



# BlockSec

## Security Audit Report for Popcorn

**Date:** Apr 18, 2023

**Version:** 1.0

**Contact:** [contact@blocksec.com](mailto:contact@blocksec.com)

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	About Target Contracts . . . . .	1
1.2	Disclaimer . . . . .	2
1.3	Procedure of Auditing . . . . .	2
1.3.1	Software Security . . . . .	2
1.3.2	DeFi Security . . . . .	3
1.3.3	NFT Security . . . . .	3
1.3.4	Additional Recommendation . . . . .	3
1.4	Security Model . . . . .	3
<b>2</b>	<b>Findings</b>	<b>5</b>
2.1	DeFi Security . . . . .	5
2.1.1	Unfair Charged Fee . . . . .	5
2.1.2	Limitations of Staking Contract Deployment for Vault Contract . . . . .	6
2.1.3	Lost Rewards in fundReward() . . . . .	7
2.1.4	Fee Shares Minted before highWaterMark Updated . . . . .	8
2.1.5	Limitation of Instant Reward Distribution in addRewardToken() . . . . .	9
2.1.6	Incorrect Calculation of Reward Amount . . . . .	10
2.1.7	Incorrect Calculation in previewWithdraw() . . . . .	11
2.2	Additional Recommendation . . . . .	12
2.2.1	Incomplete Check when Setting newPermissions . . . . .	12
2.2.2	Duplicated Array Length Check . . . . .	12
2.2.3	Incomplete Check in toggleTemplateEndorsement() . . . . .	13
2.2.4	Duplicated Update in fundReward() . . . . .	14
2.2.5	Unnecessary Revert in claimRewards() . . . . .	15
2.2.6	Potential Centralization Problem . . . . .	16
2.3	Notes . . . . .	16
2.3.1	Unset FEE_RECIPIENT in AdapterBase . . . . .	16
2.3.2	Incompatible Tokens . . . . .	16
2.3.3	Bypassed Fee in Vault . . . . .	16

## Report Manifest

Item	Description
Client	Popcorn-Limited
Target	Popcorn

## Version History

Version	Date	Description
1.0	April 18, 2023	First Version

**About BlockSec** The **BlockSec Team** focuses on the security of the blockchain ecosystem, and collaborates with leading DeFi projects to secure their products. The team is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and released detailed analysis reports of high-impact security incidents. They can be reached at **Email**, **Twitter** and **Medium**.

# Chapter 1 Introduction

## 1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The repository that has been audited includes audit2 <sup>1</sup>.

The auditing process is iterative. Specifically, we will audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following. Our audit report is responsible for the only initial version (i.e., [Version 1](#)), as well as new codes (in the following versions) to fix issues in the audit report.

Project		Commit SHA
Popcorn	<a href="#">Version 1</a>	<a href="#">5bdd143c6d990049bb77dffe67cea43c13d4e19</a>
	<a href="#">Version 2</a>	<a href="#">d4428ef9587e197c11847e280cbc34ec67642674</a>

Note that, we did **NOT** audit all the modules in the repository. The modules covered by this audit report include **audit2/src** folder contract only. Specifically, the files covered in this audit include:

- interfaces
- utils/EIP.sol
- utils/MultiRewardEscrow.sol
- utils/MultiRewardStaking.sol
- utils/Owned.sol
- utils/OwnedUpgradeable.sol
- vault/adaptor
- vault/AdminProxy.sol
- vault/CloneFactory.sol
- vault/CloneRegistry.sol
- vault/DeploymentController.sol
- vault/FeeRecipientProxy.sol
- vault/PermissionRegistry.sol
- vault/TemplateRegistry.sol
- vault/Vault.sol
- vault/VaultController.sol
- vault/VaultRegistry.sol
- vault/VaultRouter.sol

