Audit Report

# Rewardable

November 2024

Repository https://github.com/artiffine-vojtech/rewardable-contracts

Commit db6c67ab7b3aa0943de404d8c7e307bab05ef227

Audited by © cyberscope

# Table of Contents

# Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

| Severity | Likelihood / Impact of Exploitation |
| --- | --- |
| ● Critical | Highly Likely / High Impact |
| ● Medium | Less Likely / High Impact or Highly Likely/ Lower Impact |
| ● Minor / Informative | Unlikely / Low to no Impact |

# Review

| | |
|---|---|
| **Repository** | https://github.com/artiffine-vojtech/rewardable-contracts |
| **Commit** | db6c67ab7b3aa0943de404d8c7e307bab05ef227 |

## Audit Updates

| | |
|---|---|
| **Initial Audit** | 16 Oct 2024<br>https://github.com/cyberscope-io/audits/blob/main/2-reward/v1/audit.pdf |
| **Corrected Phase 2** | 14 Nov 2024 |

| | |
|---|---|
| **Test Deploys** | https://sepolia.etherscan.io/address/0xFF9D9141EbF1Fbc593C2c61d3f06738Bd3D4Ae2C<br>https://sepolia.etherscan.io/address/0xD8E9a0050A2731d1003308CECD535431aaC2cB87<br>https://sepolia.etherscan.io/address/0x660FA83C351D845503da0C1c821dD968f58a2766<br>https://sepolia.etherscan.io/address/0xb7cf368F8937Ab3Dab37909214ddaD310A953c0A<br>https://sepolia.etherscan.io/address/0x6ca7E9CBB95233B5e588881E1764c61F77aF822b |

## Source Files

| **Filename** | SHA256 |
|---|---|
| **UniV3IncentivesController.sol** | 42f0f2014aa4c2c3f0e11f079d1309e05825c6abda942ffb9156ce0b3d4dc007 |

| | |
|---|---|
| **TokenProxy.sol** | ff956778f628fefaf3eeb28f27313a6ac2a892f6d2031104c4ac7e1f70740dfe |
| **TokenEmissionsController.sol** | f79c172ab4f612a6d030254c11fb503e973c7fce024bc2d742ff7b36afd6279a |
| **RewardableLZ.sol** | cf5150b549a897d13b10c4f4b52fdb9f37543f69b2bf4372db9aec0d684e246d |
| **RewardDistributorV1LZ.sol** | c791bacc229bc587d434b07a5a4340e208d7e57d44a74212d48e3a9429ad4ebd |
| **utils/Adminable.sol** | dd9b574e06f0b2d79b3790f29af71725f283fd2963e0a451408830dc91bb021a |
| **interfaces/ITokenProxy.sol** | 1a5e4b837c67f06cdb530795f14cce57a69461cb09a5593e8e0d0a03ab304d49 |
| **interfaces/ITokenIncentivesController.sol** | bf963104b7eb2732d83c55552b60b843eb5be28b5605b3f727784900292b05c4 |
| **interfaces/ITokenEmissionsController.sol** | 6e1b89b897c4d450981a9267e79c0fdb0da6dd5dc036673606e5843205bff815 |
| **interfaces/ITokenControllerCommons.sol** | fbceb0c013dd04294f19ce9d775d4fa1298cb2ba052f4dc65a3964d07948430d |

# Overview

Rewardable is a decentralised staking platform designed to facilitate the seamless distribution of rewards for staked tokens and liquidity positions. The protocol's primary functionalities encompass four core smart contracts: the Rewardable token, the TokenEmissionsController, the UniV3IncentivesController and the RewardDistributorV1LZ.

**Rewardable token**

The rewardable token is a LayerZero token that supports minting and burning across multiple networks without the need of a bridge.

**TokenEmissionsController**

The TokenEmissionsController is responsible for the staking and distribution of rewards for the tokens in the system. The contract supports distributing rewards in multiple coins. Staked funds remain locked for a minimum of 2 months, with a maximum lock-up period of up to 4 months. Tokens staked for a longer lock-up period benefit from higher rewards. After the lock-up period, users can withdraw their tokens, and the accrued rewards are claimed during the withdrawal process. Additionally, users can claim accumulated rewards without affecting their staked positions.

