



TreasureDAO A-1

Security Audit

August 22, 2022

Version 1.0.0

Presented by [OxMacro](#)



Table of Contents

- Introduction
- Overall Assessment
- Specification
- Source Code
- Issue Descriptions and Recommendations
- Security Levels Reference
- Disclaimer

Introduction

This document includes the results of the security audit for Treasure DAO's smart contract code as found in the section titled 'Source Code'. The security audit was performed by the Macro security team from June 20, 2022 to July 22, 2022.

The purpose of this audit is to review the source code of certain Treasure DAO Solidity contracts, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While Macro's review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.



Overall Assessment

The following is an aggregation of issues found by the Macro Audit team:

| | Severity | Count | Fixed | Acknowledged | Won't Do |
|--|------------------|-------|-------|--------------|----------|
| | High | 4 | 4 | - | - |
| | Medium | 5 | 5 | - | - |
| | Low | 7 | 7 | - | - |
| | Code Quality | 18 | 11 | 1 | 6 |
| | Gas Optimization | 5 | - | 1 | 4 |

Treasure DAO was quick to respond to these issues.

Specification

Our understanding of the specification was based on the following sources:



Source Code

The following source code was reviewed during the audit:

- **Repository:** [treasure-staking](#)
- **Commit Hash:** 19c1c232b88907ac62440f1c3a6543f22b25f2af

Specifically, we audited the following contracts within this repository:

| Contract | SHA256 |
|-------------------------------------------------|-------------------------------------------------------------------|
| harvester/Harvester.sol | b3ddb43736bb974750d7b035e70e0b40e0e6f9d84ced208079c38a4df15d7641 |
| harvester/HarvesterFactory.sol | fbf50539c1c29a2d1cc8094f5227edcc9d6348e0a5d13109a3ae9858d8c1a141 |
| harvester/Middleman.sol | d1b25e883fe3158c3732d3fbff804188c751811bfff3416b0cb37978c354ebde1 |
| harvester/NftHandler.sol | 81e4f35458a9f9ad1029593b01e79319b7d069877c5dbdc3f47e6e737630ae52 |
| harvester/interfaces/IExtractorStakingRules.sol | bcc4c924d1711f69159e6682069da2c294dc0ab9c5d169061497b9296c69b6eb |
| harvester/interfaces/IHarvester.sol | 6733514f4fe96d9acc37bd794137a1a0cafa196f5d68414c3973ba8c85d4fade |



| | |
|---------------------------------------------|------------------------------------------------------------------|
| harvester/interfaces/IHarvesterFactory.sol | 61bebb2f33efee7b2bc6ce247c7d22045301f1b87338bbd0b59c0d8b0eee4023 |
| harvester/interfaces/IMiddleman.sol | e1cbd6356e18232d3f35b9565af51e9f1a3b907a952eb7748fda001573b85bc6 |
| harvester/interfaces/INftHandler.sol | b4e4a89043bc4159881e5fb18e9149fb41b0d7bd84fa5866cfb17fe300c252c2 |
| harvester/interfaces/IPartsStakingRules.sol | c0d978c53b2ce0df967a3e9ad1747babe9a9339c94e12ef538208e6b5e67bb9f |
| harvester/interfaces/IStakingRules.sol | ebe01b23135912ee2a632314c495542b4c8dbfcae13697fe8c40e921a27adb36 |
| harvester/lib/Constant.sol | 43bd37cc3a1d3a3f6278152e56ad69d11cca53f8b372641535cff4ecf565a392 |
| harvester/rules/ExtractorStakingRules.sol | 616597784a5cc4ec27f9d9b3b365b8cd7439dc545815689c77a1767e32b98a7c |
| harvester/rules/LegionStakingRules.sol | 7ca128960f6d00ec1a3dbda9f8dc00ea8d6ac4bf720f820524439647fe35989e |
| harvester/rules/PartsStakingRules.sol | e1b15b309f316beabef88ef9b8074564256159ac9e5ff551dd2d461aa0388ea7 |
| harvester/rules/StakingRulesBase.sol | 881d6246ba488038b16a689955125bac7cd627d5d4808915aaeec253809b6437 |
| harvester/rules/TreasureStakingRules.sol | 697f1233ece7c9bbf4ef2d8d53aab1f78fe8d0048fda42a402840cecefa07fb8 |

Note: This document contains an audit solely of the Solidity contracts listed above. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including



Issue Descriptions and Recommendations

Click on an issue to jump to it, or scroll down to see them all.

- ⚡ Rewards distribution not working due to incorrect utilization boost factor
- ⚡ Boost factors are incorrectly combined
- ⚡ Harvesters with TreasureStakingRules configured cannot receive rewards
- ⚡ Unfair reward distribution between harvesters due to missed checkpoints in the case of NFTs
- ⚡ Rewards distribution incorrect in presence of disabled harvesters
- ⚡ Unclaimed rewards for disabled harvester stuck in Middleman
- ⚡ Use of setExtractorAddress can break Extractor staking
- ⚡ LegionStakingRules parameter changes result in accounting discrepancies
- ⚡ Disabling the existing NftConfig leads to stuck NFTs
- ⚡ Unnecessary updateRewards executions and LogUpdateRewards event emissions
- ⚡ unstakeNft uses transferFrom instead of safeTransferFrom
- ⚡ Particular order of operations leads to deposit positions that cannot be properly cleaned up
- ⚡ Harvester#getDepositTotalBoost calculation is incorrect
- ⚡ NftHandler should inherit from ERC721HolderUpgradeable
- ⚡ Reward distribution between harvesters is not checkpointed in the case of privileged actions.
- ⚡ Depending on maxStakable, methods of ExtractorStakingRules can go out of gas
- ⚡ Call to parent initializers should be executed with the highest priority



- Q-4 Move event declarations from contract implementations to interfaces
- Q-5 Implement corresponding interfaces for all StakingRules
- ~~Q-6~~ Remove unnecessary code
- Q-7 Avoid using modifiers when they are applied only once
- ~~Q-8~~ In NftHandler modifiers canStake and canUnstake should revert
- ~~Q-9~~ Rename canStake and canUnstake in IStakingRules
- ~~Q-10~~ Remove unused event declaration in the Harvester
- ~~Q-11~~ Unused _user argument in the NftHandler modifiers
- ~~Q-12~~ Use the checks-effects-interactions pattern in stakeNft and unstakeNft
- Q-13 Split ERC721 and ERC1155 handling into separate internal functions
- ~~Q-14~~ Use established conventions for error reporting
- Q-15 Avoid using function naming conventions for variables
- ~~Q-16~~ Use common base for all constants in LegionStakingRules
- Q-17 Improve code documentation
- ~~Q-18~~ Add user info to event parameters emitted by NftHandler
- G-1 Replace unnecessary calls to getNftBoost within NftHandler
- G-2 totalRewardsEarned is tracked unnecessarily in updateRewards
- G-3 In calculateVestedPrincipal, consider removing unnecessary condition
- G-4 External calls can be avoided by storing details in the contract itself
- G-5 Consider not using the Counters library for extractorCount variable