---

<sup>1</sup><https://github.com/Popcorn-Limited/audit2>

## 1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

## 1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

### 1.3.1 Software Security

- \* Reentrancy
- \* DoS
- \* Access control
- \* Data handling and data flow
- \* Exception handling
- \* Untrusted external call and control flow
- \* Initialization consistency
- \* Events operation
- \* Error-prone randomness
- \* Improper use of the proxy system

### 1.3.2 DeFi Security

- \* Semantic consistency
- \* Functionality consistency
- \* Access control
- \* Business logic
- \* Token operation
- \* Emergency mechanism
- \* Oracle security
- \* Whitelist and blacklist
- \* Economic impact
- \* Batch transfer

### 1.3.3 NFT Security

- \* Duplicated item
- \* Verification of the token receiver
- \* Off-chain metadata security

### 1.3.4 Additional Recommendation

- \* Gas optimization
- \* Code quality and style



**Note** The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.

## 1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology <sup>2</sup> and Common Weakness Enumeration <sup>3</sup>. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.

---

<sup>2</sup>[https://owasp.org/www-community/OWASP\\_Risk\\_Rating\\_Methodology](https://owasp.org/www-community/OWASP_Risk_Rating_Methodology)

<sup>3</sup><https://cwe.mitre.org/>

**Table 1.1:** Vulnerability Severity Classification

Impact	High	High	Medium
	Low	Medium	Low
		High	Low
		Likelihood	

- **Fixed** The item has been confirmed and fixed by the client.

## Chapter 2 Findings

In total, we find **seven** potential issues. Besides, we have **six** recommendations and **three** notes as follows:

- High Risk: 2
- Medium Risk: 4
- Low Risk: 1
- Recommendations: 6
- Notes: 3

ID	Severity	Description	Category	Status
1	Medium	Unfair Charged Fee	DeFi Security	Fixed
2	Medium	Limitations of Staking Contract Deployment for Vault Contract	DeFi Security	Confirmed
3	High	Lost Rewards in fundReward()	DeFi Security	Fixed
4	Medium	Fee Shares Minted before highWaterMark Updated	DeFi Security	Fixed
5	Low	Limitation of Instant Reward Distribution in addRewardToken()	DeFi Security	Fixed
6	High	Incorrect Calculation of Reward Amount	DeFi Security	Fixed
7	Medium	Incorrect Calculation in previewWithdraw()	DeFi Security	Acknowledged
8	-	Incomplete Check when Setting newPermissions	Recommendation	Confirmed
9	-	Duplicated Array Length Check	Recommendation	Fixed
10	-	Incomplete Check in toggleTemplateEndorsement()	Recommendation	Fixed
11	-	Duplicated Update in fundReward()	Recommendation	Fixed
12	-	Unnecessary Revert in claimRewards()	Recommendation	Acknowledged
13	-	Potential Centralization Problem	Recommendation	Acknowledged
14	-	Unset FEE_RECIPIENT in AdapterBase	Note	Acknowledged
15	-	Incompatible Tokens	Note	Acknowledged
16	-	Bypassed Fee in Vault	Note	Acknowledged

The details are provided in the following sections.

### 2.1 DeFi Security

#### 2.1.1 Unfair Charged Fee

**Severity** Medium

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** In contracts [Vault](#) and [Adapter](#), the [admin](#) has the privilege to change the rate of fees dynamically. However, the management fee in [Vault](#), which is calculated based on the time passed from the last [feesUpdatedAt](#), is not charged before the fee rate updates. In this case, this period of time which



should be charged with the old fee rate is instead charged with the new fee rate directly, which is unfair for users.

```
381 function accruedManagementFee() public view returns (uint256) {
382     uint256 managementFee = fees.management;
383     return
384         managementFee > 0
385         ? managementFee.mulDiv(
386             totalAssets() * (block.timestamp - feesUpdatedAt),
387             SECONDS_PER_YEAR,
388             Math.Rounding.Down
389         ) / 1e18
390         : 0;
391 }
```