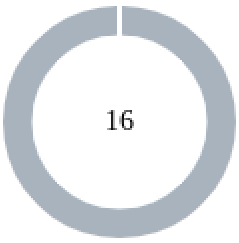
**UniV3IncentivesController**

The UniV3IncentivesController is responsible for the handling of staked liquidity positions of a Uniswap V3 pair and the distribution of staking rewards. The contract distributes rewards proportionally to the liquidity of the position. Positions of higher liquidity earn more rewards. Staked positions can be withdrawn from the contract immediately after they are deposited as there is no locking period. Rewards can also be distributed in multiple coins.

**RewardDistributorV1LZ**

The RewardDistributorV1LZ is responsible for distributing rewards to users. Users can utilize these rewards to create new tasks or top up existing ones. Additionally, requests for reward withdrawals can be processed across multiple networks using the LayerZero interface.

# Findings Breakdown




| | |
|---|---|
| Critical | 0 |
| Medium | 0 |
| Minor / Informative | 16 |

| Severity | Unresolved | Acknowledged | Resolved | Other |
|---|---|---|---|---|
| Critical | 0 | 0 | 0 | 0 |
| Medium | 0 | 0 | 0 | 0 |
| Minor / Informative | 0 | 16 | 0 | 0 |

# Diagnostics

● Critical ● Medium ● Minor / Informative

| Severity | Code | Description | Status |
|---|---|---|---|
| ● | APW | Admin Privileged Withdrawals | Acknowledged |
| ● | CCR | Contract Centralization Risk | Acknowledged |
| ● | IAI | Inadequate Admin Initialization | Acknowledged |
| ● | ISV | Inadequate Signature Verification | Acknowledged |
| ● | MTLV | Missing Time Lock Validation | Acknowledged |
| ● | MU | Modifiers Usage | Acknowledged |
| ● | PLAM | Potential Liquidity Amount Manipulation | Acknowledged |
| ● | PLTM | Potential Lock Time Manipulation | Acknowledged |
| ● | PTAI | Potential Transfer Amount Inconsistency | Acknowledged |
| ● | RFD | Redundant Function Declaration | Acknowledged |
| ● | RSML | Redundant SafeMath Library | Acknowledged |
| ● | SVMC | Signature Validation Missing ChainID | Acknowledged |
| ● | TSI | Tokens Sufficiency Insurance | Acknowledged |

| | | | |
|---|---|---|---|
| ● | L04 | Conformance to Solidity Naming Conventions | Acknowledged |
| ● | L16 | Validate Variable Setters | Acknowledged |
| ● | L19 | Stable Compiler Version | Acknowledged |

## APW - Admin Privileged Withdrawals

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | TokenEmissionsController.sol#L117 |
| **Status** | Acknowledged |

### Description

The `withdraw` function in the staking contract allows the `withdrawingAdmin` to withdraw staked tokens on behalf of any staker after the expiration of the locking period. This functionality can be exploited if the `withdrawingAdmin` account is compromised or misused.

```
function withdraw(uint _amount, address _onBehalfOf) external {
    require(msg.sender == _onBehalfOf || msg.sender ==
withdrawingAdmin, 'Not withdrawing admin');
    require(userLockTime[_onBehalfOf] <= block.timestamp,
'Locked');
    Balances storage bal = balances[_onBehalfOf];
    require(_amount <= bal.staked, 'Amount greater than staked');
    _updateReward(_onBehalfOf, rewardTokens);
    if (msg.sender == _onBehalfOf) {
        _getReward(rewardTokens);
    }
    uint scaled = _amount.mul(bal.lockBoost).div(10);
    if (bal.boosted) {
        uint multiplier = _getMultiplier(bal.nftId);
        scaled = scaled.mul(multiplier).div(10);
    }
    if (_amount == bal.staked) {
        scaled = bal.scaled;
        bal.lockBoost = 0;
    }
    bal.staked = bal.staked.sub(_amount);
    bal.lockScaled = bal.staked.mul(bal.lockBoost).div(10);
    bal.scaled = bal.scaled.sub(scaled);
    totalScaled = totalScaled.sub(scaled);
    stakingToken.safeTransfer(msg.sender, _amount);
    emit Withdrawn(_onBehalfOf, _amount, scaled);
}
```