Security Level Reference



| Level | Description |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| High | <p>The issue poses existential risk to the project, and the issue identified could lead to massive financial or reputational repercussions.</p> <p>We highly recommend fixing the reported issue. If you have already deployed, you should upgrade or redeploy your contracts.</p> |
| Medium | <p>The potential risk is large, but there is some ambiguity surrounding whether or not the issue would practically manifest.</p> <p>We recommend considering a fix for the reported issue.</p> |
| Low | <p>The risk is small, unlikely, or not relevant to the project in a meaningful way.</p> <p>Whether or not the project wants to develop a fix is up to the goals and needs of the project.</p> |
| Code Quality | <p>The issue identified does not pose any obvious risk, but fixing it would improve overall code quality, conform to recommended best practices, and perhaps lead to fewer development issues in the future.</p> |
| Informational | <p>Warnings and things to keep in mind when operating the protocol. No immediate action required.</p> |
| Gas Optimizations | <p>The presented optimization suggestion would save an amount of gas significant enough, in our opinion, to be worth the development cost of implementing it.</p> |



Issue Details

⚠️ Rewards distribution not working due to incorrect utilization boost factor

| TOPIC | STATUS | IMPACT | LIKELIHOOD |
|-------|-------------------------|--------|------------|
| Spec | Fixed ↗ | High | High |

Rewards for registered Harvesters are calculated in proportion to their shares. Harvester share is determined by three factors: `harvesterTotalBoost`, `corruptionNegativeBoost` and `utilBoost`.

```
function getHarvesterEmissionsShare(address _harvester) public view returns (uint256) {
    uint256 harvesterTotalBoost = IHarvester(_harvester).nftHandler().getHarvesterTotalBoost();
    uint256 utilBoost = getUtilizationBoost(_harvester);
    uint256 corruptionNegativeBoost = getCorruptionNegativeBoost(_harvester);

    return harvesterTotalBoost * utilBoost / Constant.ONE * corruptionNegativeBoost;
}
```

Rewards distribution requires that all three factors are correct. In section 2.3.4 of the specification, utilization is defined as “the ratio between the amount of deposited MAGIC in a particular harvester over the total deposit capacity for that harvester.” Reaching specified levels by this ratio determines utilization boost factor.

However, current implementation calculates utilization in the following way:



```

uint256 magicTotalDeposits = _harvester(_harvester).magicTotalDeposits();

uint256 len = excludedAddresses.length();
for (uint256 i = 0; i < len; i++) {
    circulatingSupply -= magic.balanceOf(excludedAddresses.at(i));
}

uint256 rewardsAmount = magic.balanceOf(_harvester) - magicTotalDeposits;
circulatingSupply -= rewardsAmount;
if (circulatingSupply != 0) {
    util = magicTotalDeposits * Constant.ONE / circulatingSupply;
}
}

```

This implementation is incorrect because it is based on `circulatingSupply`, which represents total MAGIC token supply minus some of the token balances of well-known system contracts (e.g., the Treasury and Ecosystem Fund). Considering the fact that total MAGIC token supply is much larger than total deposit capacity for individual harvesters, utilization boost factor for all harvesters will always be 0. As a result, harvesters will not be able to claim their reward, while emitted rewards will be stuck in the Middleman contract.

Consider updating the `getUtilization` function to properly follow the specification for calculating utilization boost factor, which is dependent only on the amount of deposited magic and total deposit capacity for the particular harvester.

⚡ Boost factors are incorrectly combined

TOPIC
Spec

| STATUS | IMPACT | LIKELIHOOD |
|-------------------------|--------|------------|
| Fixed ↗ | High | High |



$$\text{UserBoostedDepositA} = \text{MagicDeposit} * (1 + \text{TimelockBoost} + \text{LegionsBoost} + \text{Treasur}$$

Moreover, the specification says the following:

Deposit Boosts for NFTs are **additive**, not multiplicative. For example, a user with 100 MAGIC staked for 2 weeks, plus 2x Genesis All Class and 3x Honeycombs, would have:

Base Deposit Boost of 1x

Additive modifiers of (10%) + (2 x 200%) + (3 x 15.78%)

Total Deposit Boost of $1 * (1 + 0.1 + 4 + 0.4734) = 5.573$

However, the Harvester.sol implementation does not follow the specification, and boost factors are combined in an entirely different way. In Harvester.sol, within the `deposit` function, lock boost is applied in a multiplicative way:

```
uint256 lockLpAmount = _amount + _amount * lockBoost / ONE;
```

This `lockLpAmount` is then provided as an arg to the `_recalculateGlobalLp` function:

```
_recalculateGlobalLp(msg.sender, _amount.toInt256(), lockLpAmount.toInt256());
```

In `_recalculateGlobalLp`, `userNftBoost` is also applied in a multiplicative way that, when simplified, looks like `lockLpAmount = lockLpAmount + lockLpAmount * userNftBoost`. See line 5 below:

```
function _recalculateGlobalLp(address _user, int256 _amount, int256 _lockLpAmount)
    GlobalUserDeposit storage userGlobalDeposit = getUserGlobalDeposit[_user];
    uint256 nftBoost = nftHandler.getUserBoost(_user);
    uint256 newGlobalLockLpAmount = (userGlobalDeposit.globalLockLpAmount.toInt256() +
```



```

userGlobalDeposit.globalDepositAmount = (userGlobalDeposit.globalDepositAmou
userGlobalDeposit.globalLockLpAmount = newGlobalLockLpAmount;
userGlobalDeposit.globalLpAmount = newGlobalLpAmount;
userGlobalDeposit.globalRewardDebt += globalLpDiff * accMagicPerShare.toInt2

totalLpToken = (totalLpToken.toInt256() + globalLpDiff).toUint256();

int256 accumulatedMagic = (newGlobalLpAmount * accMagicPerShare / ONE).toInt
pendingRewards = (accumulatedMagic - userGlobalDeposit.globalRewardDebt).toU
}

```

Overall, boost factors are combined in Harvester.sol according to the following formula, which — as you may notice — is different from the one defined in the specification:

$$\text{UserBoostedDepositA} = \text{MagicDeposit} * (1 + \text{LockBoost}) * (1 + \text{NftBoost})$$

As a result, the system exhibits incorrect behavior.