Listing 2.1: vault/Vault.sol

```
481 function changeFees() external {
482     if (proposedFeeTime == 0 || block.timestamp < proposedFeeTime + quitPeriod) revert
483         NotPassedQuitPeriod(quitPeriod);
484     emit ChangedFees(fees, proposedFees);
485
486     fees = proposedFees;
487     feesUpdatedAt = block.timestamp;
488
489     delete proposedFees;
490     delete proposedFeeTime;
491 }
```

Listing 2.2: vault/Vault.sol

**Impact** The management fee charged by the contract is unfair for a period of time after it's updated.

**Suggestion** Add the modifier `takeFees()` for function `changeFees()`.

## 2.1.2 Limitations of Staking Contract Deployment for Vault Contract

**Severity** Medium

**Status** Confirmed

**Introduced by** Version 1

**Description** In the current implementation, the `admin` can deploy a `Vault` contract via the function `deployVault()` with no staking contract. However, if the staking contract for the `Vault` is not deployed initially, the `admin` will not have another opportunity to set up a staking contract for this `Vault` in the future.

```
291 function deployStaking(IERC20 asset) external canCreate returns (address) {
292     _verifyToken(address(asset));
293     return _deployStaking(asset, deploymentController);
294 }
```

Listing 2.3: vault/VaultController.sol

```
91 function deployVault(  
92     VaultInitParams memory vaultData,  
93     DeploymentArgs memory adapterData,  
94     DeploymentArgs memory strategyData,  
95     bool deployStaking,  
96     bytes memory rewardsData,  
97     VaultMetadata memory metadata,  
98     uint256 initialDeposit  
99 ) external canCreate returns (address vault) {  
100     IDeploymentController _deploymentController = deploymentController;  
101  
102     _verifyToken(address(vaultData.asset));  
103     if (  
104         address(vaultData.adapter) != address(0) &&  
105         (adapterData.id > 0 || !cloneRegistry.cloneExists(address(vaultData.adapter)))  
106     ) revert InvalidConfig();  
107  
108     if (adapterData.id > 0)  
109         vaultData.adapter = IERC4626(_deployAdapter(vaultData.asset, adapterData, strategyData,  
110             _deploymentController));  
111  
112     vault = _deployVault(vaultData, _deploymentController);  
113  
114     address staking;  
115     if (deployStaking) staking = _deployStaking(IERC20(address(vault)), _deploymentController);  
116  
117     _registerCreatedVault(vault, staking, metadata);  
118  
119     if (rewardsData.length > 0) {  
120         if (!deployStaking) revert InvalidConfig();  
121         _handleVaultStakingRewards(vault, rewardsData);  
122     }  
123  
124     emit VaultDeployed(vault, staking, address(vaultData.adapter));  
125  
126     _handleInitialDeposit(initialDeposit, IERC20(vaultData.asset), IERC4626(vault));  
127 }
```