## Recommendation

To mitigate this issue, the privilege of the withdrawingAdmin should be restricted to prevent unauthorized withdrawals. Consider implementing a multisignature (multisig) mechanism where multiple trusted parties must approve an action. Additionally, role-based access control could be used to segregate duties and limit the scope of administrative actions.

## Team Update

The team has acknowledged that this is not a security issue and states:

*Multisig is used.*

## CCR - Contract Centralization Risk

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | UniV3IncentivesController.sol#L214,224<br>RewardDistributorV1LZ.sol#L339,349,359,369,379,389,399,408,421,446<br>TokenEmissionsController.sol#L71,94,117,223,233 |
| **Status** | Acknowledged |

### Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
function deposit(uint _amount, address _onBehalfOf, LockTime _lock)
external {}
function withdraw(uint _amount, address _onBehalfOf) external {}
function addReward(address _rewardToken) external onlyAdmin {}
function notifyReward(address[] calldata _rewardTokens, uint[]
calldata _amounts, uint _rewardsDuration) external onlyAdmin {}
function setTokenAdmin(address _tokenAdmin) external onlyOwner {}
function setFeeReceiver(address _feeReceiver) external onlyOwner {}
function setBurnFee(uint _burnFee) external onlyOwner {}
function setPlatformFee(uint _platformFee) external onlyOwner {}
function setMaxDailyWithdrawal(uint _maxDailyWithdrawal) external
onlyOwner {}
function setMinWithdrawalAmount(uint _minWithdrawalAmount) external
onlyOwner {}
function recoverFees(uint _recoverAmount, address _recipient)
external onlyOwner {}
function burnFees(uint _burnAmount) external onlyOwner {}
function _processFees(uint _amount) internal returns (uint
amountAfterFees) {}
function _authorizeUpgrade(address) internal override onlyOwner {}
function startEmissions(EmissionPoint[] memory _emissions) external

onlyOwner {}

function withdrawRewards(
    address _identity,
    uint _amount,
    bytes calldata _data,
    LZSendParam calldata _lzSendParam
) external payable {}
```

## Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

## Team Update

The team has acknowledged that this is not a security issue and states:

*Multisig is used.*

## IAI - Inadequate Admin Initialization

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | TokenEmissionsController.sol#L65,94,117<br>TokenProxy.sol#L26,37 |
| **Status** | Acknowledged |

### Description

The TokenEmissionsController contract allows a withdrawingAdmin to execute deposit and withdrawal requests on behalf of a user. Such requests can also be initiated by the TokenProxy contract through its respective functions. If however the TokenProxy contract is not designated as the withdrawingAdmin of the TokenEmissionsController, all such requests will fail.

```
function deposit(uint _amount, address _onBehalfOf, LockTime _lock)
external {
require(msg.sender == _onBehalfOf || msg.sender ==
withdrawingAdmin, 'Not withdrawing admin');
...
}

function withdraw(uint _amount, address _onBehalfOf) external {
require(msg.sender == _onBehalfOf || msg.sender ==
withdrawingAdmin, 'Not withdrawing admin');
...
}
```