Consider updating the Harvester implementation to properly follow the specification. Boost factors must be summed up before they are applied to the deposited amount. Changes are required in at least the `deposit` and `_recalculateGlobalLp` functions.

RESPONSE BY TREASURE DAO:

The specification was updated to match the implementation behavior.

⚠️ Harvesters with TreasureStakingRules configured cannot receive rewards



TreasureStakingRules support only user boost; they do not support harvester boost. Therefore, when TreasureStakingRules are present on a particular harvester (alone or combined with other staking rules), the harvester boost factor, originating from TreasureStakingRules, should not affect total harvester boost. The boost should be 1, or any other outcome resulting from multiplying harvester boost factors that originate from other staking rules.

However, the following TreasureStakingRules implementation of the `getHarvesterBoost` function is incorrect:

```
function getHarvesterBoost() external pure returns (uint256) {
    // Treasure staking only boosts userBoost, not harvesterBoost
    return 0;
}
```

A harvester boost factor of 0, originating from TreasureStakingRules, affects total harvester boost calculation in the NftHandler function `getHarvesterTotalBoost`. The result: a total harvester boost of 0 whenever TreasureStakingRules is present. Consequently, this prevents a distribution of rewards given to the particular harvester.

```
function getHarvesterTotalBoost() public view returns (uint256 boost) {
    boost = Constant.ONE;

    for (uint256 i = 0; i < allAllowedNfts.length(); i++) {
        address _nft = allAllowedNfts.at(i);

        IStakingRules stakingRules = allowedNfts[_nft].stakingRules;

        if (address(stakingRules) != address(0)) {
            boost = boost * stakingRules.getHarvesterBoost() / Constant.ONE;
        }
    }
}
```



Consider updating the `getHarvesterBoost` function within `TreasureStakingRules` to the following implementation:

```
function getHarvesterBoost() external pure returns (uint256) {
    // Treasure staking only boosts userBoost, not harvesterBoost
    return Constant.ONE;
}
```

H-4 Unfair reward distribution between harvesters due to missed checkpoints in the case of NFTs

| TOPIC | STATUS | IMPACT | LIKELIHOOD |
|-----------------|-------------------------|--------|------------|
| Protocol Design | Fixed ↗ | High | High |

Rewards are collected as a payment stream from `MasterOfCoin` to `MiddleMan` and then distributed between `Harvesters` and `AtlasMine`, based on emission share. Whenever an action is made (`deposit`, `withdraw`, `harvestAll`), a call is made to `Middleman`; which calls `MasterOfCoin` and requests rewards until that time. These rewards are then distributed across harvesters on basis of their emission shares.

This emission share is calculated as:

$$\text{Harvester Mining Power} = \text{Parts} * \text{Legions} * \text{Extractors} * \text{Utilisation} * \text{Corruption}$$

$$\text{Harvester Emission Share} = \frac{\text{Harvester Mining Power}}{\text{Sum}(\text{Harvester Mining Power}(i)) + \text{Atlas Mining Power}}$$



Extractors and Corruption .

However, if an NFT is staked or unstaked, it is not being checkpointed. `stakeNFT` triggers `updateNFTBoost` (of harvester), which calls `_recalculateGlobalLp`, which checkpoints for user rewards inside harvester; but, as `updateReward` is not called, the reward is not checkpointed for harvesters. This results in unfair reward distribution.

Consider adding `updateReward` modifier to `updateNftBoost`, of the harvester, as it's done for deposits and withdrawals.

M-1 Rewards distribution incorrect in presence of disabled harvesters

| TOPIC | STATUS | IMPACT | LIKELIHOOD |
|-----------------|-------------------------|--------|------------|
| Protocol Design | Fixed ↗ | Medium | Low |

In `Middleman.sol`, rewards for harvesters are calculated proportionally to the individual harvester share of the total share.

To calculate each harvester's share, in `getHarvesterShares`, the system iterates through all of the harvesters registered in `HarvesterFactory`.

```
function getHarvesterShares(address _targetHarvester) public view returns (
    address[] memory allHarvesters,
    uint256[] memory harvesterShare,
    uint256 totalShare,
    uint256 targetIndex
) {
    allHarvesters = harvesterFactory.getAllHarvesters();
```



```

    harvesterShare[i] = getHarvesterEmissionsShare(allHarvesters[i]);
    totalShare += harvesterShare[i];

    if (allHarvesters[i] == _targetHarvester) {
        targetIndex = i;
    }
}

if (atlasMine != address(0) && atlasMineBoost != 0) {
    totalShare += atlasMineBoost;
}
}

```

In addition to providing details about all harvesters, HarvesterFactory offers capability to enable/disable an individual harvester.

```

function enableHarvester(IHarvester _harvester) external onlyRole(HF_DEPLOYER) {
    _harvester.enable();
}

function disableHarvester(IHarvester _harvester) external onlyRole(HF_DEPLOYER) {
    _harvester.disable();
}

```

It does this using corresponding functions in Harvester.sol.

```

function enable() external onlyFactory {
    disabled = false;
    emit Enable();
}

function disable() external onlyFactory {
    disabled = true;
    emit Disable();
}

```




We are looking to allow Guilds to fight over a Harvester in a game, and then have the outcome of the game disable the yield to the old harvester if the defending team loses, while we deploy a new Harvester for the winning guild.

However, Middleman.sol's rewards calculation includes enabled and disabled harvesters. As a result, the rewards of enabled harvesters will be smaller than expected in the presence of disabled harvesters — the extent of how much smaller depends on the proportional amount of harvester shares associated with the disabled harvesters that should have been excluded from consideration.

Consider filtering out disabled harvesters in the process of rewards calculation at Middleman.sol or HarvesterFactory.

