    ...
    require(newRewardPerVote > periodsByQuest[questID][currentPeriod].rewardPerVote, "QuestBoard: New
↪   reward must be higher");
    ...
    uint256 remainingDuration = _getRemainingDuration(questID);
    require(remainingDuration > 0, "QuestBoard: no more incoming QuestPeriods");
    ...
}
```

The function `_getRemainingDuration()` reverts when the quest is in the past, as `currentPeriod` will be larger than `lastPeriod`. The is not what you would expect from this function.

```
function _getRemainingDuration(uint256 questID) internal view returns(uint256) {
    uint256 lastPeriod = questPeriods[questID][questPeriods[questID].length - 1];
    return (lastPeriod - currentPeriod) / WEEK;  // can revert
}
```

Recommendation: In both the functions `increaseQuestReward()` and `increaseQuestObjective()`, move the check `_getRemainingDuration()` towards the beginning of the function.

Adapt `_getRemainingDuration()` to return `0` if the quest is in the past. Or add a comment if the revert is the preferred action.

Paladin: Implemented in #8.

Spearbit: Acknowledged.

5.5.5 Should make use of `token.balanceOf(address(this));` to recover tokens

Severity: *Informational*

Context: MultiMerkleDistributorsol#L296

Description: Currently when calling the `recoverERC20()` function there is no way to calculate what the proper `amount` should be without having to check the contracts balance of `token` before hand. This will require an extra step and can be easily done inside the function itself.

Recommendation: Consider using `token.balanceOf(address(this));` to obtain the `amount` of the `token` to be transferred. Then the `amount` argument will not need to be passed in manually as an argument. A check should be added that the `amount` to transfer is larger than 0 as well.

```
- function recoverERC20(address token, uint256 amount) external onlyOwner returns(bool) {
+ function recoverERC20(address token) external onlyOwner returns(bool) {

+   uint256 amount = token.balanceOf(address(this));
+   require(amount > 0, "Amount cannot be zero);
    IERC20(token).safeTransfer(owner(), amount);
    return true;
}
```

Paladin: Implemented in #6.

Spearbit: Acknowledged.

5.5.6 Floating `pragma` is set

Severity: *Informational*

Context: All Contracts.

Description: The current pragma Solidity directive is `^0.8.10`. It is recommended to specify a specific compiler version to ensure that the byte code produced does not vary between builds. Contracts should be deployed using the same compiler version/flags with which they have been tested. Locking the pragma (for e.g. by not using ^ in pragma solidity `0.8.10`) ensures that contracts do not accidentally get deployed using an older compiler version with known compiler bugs.

Recommendation: Lock the compiler to a specific version.

```
pragma solidity 0.8.10;
```

Paladin: Implemented in #6.

Spearbit: Acknowledged.

5.5.7 Deflationary reward tokens are not handled uniformly across the protocol

Severity: *Informational*

Context: QuestBoard.sol#L307, QuestBoard.sol#L403, QuestBoard.sol#L478, QuestBoard.sol#L546

Description: The code base does not support `rebasing/deflationary/inflationary` reward tokens whose balance changes during transfers or over time. The necessary checks include at least verifying the amount of tokens transferred to contracts before and after the actual transfer to infer any fees/interest.

Recommendation: Consider checking previous balance/after balance equals to amount for any `rebasing/inflation/deflation` reward tokens.

```
+   uint256 oldBalance = rewardToken.balanceOf(address(this));
    IERC20(rewardToken).safeTransferFrom(vars.creator, address(this), totalRewardAmount);
+   uint256 newBalance = rewardToken.balanceOf(address(this));
+   require(oldBalance + totalRewardAmount == newBalance,"Fee-on-transfer Tokens Not Supported");
```

Paladin: Since we have a whitelist for tokens that can be used for Quest rewards, this safeguard should act to prevent rebasing/inflation/deflation type of token (by possibly having to wrap them to be used for the rewards).

Spearbit: Acknowledged (solution based on procedures, not technical).

5.5.8 Typo on comment

Severity: *Informational*

Context: QuestBoard.sol#L231

Description: Across the codebase, there is a typo on the comment. The comment can be seen from the below.

```
    * @dev Returns the number of periods to come for a give nQuest
```

Recommendation: Consider correcting the typo and review the codebase to check for more to improve code readability.

```
-   * @dev Returns the number of periods to come for a give nQuest
+   * @dev Returns the number of periods to come for a given Quest
```

Paladin: Implemented in #6.

Spearbit: Acknowledged.

5.5.9 Require statement with `gauge_types` function call is redundant

Severity: *Informational*

Context: QuestBoard.sol#L293

Description: The `gauge_types` function of the `Curve` reverts when an invalid gauge is given as a parameter, the `QuestBoard: Invalid Gauge` error message will not be seen in the `QuestBoard` contract. The documentation can be seen from the Querying Gauge and Type Weights.

```
function createQuest(...) ... {
    ...
    require(IGaugeController(GAUGE_CONTROLLER).gauge_types(gauge) >= 0, "QuestBoard: Invalid Gauge");
    ...
}
```

Recommendation: Make sure you don't rely on the message "*QuestBoard: Invalid Gauge*". Consider removing the `require` statement.

Paladin: In the current state, working with the Curve Gauge Controller, this require is redundant, and should never be reached. But this was added, in case of future Quest deployements of copies of the Curve Gauge system, where that revert might have been modified, or the gauge type was customized, this require would act as expected to prevent creating quests for invalid Gauges (and so that, in case it's removed, it will not be forgotten when needed).

Spearbit: Acknowledged.

5.5.10 Missing `setter` function for the `GAUGE_CONTROLLER`

Severity: *Informational*

Context: QuestBoard.sol#L31

Description: The `GAUGE_CONTROLLER` address is immutable and set in the constructor. If `Curve` adds a new version of the gauge controller, the value of `GAUGE_CONTROLLER` cannot be updated and the contract `QuestBoard` needs to be deployed again.

```
address public immutable GAUGE_CONTROLLER;

constructor(address _gaugeController, address _chest){
    GAUGE_CONTROLLER = _gaugeController;
   ...
}
```

Recommendation: Consider defining a setter function for the `GAUGE_CONTROLLER`.

Paladin: In case it does happen, it might be because the Controller system (and then maybe the voting process) would be changed, requiring to deploy a new version of the QuestBoard. We prefer to re-deploy a new version of Quest in the unlikely case the Gauge Controller is replaced in the Curve ecosystem.

Spearbit: Acknowledged (solution based on redeploy).

5.5.11 Empty events emitted in `killBoard()` and `unkillBoard()` functions

Severity: *Informational*

Context: QuestBoard.sol#L160-162

Description: When an `event` is emitted, it stores the arguments passed in for the transaction logs. Currently the `Killed()` and `Unkilled()` events are emitted without any arguments passed into them defeating the purpose of using an `event`.

Recommendation: Consider using `block.timestamp` as the argument passed into the `Killed()` and `Unkilled()` events as the value is used in both functions they are emitted in.

```
- event Killed();
+ event Killed(uint256 _killedTime);

- event Unkilled();
+ event UnKilled(uint256 _unkilledTime);
```

Paladin: Implemented in #6.

Spearbit: Acknowledged.