**Listing 2.4:** vault/VaultController.sol

**Impact** A `Vault` contract that was deployed without a staking contract initially can not have one in the future.

**Suggestion** Implement the logic to allow the staking contract to be added for `Vault` correspondingly.

### 2.1.3 Lost Rewards in fundReward()

**Severity** High

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** The function `fundReward()` in the contract `MultiRewardStaking` allows users to fund specific reward tokens into the contract. However, when the `rewardsPerSecond` and `supplyTokens` are both 0, the rewards transferred into the contract will not be recorded, and will not be distributed in the future.

```
325 function fundReward(IERC20 rewardToken, uint256 amount) external {
326     if (amount == 0) revert ZeroAmount();
327
328     // Cache RewardInfo
329     RewardInfo memory rewards = rewardInfos[rewardToken];
330
331     // Make sure that the reward exists
332     if (rewards.lastUpdatedTimestamp == 0) revert RewardTokenDoesntExist(rewardToken);
333
334     // Transfer additional rewardToken to fund rewards of this vault
335     rewardToken.safeTransferFrom(msg.sender, address(this), amount);
336
337     uint256 accrued = rewards.rewardsPerSecond == 0 ? amount : _accrueStatic(rewards);
338
339     // Update the index of rewardInfo before updating the rewardInfo
340     _accrueRewards(rewardToken, accrued);
341     uint32 rewardsEndTimestamp = rewards.rewardsEndTimestamp;
342     if (rewards.rewardsPerSecond > 0) {
343         rewardsEndTimestamp = _calcRewardsEnd(rewards.rewardsEndTimestamp, rewards.rewardsPerSecond,
344             amount);
345         rewardInfos[rewardToken].rewardsEndTimestamp = rewardsEndTimestamp;
346     }
347     rewardInfos[rewardToken].lastUpdatedTimestamp = block.timestamp.safeCastTo32();
348
349     emit RewardInfoUpdate(rewardToken, rewards.rewardsPerSecond, rewardsEndTimestamp);
350 }
```

Listing 2.5: `utils/MultiRewardStaking.sol`

```
403 function _accrueRewards(IERC20 _rewardToken, uint256 accrued) internal {
404     uint256 supplyTokens = totalSupply();
405     uint224 deltaIndex;
406     if (supplyTokens != 0)
407         deltaIndex = accrued.mulDiv(uint256(10**decimals()), supplyTokens, Math.Rounding.Down).
408             safeCastTo224();
409     rewardInfos[_rewardToken].index += deltaIndex;
410     rewardInfos[_rewardToken].lastUpdatedTimestamp = block.timestamp.safeCastTo32();
411 }
```

Listing 2.6: `utils/MultiRewardStaking.sol`

**Impact** Rewards funded by the users are lost.

**Suggestion** Add the check to ensure `supplyToken` is not equal to 0 in `fundReward()`.

### 2.1.4 Fee Shares Minted before highWaterMark Updated

**Severity** Medium

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** The modifier `takeFees()` is used to collect the performance fee. The calculation of performance fee is based on the difference between the value of each share and `highWaterMark`, and `highWaterMark` will be updated to the newly calculated value of each share for each fee charge.

However, the fee shares is minted before the update of `highWaterMark`, which results in the smaller value of `highWaterMark` than expected.

```
415 modifier takeFees() {
416     _;
417
418     uint256 fee = accruedPerformanceFee();
419     if (fee > 0) _mint(FEE_RECIPIENT, convertToShares(fee));
420
421     uint256 shareValue = convertToAssets(1e18);
422     if (shareValue > highWaterMark) highWaterMark = shareValue;
423 }
```