M-2 Unclaimed rewards for disabled harvester stuck in Middleman

| TOPIC | STATUS | IMPACT | LIKELIHOOD |
|-----------------|-------------------------|--------|------------|
| Protocol Design | Fixed ↗ | Medium | Low |

In Middleman.sol, rewards calculation is triggered by an external call to the `distributeRewards` function. As part of this process, rewards accrue for each registered harvester within Middleman. Rewards are transferred, or pulled, to the individual harvester on request (more precisely, by the individual harvester calling `Middleman#requestRewards`). If this call is not triggered for a long period of time, one can expect a potentially significant amount of accrued rewards — meant for particular harvester — to become stored in Middleman. As part of the regular system operation, anyone can trigger `deposit`, `harvestAll`, or `withdrawPosition` to pull rewards from Middleman to Harvester.



updateRewards modifier. As a result, users/depositors of the particular harvester will be negatively affected because their share of earned rewards will be stuck in the Middleman contract.

```
modifier updateRewards() {
    uint256 lpSupply = totalLpToken;
    if (lpSupply > 0 && !disabled) {
        uint256 distributedRewards = factory.middleman().requestRewards();
        totalRewardsEarned += distributedRewards;
        accMagicPerShare += distributedRewards * ONE / lpSupply;
        emit LogUpdateRewards(distributedRewards, lpSupply, accMagicPerShare);
    }
}
-;
```

Consider updating updateRewards and pendingRewardsAll to remove the !disabled condition guard so that users can pull previously earned rewards from Middleman, even if their harvester is disabled.

~~M-3~~ Use of setExtractorAddress can break Extractor staking

| TOPIC | STATUS | IMPACT | LIKELIHOOD |
|-----------|-------------------------|--------|------------|
| Use Cases | Fixed ↗ | Medium | Low |

In ExtractorStakingRules, setExtractorAddress is a privileged function that enables admin to change the address of the ERC1155 contract, which manages tokens that are eligible to be used as Extractor tokens.

ExtractorStakingRules implementation has maxStakeable spots per NftConfig enabled within particular harvester. When these spots are filled and individual extractor tokens — which are staked — expire, the



The `replaceExtractor` function, as part of its implementation, contains the following code, which performs core processing:

```
(
    address user,
    uint256 replacedTokenId,
    uint256 replacedAmount
) = stakingRules.canReplace(msg.sender, _nft, _tokenId, _amount, _replacedSpotId

IERC1155(_nft).safeTransferFrom(msg.sender, address(this), _tokenId, _amount, by
ERC1155Burnable(_nft).burn(address(this), replacedTokenId, replacedAmount);

stakedNfts[user][_nft][replacedTokenId] -= replacedAmount;
stakedNfts[msg.sender][_nft][_tokenId] += _amount;
```

In normal circumstances, the system operates properly. However, when `setExtractorAddress` is invoked, it changes the ERC1155 contract address for acceptable tokens. And, when `replaceExtractor` is called with a new `_nft` contract address, the following results occur:

- `canReplace` executes successfully.
- `IERC1155(_nft).safeTransferFrom` executes successfully.
- `ERC1155Burnable(_nft).burn`, referencing the incorrect `_nft` contract, fails or — even worse — destroys the incorrect token from the new NFT contract address.
- Additionally, if the previous line executes without any errors, the following line may also cause revert, since the left side will evaluate to 0, resulting in an underflow error:
`stakedNfts[user][_nft][replacedTokenId] -= replacedAmount;`



Consider:

- removing the privileged method, such as `setExtractorAddress`, or
- updating the `replaceExtractor` function to handle this edge case properly.

M-4 LegionStakingRules parameter changes result in accounting discrepancies

| TOPIC | STATUS | IMPACT | LIKELIHOOD |
|-----------|-------------------------|--------|------------|
| Use Cases | Fixed ↗ | Medium | Medium |

In `LegionStakingRules`, `legionBoostMatrix`, `legionRankMatrix` and `legionWeightMatrix` are runtime configurable variables. These matrixes of parameters affect user boost factor, harvester boost factor (total rank) and per user level constraint (`weightStaked`).

```
function getUserBoost(address, address, uint256 _tokenId, uint256) external view
    ILegionMetadataStore.LegionMetadata memory metadata = legionMetadataStore.me
    return getLegionBoost(uint256(metadata.legionGeneration), uint256(metadata.l
}
```

```
function getLegionBoost(uint256 _legionGeneration, uint256 _legionRarity) public
    if (_legionGeneration < legionBoostMatrix.length && _legionRarity < legionBo
        return legionBoostMatrix[_legionGeneration][_legionRarity];
    }

    return 0;
}
```



In `NftHandler`, `getNftBoost` wraps call to `getUserBoost` on associated `LegionStakingRules` instance.

```
function getNftBoost(address _user, address _nft, uint256 _tokenId, uint256 _amo
    IStakingRules stakingRules = allowedNfts[_nft].stakingRules;

    if (address(stakingRules) != address(0)) {
        boost = stakingRules.getUserBoost(_user, _nft, _tokenId, _amount);
    }
}
```

In `NftHandler`, when user stakes an NFT token associated with `LegionStakingRules`, within `canStake` function system increments `getUserBoost` accumulator.

```
getUserBoost[msg.sender] += getNftBoost(msg.sender, _nft, _tokenId, _amount);
harvester.updateNftBoost(msg.sender);
```

Also, in the `canUnstake` function, the system correspondingly decrements the `getUserBoost` accumulator:

```
getUserBoost[msg.sender] -= getNftBoost(msg.sender, _nft, _tokenId, _amount);
harvester.updateNftBoost(msg.sender);
```

Notice that when `getNftBoost` has deterministic result for particular token, `stakeNft` followed by `unstakeNft` will result `getUserBoost[msg.sender]` having initial value of 0. And that is what the implementation implicitly assumes.

However, as previously described `LegionStakingRules` parameters are changeable. Therefore following two cases are also possible:



LegionStakingRules are updated so boost for particular token is reduced

In `unstakeNft` boost is $N-1$

`getUserBoost[msg.sender]` is 1, while user doesn't have staked NFTs

2. User cannot unstake previously staked Legion

In `stakeNft` boost is N

LegionStakingRules are updated so boost for particular token is increased

In `unstakeNft` boost is $N+1$

Following line reverts with underflow

```
getUserBoost[msg.sender] -= getNftBoost(msg.sender, _nft, _tokenId, _amount);
```

Similar edge cases and associated issues are also possible in LegionStakingRules `_canStake` and `_canUnstake` functions with regards to `totalRank` and `weightStaked`.

```
function _canStake(address _user, address, uint256 _tokenId, uint256) internal o
    staked++;
    totalRank += getRank(_tokenId);
    weightStaked[_user] += getWeight(_tokenId);

    if (weightStaked[_user] > maxLegionWeight) revert("MaxWeight()");
}
```

```
function _canUnstake(address _user, address, uint256 _tokenId, uint256) internal
    staked--;
    totalRank -= getRank(_tokenId);
    weightStaked[_user] -= getWeight(_tokenId);
}
```

Consider removing capability for changing LegionStakingRules parameters.