```
function deposit(uint _amount, ITokenEmissionsController.LockTime
_lock) external {
proxiedToken.safeTransferFrom(msg.sender, address(this), _amount);
_mint(address(this), _amount);
_approve(address(this), address(controller), _amount);
controller.deposit(_amount, msg.sender, _lock);
}

function withdraw(uint _amount) external {
controller.withdraw(_amount, msg.sender);
_burn(address(this), _amount);
proxiedToken.safeTransfer(msg.sender, _amount);
}
```

## Recommendation

The team needs to ensure the TokenProxy contract is set as the withdrawingAdmin of the TokenEmissionsController to guarantee proper execution.

## Team Update

The team has acknowledged that this is not a security issue and states:

*The checks are performed in the deployment scripts*

# ISV - Inadequate Signature Verification

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | RewardDistributorV1LZ.sol#L241 |
| **Status** | Acknowledged |

## Description

The contract implements a signature verification process within the `withdrawRewards` function, which is designed to authenticate transactions based on digital signatures. The contract's logic validates the signature by comparing the extracted message with an expected message. However, the latter is formed from the data of the signature and the arguments passed by the user when the function is called. Since both elements of the comparison are user dependent variables, this approach introduces a security risk, as it inherently trusts the input of the user without independent verification.

```
(uint totalAmount, bytes32 message, bytes memory signature) =
abi.decode(_data, (uint, bytes32, bytes));
bytes32 expectedMessage = keccak256(abi.encodePacked('\x19Ethereum
Signed Message:\n52', _identity, totalAmount));
require(message == expectedMessage, 'Invalid proof of rewards');
```

## Recommendation

To enhance the security of the signature verification process, the expected message should be separated from user input. It is recommended to use additional validation mechanisms to ensure the message's authenticity before performing signature comparison. In this specific implementation, validating that the signer of the message is the tokenAdmin before message validation could prevent counterfeit signatures from bypassing a conditional statement.

## Team Update

The team has acknowledged that this is not a security issue and states:

*We actually check that tokenAdmin signed the message.*

## MTLV - Missing Time Lock Validation

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | UniV3IncentivesController.sol#L109 |
| **Status** | Acknowledged |

### Description

The contract implements a staking mechanism that distributes rewards to staked positions based on their liquidity. Currently, the implementation allows new positions to be accepted without a lock-up period, enabling them to be withdrawn immediately after submission, potentially within the same block. This enables attempts to manipulate the internal state of the contract without depositing funds.

```
function deposit(uint[] calldata _nftIds) external {
    updateReward(msg.sender, rewardTokens);
    uint length = _nftIds.length;
    for (uint i = 0; i < length; i++) {
    uint nftId = _nftIds[i];
    (
    ,
    ,
    address _token0,
    address _token1,
    ,
    int24 _tickLower,
    int24 _tickUpper,
    uint128 liquidity,
    ,
    ,
    ,
    ) = INonfungiblePositionManager(address(nft)).positions(nftId);
    require(posConfig.tickLower <= _tickLower, 'Invalid lower
tick');
    require(posConfig.tickUpper >= _tickUpper, 'Invalid upper
tick');
    require(posConfig.token0 == _token0, 'Invalid token0');
    require(posConfig.token1 == _token1, 'Invalid token1');
    require(liquidity > 0, 'Invalid liquidity');
    positions[msg.sender].add(nftId);
    nftLiquidity[nftId] = liquidity;
    userLiquidity[msg.sender] += liquidity;
    totalLiquidity += liquidity;
    nft.safeTransferFrom(msg.sender, address(this), nftId);
    emit Deposited(msg.sender, nftId, liquidity);
    }
}
```

## Recommendation

The team is advised to implement a locking period for the deposited positions. This will prevent threads that could target price and parameter manipulation.

## MU - Modifiers Usage

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | TokenEmissionsController.sol#L95,118 |
| **Status** | Acknowledged |

### Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(msg.sender == _onBehalfOf || msg.sender ==
withdrawingAdmin, 'Not withdrawing admin');
```

### Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

## PLAM - Potential Liquidity Amount Manipulation

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | UniV3IncentivesController.sol#L109 |
| **Status** | Acknowledged |

### Description

The contract allows users to deposit liquidity positions for a specific trading pair. To determine the liquidity of a position, the contract calls the `positions` function from the `NonfungiblePositionManager`. The value returned by this function is used to calculate the user's balance for the reward system. However, this method has some limitations due to the non-linear nature of liquidity calculations, which depend on the price range of the position.