**Listing 2.7:** vault/adapter/abstracts/AdapterBase.sol

**Impact** Since `highWaterMark` is smaller, the performance fee will be larger, which is unfair to users.

**Suggestion** Update the `highWaterMark` to the newest `shareValue` before minting fee shares.

### 2.1.5 Limitation of Instant Reward Distribution in `addRewardToken()`

**Severity** Low

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** Function `addRewardToken()` allows the `admin` to add and configure new reward tokens for the contract `MultiRewardStaking`. According to the design, if the parameter `rewardsPerSecond` is set as 0, then it means that the rewards will be paid out instantly. However, the function doesn't allow the `admin` to distribute rewards instantly by setting the parameter `rewardsPerSecond` as 0 while the amount is larger than 0, which is against the design.

```
245 function addRewardToken(
246     IERC20 rewardToken,
247     uint160 rewardsPerSecond,
248     uint256 amount,
249     bool useEscrow,
250     uint192 escrowPercentage,
251     uint32 escrowDuration,
252     uint32 offset
253 ) external onlyOwner {
254     if (rewardTokens.length == 20) revert InvalidConfig();
255     if (asset() == address(rewardToken)) revert RewardTokenCantBeStakingToken();
256
257     RewardInfo memory rewards = rewardInfos[rewardToken];
258     if (rewards.lastUpdatedTimestamp > 0) revert RewardTokenAlreadyExist(rewardToken);
259 }
```

```
260     if (amount > 0) {
261         if (rewardsPerSecond == 0) revert ZeroRewardsSpeed();
262         rewardToken.safeTransferFrom(msg.sender, address(this), amount);
263     }
264
265     // Add the rewardToken to all existing rewardToken
266     rewardTokens.push(rewardToken);
267
268     if (useEscrow) {
269         if (escrowPercentage == 0 || escrowPercentage > 1e18) revert InvalidConfig();
270         escrowInfos[rewardToken] = EscrowInfo({
271             escrowPercentage: escrowPercentage,
272             escrowDuration: escrowDuration,
273             offset: offset
274         });
275         rewardToken.safeApprove(address(escrow), type(uint256).max);
276     }
277
278     uint64 ONE = (10**IERC20Metadata(address(rewardToken)).decimals()).safeCastTo64();
279     uint32 rewardsEndTimestamp = rewardsPerSecond == 0
280         ? block.timestamp.safeCastTo32()
281         : _calcRewardsEnd(0, rewardsPerSecond, amount);
282
283     rewardInfos[rewardToken] = RewardInfo({
284         ONE: ONE,
285         rewardsPerSecond: rewardsPerSecond,
286         rewardsEndTimestamp: rewardsEndTimestamp,
287         index: ONE,
288         lastUpdatedTimestamp: block.timestamp.safeCastTo32()
289     });
290
291     emit RewardInfoUpdate(rewardToken, rewardsPerSecond, rewardsEndTimestamp);
292 }
```

**Listing 2.8:** utils/MultiRewardStaking.sol

**Impact** The `admin` is prohibited from immediately distributing reward tokens in the function `addRewardToken()`.

**Suggestion** Implement the logic to allow the `admin` to distribute rewards in the function `addRewardToken()`.

### 2.1.6 Incorrect Calculation of Reward Amount

**Severity** High

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** The modifier `accrueRewards()` in the contract `MultiRewardStaking` is designed to update the status of all reward tokens and users.

However, the calculation of the reward amount for the user is incorrect. Since the decimal of `deltaIndex` is the same as `rewards.ONE`. Thus, the decimal of the `supplierDelta` (i.e., amount of rewards) is the same as the staking tokens rather than the reward token, which is incorrect.

```
414 function _accrueUser(address _user, IERC20 _rewardToken) internal {
415     RewardInfo memory rewards = rewardInfos[_rewardToken];
416
417     uint256 oldIndex = userIndex[_user][_rewardToken];
418
419     // If user hasn't yet accrued rewards, grant them interest from the strategy beginning if they
         have a balance
420     // Zero balances will have no effect other than syncing to global index
421     if (oldIndex == 0) {
422         oldIndex = rewards.ONE;
423     }
424
425     uint256 deltaIndex = rewards.index - oldIndex;
426
427     // Accumulate rewards by multiplying user tokens by rewardsPerToken index and adding on
         unclaimed
428     uint256 supplierDelta = balanceOf(_user).mulDiv(deltaIndex, uint256(rewards.ONE), Math.
         Rounding.Down);
429
430     userIndex[_user][_rewardToken] = rewards.index;
431
432     accruedRewards[_user][_rewardToken] += supplierDelta;
433 }
```