M-5 Disabling the existing NftConfig leads to stuck NFTs



In NftHandler, `setNftConfig` is a privileged action that allows admin to disable particular NftConfig:

```
function _setNftConfig(address _nft, NftConfig memory _nftConfig) internal {
    if (address(_nftConfig.stakingRules) != address(0)) {
        // it means we are adding _nft or updating its config
        // ignore return value in case we are just updating config
        allAllowedNfts.add(_nft);
    } else {
        if (!allAllowedNfts.remove(_nft)) revert("AlreadyDisallowed()");
        _nftConfig.supportedInterface = Interfaces.Unsupported;
    }

    allowedNfts[_nft] = _nftConfig;
    emit NftConfigSet(_nft, _nftConfig);
}
```

However, when that happens, users with staked NFTs will be left with no ability to unstake them. When users attempt to unstake their NFTs, `unstake` method execution will result in a revert, with the message `NftNotAllowed()`.

Consider updating the `unstakeNft` functionality to allow unstaking, even if a particular NFT contract is not currently allowed — or remove the capability to disable NFT configs.

←4 Unnecessary updateRewards executions and LogUpdateRewards event emissions

TOPIC

Use Cases

STATUS

Fixed [↗](#)

IMPACT

Low

LIKELIHOOD

Medium



`middleman().requestRewards()` . Currently, however, on each invocation, `LogUpdateRewards` event will be emitted and additional calculations will be performed which is unnecessary.

```
modifier updateRewards() {
    uint256 lpSupply = totalLpToken;
    if (lpSupply > 0 && !disabled) {
        uint256 distributedRewards = factory.middleman().requestRewards();
        totalRewardsEarned += distributedRewards;
        accMagicPerShare += distributedRewards * ONE / lpSupply;
        emit LogUpdateRewards(distributedRewards, lpSupply, accMagicPerShare);
    }
}
_;
```

Consider adding a guard and performing corresponding actions only if `distributedRewards ≠ 0`.

⚡️ unstakeNft uses transferFrom instead of safeTransferFrom

| TOPIC | STATUS | IMPACT | LIKELIHOOD |
|------------------|-------------------------|--------|------------|
| Coding Standards | Fixed 🔗 | Low | Low |

In `NftHandler.sol`, within the `unstakeNft` function, token is transferred to `msg.sender` in the following way (which does not perform a check if the receiver can handle the ERC721 token, in case it is a contract, not an EOA):

```
IERC721(_nft).transferFrom(address(this), msg.sender, _tokenId);
```




↳ Particular order of operations leads to deposit positions that cannot be properly cleaned up

| TOPIC | STATUS | IMPACT | LIKELIHOOD |
|-----------|-------------------------|--------|------------|
| Use Cases | Fixed ↗ | Low | Low |

In Harvester.sol, the `withdrawPosition` function calls another function to remove deposit position entry. Call is guarded by a condition that requires associated deposit and pending rewards to be withdrawn already (i.e., 0).

```
if (user.depositAmount == 0 && user.lockLpAmount == 0 && pendingRewards == 0) {
    _removeDeposit(msg.sender, _depositId);
}
```

However, if `withdrawPosition/withdrawAll` is called first, with `pendingRewards` still present for the particular user, `_removeDeposit` will not be executed because this guard condition will evaluate to be false.

In addition, on each followup attempt to call `withdrawPosition`, this code would not be reachable. The only case when the `_removeDeposit` function is executed is when `harvestAll` is called before `withdrawPosition`, since the `harvestAll` execution would result in `pendingRewards` being 0.

This may result in a continuous increase of the number of records in the `allUserDepositIds` variable. Consequently, functions that iterate through `allUserDepositIds`, such as `withdrawAll` and



Consider:

- implement cleanup functionality also in `harvestAll`, or
- prevent direct calls to `withdrawPosition/withdrawAll` so that it can be invoked only after `harvestAll`.

RESPONSE BY TREASURE DAO:

Pending rewards check was a leftover from the previous implementation.

⚠️ Harvester#getDepositTotalBoost calculation is incorrect

| TOPIC | STATUS | IMPACT | LIKELIHOOD |
|-------|-------------------------|--------|------------|
| Spec | Fixed 🔗 | Low | Low |

In `Harvester.sol`, `getDepositTotalBoost` is implemented in the following way:

```
function getDepositTotalBoost(address _user, uint256 _depositId) external view returns (uint256 lockBoost, ) = getLockBoost(userInfo[_user][_depositId].lock);
uint256 userNftBoost = nftHandler.getUserBoost(_user);
// see: `_recalculateGlobalLp`.
// `userNftBoost` multiplies lp amount that already has `lockBoost` added
// that's why we have to add `lockBoost * userNftBoost / ONE` for correct re
return lockBoost + userNftBoost + lockBoost * userNftBoost / ONE;
}
```

However, the boosted deposit amount calculation, as defined in section 2.3 of the specification, is:



Therefore, consider updating the final expression in `getDepositTotalBoost` to the following:

```
return 1 + lockBoost + userNftBoost
```

↳5 NftHandler should inherit from ERC721HolderUpgradeable

| TOPIC | STATUS | IMPACT | LIKELIHOOD |
|------------------|-------------------------|--------|------------|
| Coding Standards | Fixed ↗ | Low | Low |

NftHandler manages both ERC721 and ERC1155 tokens. Also, NftHandler properly advertises ERC1155 support by inheriting the `ERC1155HolderUpgradeable` contract. However, it does not do the same for ERC721. Thus, third party smart contracts, such as smart wallets, may be unable to transfer ERC721 tokens to the NftHandler contract because of built-in checks, which require NftHandler to properly advertise ERC721 token support.

Consider updating NftHandler to also inherit from `ERC721HolderUpgradeable`. Corresponding changes are also necessary in both `NftHandler#init` and `NftHandler#supportsInterface` functions.

↳6 Reward distribution between harvesters is not checkpointed in the case of privileged actions.

| TOPIC | STATUS | IMPACT | LIKELIHOOD |
|-----------------|-------------------------|--------|------------|
| Protocol Design | Fixed ↗ | Low | Medium |



Rewards between harvesters are decided based on emission shares, which is calculated from various boosts.

$$\text{Harvester Mining Power} = \text{Parts} * \text{Legions} * \text{Extractors} * \text{Utilisation} * \text{Corruption}$$

$$\text{Harvester Emission Share} = \frac{\text{Harvester Mining Power}}{\text{Sum (Harvester Mining Power (i))} + \text{Atlas Mining Power}}$$

Following boost factors are updatable for an admin.