Specifically, depositing the same amount of tokens within a narrower price range results in a higher liquidity value compared to a broader range with the same amount. This is in part due to the calculated liquidity being inversely proportional to the expression `sqrt(Pu)-sqrt(Pl)`, where `Pu` is the upper price limit and `Pl` is the lower price limit of the position.

Therefore, interpreting liquidity as an absolute number allows smaller positions concentrated in a narrower price range to appear more significant than larger positions spread over a wider range. This effectively incentivizes users to divide their funds into multiple smaller positions within narrow ranges and deposit them into the contract. By following this strategy, a user may accumulate enough liquidity to claim the majority of the rewards. Consequently, it becomes possible to withdraw a disproportionately large amount of rewards from relatively smaller positions.

To address this issue, the contract requires liquidity to be provided within a specific tick range. While this approach mitigates liquidity manipulation, it complicates the process for users to deposit their liquidity into the contract, as it necessitates specific tick settings on their part.

```
function deposit(uint[] calldata _nftIds) external {
    updateReward(msg.sender, rewardTokens);
    uint length = _nftIds.length;
    for (uint i = 0; i < length; i++) {
    uint nftId = _nftIds[i];
    (
    ,
    ,
    address _token0,
    address _token1,
    ,
    int24 _tickLower,
    int24 _tickUpper,
    uint128 liquidity,
    ,
    ,
    ,
    ) = INonfungiblePositionManager(address(nft)).positions(nftId);
    require(posConfig.tickLower <= _tickLower, 'Invalid lower tick');
    require(posConfig.tickUpper >= _tickUpper, 'Invalid upper tick');
    require(posConfig.token0 == _token0, 'Invalid token0');
    require(posConfig.token1 == _token1, 'Invalid token1');
    require(liquidity > 0, 'Invalid liquidity');
    positions[msg.sender].add(nftId);
    nftLiquidity[nftId] = liquidity;
    userLiquidity[msg.sender] += liquidity;
    totalLiquidity += liquidity;
    nft.safeTransferFrom(msg.sender, address(this), nftId);
    emit Deposited(msg.sender, nftId, liquidity);
    }
}
```

## Recommendation

The team is advised to revise the implementation of the reward mechanism in the UniV3IncentivesController contract. Using a position's liquidity as a record of the staked balance is prone to errors and may lead to several potential threats, including price manipulation. Implementing restrictions on the specific ticks where users may provide liquidity is counterintuitive and may hinder user access to the contract. A solution that maintains the term `sqrt(Pu) - sqrt(Pl)` constant is preferable, as it allows greater flexibility for users. Alternatively, the team could develop an interface to facilitate user deployment of liquidity at specific intervals. For more information on the relationship between tick size and price the team is advised to visit the Uniswap documentation.
https://app.uniswap.org/whitepaper-v3.pdf

## Team Update

The team has acknowledged that this is not a security issue and states:

*The staking will be allowed only for the full range LPTs, thus mitigating this issue.*

## PLTM - Potential Lock Time Manipulation

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | TokenEmissionsController.sol#L94 |
| **Status** | Acknowledged |

### Description

The contract implements a timelock mechanism for new deposits, where the `userLockTime` mapping stores a lock period for each user. Users can withdraw their staked funds and earned rewards only after the specified lock period. The rewards are proportional to the duration the funds are locked, with longer lock periods yielding higher rewards. The contract offers three specific locking periods: 60, 90, and 120 days.