**Listing 2.9:** utils/MultiRewardStaking.sol

**Impact** The amount of rewards distributed to users is wrong.

**Suggestion** Replace the `rewards.ONE` with the decimals of the staking token.

### 2.1.7 Incorrect Calculation in `previewWithdraw()`

**Severity** Medium

**Status** Acknowledged

**Introduced by** Version 2

**Description** The function `previewWithdraw()` allows users to provide the amount of assets they wish to withdraw, and then calculates and returns the number of shares that need to be burned by the protocol. When calculating the total amount of assets (line 116), it should use rounding up instead of rounding down.

```
106 function previewWithdraw(uint256 assets) public view override returns (uint256) {
107     IBeefyStrat strat = IBeefyStrat(beefyVault.strategy());
108
109     uint256 beefyFee;
110     try strat.withdrawalFee() returns (uint256 _beefyFee) {
111         beefyFee = _beefyFee;
112     } catch {
113         beefyFee = strat.withdrawFee();
114     }
115
116     if (beefyFee > 0) assets = assets.mulDiv(BPS_DENOMINATOR, BPS_DENOMINATOR - beefyFee, Math.
         Rounding.Down);
```

```
117
118     return _convertToShares/assets, Math.Rounding.Up);
119 }
```

**Listing 2.10:** vault/adaptor/beefy/BeefyAdapter.sol

**Impact** The returned value of asset may be incorrect.

**Suggestion** Use `Math.Rounding.Up` in the calculation.

## 2.2 Additional Recommendation

### 2.2.1 Incomplete Check when Setting newPermissions

**Status** Confirmed

**Introduced by** Version 1

**Description** In the current implementation, if `newPermissions[i].endorsed` and `newPermissions[i].rejected` are both `true`, the transaction will revert. However, the contract does not check the case where both elements are `false`. This scenario should also not be allowed.

```
38 function setPermissions(address[] calldata targets, Permission[] calldata newPermissions)
    external onlyOwner {
39     uint256 len = targets.length;
40     if (len != newPermissions.length) revert Mismatch();
41
42     for (uint256 i = 0; i < len; i++) {
43         if (newPermissions[i].endorsed && newPermissions[i].rejected) revert Mismatch();
44
45         emit PermissionSet(targets[i], newPermissions[i].endorsed, newPermissions[i].rejected);
46
47         permissions[targets[i]] = newPermissions[i];
48     }
49 }
```

**Listing 2.11:** vault/PermissionRegistry.sol

**Suggestion I** Add the check to make sure the permission either be `endorsed` or `rejected`.

**Feedback from the Project** False and false is a valid state. Its also the default state for each address. In this case, `_verifyToken` will be bypassed by default, which is intended.

### 2.2.2 Duplicated Array Length Check

**Status** Fixed in Version 2

**Introduced by** Version 1

**Description** Function `setFees()` in the contract `MultiRewardEscrow` will be invoked by function `setEscrowTokenFees()`. However, two arrays (i.e., `tokens[]` and `tokenFees[]`) are both checked in these two functions to make sure they have the same length, which is duplicated.

```
208 function setFees(IERC20[] memory tokens, uint256[] memory tokenFees) external onlyOwner {
209     if (tokens.length != tokenFees.length) revert ArraysNotMatching(tokens.length, tokenFees.
        length);
210
211     for (uint256 i = 0; i < tokens.length; i++) {
212         if (tokenFees[i] >= 1e17) revert DontGetGreedy(tokenFees[i]);
213
214         fees[tokens[i]].feePerc = tokenFees[i];
215         emit FeeSet(tokens[i], tokenFees[i]);
216     }
217 }
```

**Listing 2.12:** utils/MultiRewardEscrow.sol

```
622 function setEscrowTokenFees(IERC20[] calldata tokens, uint256[] calldata fees) external
    onlyOwner {
623     _verifyEqualArrayLength(tokens.length, fees.length);
624     (bool success, bytes memory returnData) = adminProxy.execute(
625         address(escrow),
626         abi.encodeWithSelector(IMultiRewardEscrow.setFees.selector, tokens, fees)
627     );
628     if (!success) revert UnderlyingError(returnData);
629 }
```