- Drip corruption tokens to the particular harvester.
- Update `ExtractorStakingRules` parameters (such as a lifetime and token boost factor) through `setExtractorLifetime` and `setExtractorBoost`.
- Update `LegionStakingRules` parameters through `setLegionBoostMatrix`, `setLegionWeightMatrix`, `setLegionRankMatrix`, `setBoostFactor`.
- Update `PartsStakingRules` parameters (such as a boost factor) through `setBoostFactor`.

The preceding updates change emission shares for harvesters. However, if `distributeRewards` is not called before these admin actions, the middle checkpoint is missed. As a result system exhibits **an unfair reward distribution** behavior.

Consider making the call to `distributeRewards` mandatory before doing these admin actions.

↳ **Depending on maxStakable, methods of ExtractorStakingRules can go out of gas**



The following loop is executed for each extractor whenever an extractor is staked:

```
_canStake(address _user, address _nft, uint256 _tokenId, uint256 _amount) =>
    for (uint256 i = 0; i < _amount; i++) {
        uint256 spotId = extractorCount.current();
        stakedExtractor[spotId] = ExtractorData(_user, _tokenId, block.timestamp);
        extractorCount.increment();
    }
```

More importantly, whenever `totalBoost` is calculated for each harvester in

`Middleman#distributeRewards`, the following loop is executed for each extractor within each harvester:

```
function getExtractorsTotalBoost() public view returns (uint256 totalBoost) {
    for (uint256 i = 0; i < extractorCount.current(); i++) {
        if (isExtractorActive(i)) {
            totalBoost += extractorBoost[stakedExtractor[i].tokenId];
        }
    }
}
```

This loop is redundant and may go above the gas limit, depending on the `maxStakable` value.

`Middleman#distributeRewards` is a core system function that must be executed at least once per block; this function iterates through all staked extractor spots as part of an underlying execution. If the number of staked extractor spots becomes large enough, the system may not operate properly because executing `Middleman#distributeRewards` will be expensive to run or may halt due to an out-of-gas error.

Consider the following options to resolve this:

1. Define acceptable input range for the `_maxStakeable` parameter and add corresponding guards.



Q4 Call to parent initializers should be executed with the highest priority

| TOPIC | STATUS | IMPACT |
|--------------|-------------------------|--------|
| Code Quality | Fixed ↗ | Low |

In `NftHandler.sol`'s `init` (and similarly in `Harvester.sol`'s), the method calls to parent initializers `__AccessControlEnumerable_init()` and `__ERC1155Holder_init()`, which are placed at the end of the method after role operations have already been performed. This may have resulted in an invalid setup.

For this particular case, it is not a cause of concern because parent initializers are calls with no changes. However, to avoid issues in similar situations — ones with initializers that do perform their own setup — it is recommended to follow best practices and put initializers at the top of the `init` function.

Consider updating the function to the following:

```
function init(
    address _admin,
    address _harvester,
    address[] memory _nfts,
    INftHandler.NftConfig[] memory _nftConfigs
) external initializer {
    __AccessControlEnumerable_init();
    __ERC1155Holder_init();

    _setRoleAdmin(NH_ADMIN, NH_ADMIN);
    _grantRole(NH_ADMIN, _admin);

    harvester = IHarvester(_harvester);

    if (_nfts.length != _nftConfigs.length) revert("InvalidData()");
```



```
}
}
```

Q-2 Unnecessary code duplication

| TOPIC | STATUS | IMPACT |
|--------------|-------------------------|--------|
| Code Quality | Fixed ↗ | Low |

In Harvester.sol, the `withdrawAndHarvestAll` function has redundant code which is already implemented in the `withdrawAll` function.

```
function withdrawAndHarvestAll() public {
    harvestAll();
    // replace following with call to withdrawAll()
    uint256[] memory depositIds = allUserDepositIds[msg.sender].values();
    for (uint256 i = 0; i < depositIds.length; i++) {
        withdrawPosition(depositIds[i], type(uint256).max);
    }
}
```

Q-3 Important Harvester methods not defined in IHarvester interface

| TOPIC | STATUS | IMPACT |
|--------------|---------|--------|
| Code Quality | Wont Do | Low |

Following is the list of the methods missing in IHarvester interface



- withdrawAll
- harvestAll
- withdrawAndHarvestPosition
- withdrawAndHarvestAll
- getTimelockOptionIds
- getUserBoost
- getDepositTotalBoost
- getNftBoost
- getAllUserDepositIds, getAllUserDepositIdsLength
- getUserDepositCap
- getLockBoost, getVestingTime
- pendingRewardsAll
- calculateVestedPrincipal
- setNftHandler, setDepositCapPerWallet, setTotalDepositCap
- addTimelockOption, enableTimelockOption, disableTimelockOption
- setUnlockAll

Consider declaring all external and public functions (including public variables) in the corresponding interface with proper Natspec comments. Check other interfaces and make sure they declare all public/external methods for corresponding contracts.

Q-4 Move event declarations from contract implementations to interfaces



Events are part of contract interface rather than implementation. Consider moving all event declarations to corresponding interfaces. Document all declarations with corresponding Natspec comments. This applies to:

- Harvester
- NftHandler
- PartsStakingRules
- ExtractorStakingRules
- LegionStakingRules
- TreasureStakingRules

Q-5 Implement corresponding interfaces for all StakingRules

| TOPIC | STATUS | IMPACT |
|--------------|---------|--------|
| Code Quality | Wont Do | Low |

`ExtractorStakingRules` has the `IExtractorStakingRules` interface, which defines custom functionality. `PartsStakingRules`, `LegionStakingRules`, and `TreasureStakingRules` also contain custom functionality, but do not have their own corresponding interfaces.

Consider implementing interfaces for all contracts that inherit `StakingRulesBase` and define all public and external functions/variables.



| TOPIC | STATUS | IMPACT |
|--------------|-------------------------|--------|
| Code Quality | Fixed ↗ | Low |

`user.vestingLastUpdate` is not used at all.

Consider removing it along with other related, redundant code, such as `_vestedPrincipal` function.

Q-7 Avoid using modifiers when they are applied only once

| TOPIC | STATUS | IMPACT |
|--------------|---------|--------|
| Code Quality | Wont Do | Low |

Modifiers are helpful when they are used to ensure a particular check is enforced at various entry points. However, when there is only a single-entry-point modifier, usage does not add value; on the contrary, it only negatively affects code readability.