A potential issue arises when a user deposits a large amount of tokens, selecting the longest lock period to maximize rewards. During this period, the user can make a smaller deposit, resetting the lock time to the shortest period of 60 days from the second deposit. This allows the user to claim high rewards calculated for the longest lock period, but withdraw them sooner than that. For example, a user initially commits a large deposit with a 4-month lock period, then later resets the lock period to 2 months, enabling them to withdraw rewards intended for a 4-month commitment after just 2 months. This approach allows users to optimize their deposits and reset timestamps to maximize earnings against the intended functionality of the contract.

```
function deposit(uint _amount, address _onBehalfOf, LockTime _lock)
external {
    ...
    userLockTime[_onBehalfOf] = block.timestamp.add(60
days).add(uint256(_lock).mul(30 days));
    stakingToken.safeTransferFrom(msg.sender, address(this),
_amount);
    emit Deposited(_onBehalfOf, _amount, scaled);
}
```

## Recommendation

The team is advised to revise the implementation of the deposit and withdraw functions. Specifically, it is suggested to prevent users from resetting the `userLockTime` to a lower time frame than the currently defined.

# PTAI - Potential Transfer Amount Inconsistency

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | TokenProxy.sol#L26,37 |
| **Status** | Acknowledged |

## Description

The contract implements the `deposit` and `withdraw` functions that allow external users to interact with the `TokenEmissionsController` through the `TokenProxy` contract. As part of these functions, ERC20 tokens are minted and burned in the same amount as the tokens requested. However, the contract fails to verify whether fees are applied during the transfer. In that case, the `TokenProxy` contract will mistakenly assume an amount different than the one received.

The following example depicts the diversion between the expected and actual amount.

| Tax | Amount | Expected | Actual |
|---|---|---|---|
| No Tax | 100 | 100 | 100 |
| 10% Tax | 100 | 100 | 90 |

```
function deposit(uint _amount, ITokenEmissionsController.LockTime
_lock) external {
    proxiedToken.safeTransferFrom(msg.sender, address(this),
_amount);
    _mint(address(this), _amount);
    _approve(address(this), address(controller), _amount);
    controller.deposit(_amount, msg.sender, _lock);
}

function withdraw(uint _amount) external {
    controller.withdraw(_amount, msg.sender);
    _burn(address(this), _amount);
    proxiedToken.safeTransfer(msg.sender, _amount);
}
```

## Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

`Actual Transferred Amount = Balance After Transfer - Balance Before Transfer`

## Team Update

The team has acknowledged that this is not a security issue and states:

*There is no tax on token transfers*

## RFD - Redundant Function Declaration

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | TokenEmissionsController.sol#L146,165,352 |
| **Status** | Acknowledged |

### Description

The contract contains functions with no scope or functions that are never executed. Implementing such functions hinders the overall complexity and readability. In particular, the `stakeNFT` and `unstakeNFT` functions have commented-out code and the execution of the `_getMultiplier` function is never reached.

```
function stakeNFT(uint _tokenId) external {}
function unstakeNFT() external {}
function _getMultiplier(uint256 _tokenId) internal view returns
(uint256) {
uint256 level = boosterNFT.getLevelOfTokenById(_tokenId);

        if (level == 0) {
            // Diamond
            return 15; // 1.5 * SCALING_FACTOR
        } else if (level == 1) {
            // Platinum
            return 14; // 1.4 * SCALING_FACTOR
        } else if (level == 2) {
            // Gold
            return 13; // 1.3 * SCALING_FACTOR
        } else if (level == 3) {
            // Silver
            return 12; // 1.2 * SCALING_FACTOR
        } else if (level == 4) {
            // Bronze
            return 11; // 1.1 * SCALING_FACTOR
        } else {
            revert('Invalid token level');
        }
}
```

## Recommendation

Functions with no scope or unreachable execution should be removed from the code base to improve the efficiency and readability of the contract.