**Listing 2.13:** vault/VaultController.sol

**Suggestion I** It's suggested to remove the length check in the function `setEscrowTokenFees()`.

## 2.2.3 Incomplete Check in `toggleTemplateEndorsement()`

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** The function `toggleTemplateEndorsement()` allows the admin to toggle the endorsement of a `template`. It will validate whether the `templateId` exists before the operation. However, the `templateCategory` is not checked, which may make the endorsement of the non-existent `template` become true.

```
102 function toggleTemplateEndorsement(bytes32 templateCategory, bytes32 templateId) external
    onlyOwner {
103     if (!templateExists[templateId]) revert KeyNotFound(templateId);
104
105     bool oldEndorsement = templates[templateCategory][templateId].endorsed;
106     templates[templateCategory][templateId].endorsed = !oldEndorsement;
107
108     emit TemplateEndorsementToggled(templateCategory, templateId, oldEndorsement, !oldEndorsement)
        ;
109 }
```

**Listing 2.14:** vault/TemplateRegistry.sol

**Suggestion I** Add the check to make sure the specified `templateCategory` exists.



## 2.2.4 Duplicated Update in fundReward()

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** In the function `fundReward()`, the `lastUpdatedTimestamp` of `rewardToken` will be updated by invoking the internal function `_accrueRewards()` (line 410). However, the function `fundReward()` updates it again in line 347, which is duplicated.

```
325 function fundReward(IERC20 rewardToken, uint256 amount) external {
326     if (amount == 0) revert ZeroAmount();
327
328     // Cache RewardInfo
329     RewardInfo memory rewards = rewardInfos[rewardToken];
330
331     // Make sure that the reward exists
332     if (rewards.lastUpdatedTimestamp == 0) revert RewardTokenDoesntExist(rewardToken);
333
334     // Transfer additional rewardToken to fund rewards of this vault
335     rewardToken.safeTransferFrom(msg.sender, address(this), amount);
336
337     uint256 accrued = rewards.rewardsPerSecond == 0 ? amount : _accrueStatic(rewards);
338
339     // Update the index of rewardInfo before updating the rewardInfo
340     _accrueRewards(rewardToken, accrued);
341     uint32 rewardsEndTimestamp = rewards.rewardsEndTimestamp;
342     if (rewards.rewardsPerSecond > 0) {
343         rewardsEndTimestamp = _calcRewardsEnd(rewards.rewardsEndTimestamp, rewards.rewardsPerSecond,
            amount);
344         rewardInfos[rewardToken].rewardsEndTimestamp = rewardsEndTimestamp;
345     }
346
347     rewardInfos[rewardToken].lastUpdatedTimestamp = block.timestamp.safeCastTo32();
348
349     emit RewardInfoUpdate(rewardToken, rewards.rewardsPerSecond, rewardsEndTimestamp);
350 }
```

**Listing 2.15:** `utils/MultiRewardStaking.sol`

```
403 function _accrueRewards(IERC20 _rewardToken, uint256 accrued) internal {
404     uint256 supplyTokens = totalSupply();
405     uint224 deltaIndex;
406     if (supplyTokens != 0)
407         deltaIndex = accrued.mulDiv(uint256(10**decimals()), supplyTokens, Math.Rounding.Down).
            safeCastTo224();
408
409     rewardInfos[_rewardToken].index += deltaIndex;
410     rewardInfos[_rewardToken].lastUpdatedTimestamp = block.timestamp.safeCastTo32();
411 }
```

**Listing 2.16:** `utils/MultiRewardStaking.sol`

**Suggestion I** Remove the update in the function `fundReward()`.

## 2.2.5 Unnecessary Revert in claimRewards()

**Status** Acknowledged

**Introduced by** Version 1

**Description** Function `claimRewards()` in the contract `MultiRewardStaking` allows the user to claim their rewards by inputting an array of `_rewardTokens`. The function will iterate the array to collect and calculate the reward amount for each reward token. However, if one of the reward tokens' claimable amount is zero, the whole transaction will revert, which is unnecessary.