The following modifiers are used only once:

- `Middleman#runIfNeeded`
- `NftHandler#canStake`
- `NftHandler#canUnstake`

Consider inlining them into corresponding functions to improve code readability.



| TOPIC | STATUS | IMPACT |
|--------------|-------------------------|--------|
| Code Quality | Fixed ↗ | Low |

The modifiers `NftHandler#canStake` and `NftHandle#canUnstake` do not revert if the address of `stakingRules` is 0. Execution continues within corresponding calling functions (`stakeNft` and `unstakeNft`), despite having no purpose for doing that.

This implementation behavior, at the moment, does not lead to a particular security issue due to other checks. However, consider updating the corresponding modifiers/functions to revert/return early, if corresponding conditions are not satisfied, to avoid security issues in the future.

RESPONSE BY TREASURE DAO:

- `canStake` reverts when `stakingRules` are not set making it impossible to stake tokens without proper configuration.
- `canUnstake` does not revert so it's possible to unstake tokens when the configuration is missing.

🔗 Rename `canStake` and `canUnstake` in `IStakingRules`

| TOPIC | STATUS | IMPACT |
|--------------|-------------------------|--------|
| Code Quality | Fixed ↗ | Low |

In `IStakingRules`, `canStake` and `canUnstake` external methods are declared. All child contracts with various strategies for staking rules implement these two methods using strategy specific behavior for the staking and unstaking of NFT assets.



to incorrectly conclude that these two functions perform a set of checks, but do not update the state.

However, both of these methods update the contract-specific state.

Therefore, consider renaming these two methods to more properly represent their underlying behavior (e.g., `stake/unstake` , `doStake/doUnstake` , `processStake/processUnstake`).

~~Q-10~~ Remove unused event declaration in the Harvester

| TOPIC | STATUS | IMPACT |
|--------------|-------------------------|--------|
| Code Quality | Fixed ↗ | Low |

In the Harvester, the `UndistributedRewardsWithdraw` event is declared but never used:

```
event UndistributedRewardsWithdraw(address indexed to, uint256 amount);
```

Consider removing this event declaration.

~~Q-11~~ Unused `_user` argument in the NftHandler modifiers

| TOPIC | STATUS | IMPACT |
|--------------|-------------------------|--------|
| Code Quality | Fixed ↗ | Low |



Instead, `msg.sender` is used within these modifiers.

Consider updating the modifiers' implementation by removing the `_user` argument or replacing the reference to `msg.sender` with `_user`.

Q-12 Use the checks-effects-interactions pattern in stakeNft and unstakeNft

| TOPIC | STATUS | IMPACT |
|--------------|-------------------------|--------|
| Code Quality | Fixed ↗ | Low |

In the `NftHandler`, the `stakeNft` and `unstakeNft` function implementations do not follow the checks-effects-interactions pattern, nor do they feature re-entrancy guards. Currently, this does not result in identified security issues.

However, the `NftHandler` is meant to be upgradable. Therefore, to avoid security issues being introduced in the future, consider:

- updating the implementation of these methods, to follow the checks-effects-interactions pattern, or
- add re-entrancy protection mechanism, such as OZ's `ReentrancyGuardUpgradeable`.

Q-13 Split ERC721 and ERC1155 handling into separate internal functions

| TOPIC | STATUS | IMPACT |
|--------------|---------|--------|
| Code Quality | Went De | Low |



In the `NftHandler`, `stakeNft` and `unstakeNft` handle both ERC721 and ERC1155 assets within the same function. Code duplication is minimized in this approach at the cost of a less readable code.

Consider updating these functions to split the handling and processing of ERC721 and ERC1155 assets into different helper functions.

Q-14 Use established conventions for error reporting

| TOPIC | STATUS | IMPACT |
|--------------|-------------------------|--------|
| Code Quality | Fixed ↗ | Low |

Error reporting within this project uses an unconventional approach — it reverts with a string message, formatted in a way that resembles custom error.

```
// In ExtractorStakingRules.sol there is following which
// looks like custom errors but it is not
function _canUnstake(address, address, uint256, uint256) internal pure override
    revert("CannotUnstake()");
}
```

This is not the proper approach to generate custom errors. A previous code with the proper application of Custom Errors, feature introduced in Solidity 0.8.4, would look like following:

```
error CannotUnstake();

function _canUnstake(address, address, uint256, uint256) internal pure override
    revert CannotUnstake();
```



Moreover, within codebase, different conventions for checks and error reporting are used. For example, in the `NftHandler#validateInput` modifier, checks and error reporting are done in following way:

```
modifier validateInput(address _nft, uint256 _amount) {
    if (_nft == address(0)) revert("InvalidNftAddress()");
    if (_amount == 0) revert("NothingToStake()");
    _;
}
```

However, in the `ExtractorStakingRules#validateInput` modifier, checks and error reporting is implemented differently:

```
modifier validateInput(address _nft, uint256 _amount) {
    require(_nft == extractorAddress, "InvalidAddress()");
    require(_amount > 0, "ZeroAmount()");
    _;
}
```

Consider choosing a single approach — we recommend one relying on custom errors — and applying it consistently within the whole project.

Q-15 Avoid using function naming conventions for variables

TOPIC
Code Quality

STATUS IMPACT
Wont Do Low



- Harvester#getUserGlobalDeposit
- NftHandler#getUserBoost
- PartsStakingRules#getAmountStaked
- TreasureStakingRules#getAmountTreasuresStaked

Consider making these variables internal and implementing custom getters if particular function names are desired.

Q-16 Use common base for all constants in LegionStakingRules

| TOPIC | STATUS | IMPACT |
|--------------|-------------------------|--------|
| Code Quality | Fixed ↗ | Low |

In LegionStakingRules — within constructor legionWeightMatrix — values are defined with different bases. Some values use e18 as a base, whereas others use e17.

```
legionWeightMatrix = [
  // GENESIS
  // LEGENDARY, RARE, SPECIAL, UNCOMMON, COMMON, RECRUIT
  [uint256(120e18), uint256(40e18), uint256(15e18), uint256(20e18), uint256(10
  // AUXILIARY
  // LEGENDARY, RARE, SPECIAL, UNCOMMON, COMMON, RECRUIT
  [illegalWeight, uint256(55e17), illegalWeight, uint256(4e18), uint256(25e17)
  // RECRUIT
  // LEGENDARY, RARE, SPECIAL, UNCOMMON, COMMON, RECRUIT
  [illegalWeight, illegalWeight, illegalWeight, illegalWeight, illegalWeight,
];
```




In addition, consider using a constant value for a base to avoid typos. Instead of:

```
[uint256(600e16), uint256(200e16), uint256(75e16), uint256(100e16), uint256(50e1
```

You may use the following approach:

```
// add constant  
uint256 BASE_WEIGHT = 1e18;
```

```
[uint256(6 * BASE_WEIGHT), uint256(2 * BASE_WEIGHT), uint256(0.75 * BASE_WEIGHT)
```

Q-17 Improve code documentation

| TOPIC | STATUS | IMPACT |
|--------------|--------------|--------|
| Code Quality | Acknowledged | Low |

While some parts of the audited project are documented using Natspec comments, the majority of the project is missing them. Additionally, a better approach for handling documents through inheritance is only applied within one part of the code; see `IStakingRules.sol` and `StakingRulesBase.sol`.

Consider updating the project code to include Natspec comments for all public-facing functions and variables. Follow the same approach as the one already implemented in `IStakingRules.sol` and `StakingRulesBase.sol`.



- **IStakingRules**
Natspec comment for `IStakingRules#getUserBoost` and `StakingRules#getHarvesterBoost` should provide details related to the number precision of return values.
- **ExtractorStakingRules**
Incorrect natspec for `ExtractorStakingRules.extractorBoost`, as it should be `maps token Id => boost value`.

```
/// @dev maps address => token Id => boost value
mapping(uint256 => uint256) public extractorBoost;
```

Incorrect natspec for `ExtractorStakingRules.extractorCount`, as it should be similar to the following:

```
current number of extractor spots taken / next extractor spot
/// @dev latest spot Id
Counters.Counter public extractorCount;
```

Missing natspec for the `_user` and `_nft` params for the `IExtractorStakingRules.canReplace` method.
Missing natspec in `ExtractorStakingRules` for the following:
`setMaxStakeable`
`setExtractorBoost`
`setExtractorAddress`
`setExtractorLifetime`
`isExtractorActive`
`getExtractorCount`
`getExtractors` - partially
`getExtractorsTotalBoost` - partially
- **LegionStakingRules**
all public variables
all events
all public methods
- **ILegionILegionMetadataStore**
all functions
- **PartsStakingRules**
all public variables



- HarvesterFactory
 - all public variables
 - all events
 - all public/external functions
- NftHandler
 - all public variables
 - all events
 - all public/external functions
 - does not use `/// @inheritdoc INftHandler` for interface method implementations
- Middleman
 - all public variables
 - all events
 - all public/external functions
- IHarvester
 - all
- Harvester
 - all public variables
 - all events
 - all public/external functions
- TreasureStakingRules
 - all public variables
 - maxStakeablePerUser
 - getAmountTreasuresStaked
 - all events
 - MaxStakeablePerUser
 - all public/external functions
 - setMaxStakeablePerUser
 - getTreasureBoost

Add user info to event parameters emitted by NftHandler



The NftHandler emits the following events, as part of execution in `stakeNft`, `unstakeNft`, and `replaceNft` functions:

```
event Staked(address indexed nft, uint256 tokenId, uint256 amount);
event Unstaked(address indexed nft, uint256 tokenId, uint256 amount);
event Replaced(address indexed nft, uint256 tokenId, uint256 amount, uint256 rep
```

However, in each case, information about the user in relation to a particular action is missing.

Consider adding the new parameter `address indexed user` for the above events to facilitate off-chain event indexing and monitoring.

G-1 Replace unnecessary calls to `getNftBoost` within `NftHandler`

| TOPIC | STATUS | IMPACT |
|------------------|---------|--------|
| Gas optimization | Wont Do | Low |

The following line is in `stakeNft` :

```
getUserBoost[msg.sender] += getNftBoost(msg.sender, _nft, _tokenId, _amount);
```

Correspondingly, the following line is in `unstakeNft` :



In NftHandler, `getNftBoost` is implemented:

```
function getNftBoost(address _user, address _nft, uint256 _tokenId, uint256 _amount)
    IStakingRules stakingRules = allowedNfts[_nft].stakingRules;

    if (address(stakingRules) != address(0)) {
        boost = stakingRules.getUserBoost(_user, _nft, _tokenId, _amount);
    }
}
```

The only added value of the `getNftBoost` function is a guard check, in case `stakingRules` is not set.

However, at places where `getNftBoost` is called within `stakeNft` and `unstakeNft`, `stakingRules` cannot be 0. Therefore, call `getUserBoost` on `stakingRules` directly, instead of through `getNftBoost`, to avoid unnecessary checks.

For example:

```
// at the beginning of stakeNft or unstakeNft
IStakingRules stakingRules = allowedNfts[_nft].stakingRules;

// later
getUserBoost[msg.sender] += stakingRules.getUserBoost(msg.sender, _nft, _tokenId
```

G-2 totalRewardsEarned is tracked unnecessarily in updateRewards

TOPIC

STATUS

IMPACT



`updateRewards` is present in most call paths, so each save that is done in `updateRewards` matters. If the purpose is for usability, one can derive `totalRewardsEarned` from the event `LogUpdateRewards`.

This optimization saves 1 `SSTORE` per execution.

G-3 In `calculateVestedPrincipal`, consider removing unnecessary condition

| TOPIC | STATUS | IMPACT |
|------------------|---------|--------|
| Gas optimization | Wont Do | Low |

Following condition is always positive, therefore it may be removed.

```
if (amountWithdrawn < amountVested) {
    amount = amountVested - amountWithdrawn;
}
```

G-4 External calls can be avoided by storing details in the contract itself

| TOPIC | STATUS | IMPACT |
|------------------|--------------|--------|
| Gas optimization | Acknowledged | Low |

For example, in the case of the Harvesters `factory.middleman()` and `factory.magic()`, consider defining them in the original contract only if they are not going to be changed regularly.



G-5 Consider not using the Counters library for extractorCount variable

| TOPIC | STATUS | IMPACT |
|------------------|---------|--------|
| Gas optimization | Wont Do | Low |

Consider implementing uint256 counter without external library. This is not a very significant optimization, but it is redundant.

Disclaimer

Macro makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Macro specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Macro will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand by any other party. In no event will Macro be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Macro has been advised of the possibility of such damages.



report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Macro. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Macro is not responsible for the content or operation of such websites, and that Macro shall have no liability to your or any other person or entity for the use of third party websites. Macro assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.