The same problem also exists in the function `claimRewards()` in the contract `MultiRewardEscrow`.

```
170 function claimRewards(address user, IERC20[] memory _rewardTokens) external accrueRewards(msg.  
    sender, user) {  
171     for (uint8 i; i < _rewardTokens.length; i++) {  
172         uint256 rewardAmount = accruedRewards[user][_rewardTokens[i]];  
173  
174         if (rewardAmount == 0) revert ZeroRewards(_rewardTokens[i]);  
175  
176         accruedRewards[user][_rewardTokens[i]] = 0;  
177  
178         EscrowInfo memory escrowInfo = escrowInfos[_rewardTokens[i]];  
179  
180         if (escrowInfo.escrowPercentage > 0) {  
181             _lockToken(user, _rewardTokens[i], rewardAmount, escrowInfo);  
182             emit RewardsClaimed(user, _rewardTokens[i], rewardAmount, true);  
183         } else {  
184             _rewardTokens[i].transfer(user, rewardAmount);  
185             emit RewardsClaimed(user, _rewardTokens[i], rewardAmount, false);  
186         }  
187     }  
188 }
```

**Listing 2.17:** utils/MultiRewardStaking.sol

```
155 function claimRewards(bytes32[] memory escrowIds) external {  
156     for (uint256 i = 0; i < escrowIds.length; i++) {  
157         bytes32 escrowId = escrowIds[i];  
158         Escrow memory escrow = escrows[escrowId];  
159  
160         uint256 claimable = _getClaimableAmount(escrow);  
161         if (claimable == 0) revert NotClaimable(escrowId);  
162  
163         escrows[escrowId].balance -= claimable;  
164         escrows[escrowId].lastUpdateTime = block.timestamp.safeCastTo32();  
165  
166         escrow.token.safeTransfer(escrow.account, claimable);  
167         emit RewardsClaimed(escrow.token, escrow.account, claimable);  
168     }  
169 }
```

**Listing 2.18:** utils/MultiRewardEscrow.sol

**Suggestion I** Continue the iteration when one of the claimable amounts is 0.

## 2.2.6 Potential Centralization Problem

**Status** Acknowledged

**Introduced by** [Version 1](#)

**Description** This project has potential centralization problems. The [admin](#) of the [VaultController](#) has the privilege to set [DeploymentController](#), change [Adapter](#) of [Vaults](#), set permissions for assets, pause and unpause [Vaults](#) and [Adapters](#), and configure a number system parameters (e.g., change [fees](#) and [harvestCooldown](#)).

**Suggestion I** It is recommended to introduce a decentralization design in the contract, such as a multi-signature or a public [DAO](#).

## 2.3 Notes

### 2.3.1 Unset FEE\_RECIPIENT in AdapterBase

**Status** Acknowledged

**Introduced by** [version 1](#)

**Description** In the contract [AdapterBase](#), the [FEE\\_RECIPIENT](#) is set as [address\(0x4444\)](#), which is a dead address. According to the documentation, the team will deploy a deterministic address as [FEE\\_RECIPIENT](#) as they didn't deploy the proxy yet on their target chains. Once the proxy exists they will switch out the placeholder address.

### 2.3.2 Incompatible Tokens

**Status** Acknowledged

**Introduced by** [version 1](#)

**Description** Elastic supply tokens are not compatible with the protocol. They could dynamically adjust their price, supply, user's balance, etc. Such as inflation tokens, deflation tokens, rebasing tokens, and so forth. The inconsistency could result in security impacts if some critical operations are based on the recorded amount of transferred tokens.

### 2.3.3 Bypassed Fee in Vault

**Status** Acknowledged

**Introduced by** [version 1](#)

**Description** In the current implementation, the [Vault](#) contract charges users four different types of fees: deposit fees, withdrawal fees, management fees, and performance fees. Among them, deposit fee, withdrawal fee and management fee can be bypassed. Specifically, users can interact directly with contract [Adapter](#) to use the same services without incurring those fees.

Besides, the performance fees are both charged in [Vault](#) and [Adapter](#), due to the implementation of two separate fee structures.