## RSML - Redundant SafeMath Library

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | TokenEmissionsController.sol |
| **Status** | Acknowledged |

### Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily in cases where the explanatory error message is not used.

```
library SafeMath {...}
```

### Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/stable/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

## SVMC - Signature Validation Missing ChainID

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | RewardDistributorV1LZ.sol#L241 |
| **Status** | Acknowledged |

### Description

The contract's `withdrawRewards` function is designed to validate off-chain signatures for operations involving token transfers between addresses. However, the function does not include the `chainId` as part of the parameters in the signature verification process. While the use of a nonce can prevent replay attacks within the same network by ensuring each signature is unique for a particular transaction, it does not safeguard against replay attacks across different networks. Without the inclusion of `chainId`, a legitimate signature on one blockchain could be maliciously reused on another chain, potentially resulting in unintended or unauthorized token transfers, thus exposing the contract to cross-network vulnerabilities.

```
function withdrawRewards(
    address _identity,
    uint _amount,
    bytes calldata _data,
    LZSendParam calldata _lzSendParam
    ) external payable {
    // Validate proof of withdrawal
    (uint totalAmount, bytes32 message, bytes memory signature) =
abi.decode(_data, (uint, bytes32, bytes));
    bytes32 expectedMessage =
keccak256(abi.encodePacked('\x19Ethereum Signed Message:\n52',
_identity, totalAmount));
    require(message == expectedMessage, 'Invalid proof of
rewards');
    require(message.recover(signature) == tokenAdmin, 'Invalid
proof signer');

    ...
}
```

## Recommendation

It is recommended to incorporate the `chainId` in the signature verification process by including it in the parameters hashed during the signature construction. By doing so, the signatures will be explicitly tied to a specific network, effectively preventing them from being reused across different chains.

## TSI - Tokens Sufficiency Insurance

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | TokenEmissionsController.sol#L223,232<br>UniV3IncentivesController.sol#L214,224 |
| **Status** | Acknowledged |

### Description

The owner can call the `addReward` function to include a new token in the reward mechanism. However, this does not ensure that the contract holds the necessary funds to distribute the rewards. The owner must also call the `notifyReward` function to deposit the rewards. If the rewards for an initialized reward token are not present, it could lead to transaction failures.

```
function addReward(address _rewardToken) external onlyAdmin {
        _addReward(_rewardToken);
}
```

```
function notifyReward(address[] calldata _rewardTokens, uint[]
calldata _amounts, uint _rewardsDuration) external onlyAdmin {
        require(_rewardsDuration > 0, 'Duration is zero');
        require(_rewardTokens.length == _amounts.length, 'Invalid
input');
        _updateReward(address(this), _rewardTokens);
        uint length = _rewardTokens.length;
        for (uint i; i < length; i++) {
            address token = _rewardTokens[i];
            if (token == rewardTokens[0]) continue;
            Reward storage r = rewardData[token];
            require(r.periodFinish > 0, 'Unknown reward token');
            IERC20(token).safeTransferFrom(msg.sender,
address(this), _amounts[i]);
            uint unseen =
IERC20(token).balanceOf(address(this)).sub(r.balance);
            _notifyReward(token, unseen, _rewardsDuration);
            r.balance = r.balance.add(unseen);
        }
}
```

## Recommendation

It is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution is to deposit the tokens by calling the notifyReward function at the time of the creation. If the contract guarantees the process it can enhance its reliability, security, and participant trust, ultimately leading to a more successful and efficient process.

## Team Update

The team has acknowledged that this is not a security issue and states:

*The process is intentionally split to two methods to align with offchain processes*

## L04 - Conformance to Solidity Naming Conventions

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | UniV3IncentivesController.sol#L109,142,159,170,185,194,207,214,224<br>TokenProxy.sol#L26,37,44<br>TokenEmissionsController.sol#L71,94,117,183,196,204,223,233<br>RewardDistributorV1LZ.sol#L120,121,122,123,124,125,126,127,160,161,162,163,164,165,166,179,202,203,204,205,206,207,208,209,223,242,243,244,245,301,302,303,339,349,359,369,379,389,399,408,421 |
| **Status** | Acknowledged |

### Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
uint[] calldata _nftIds
address[] calldata _rewardTokens
address _account
address _rewardsToken
int24 _tickLower
int24 _tickUpper
address _rewardToken
uint _rewardsDuration
uint[] calldata _amounts
ITokenEmissionsController.LockTime _lock
uint _amount
address _controller
EmissionPoint[] memory _emissions
address _onBehalfOf

...
```

## Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/stable/style-guide.html#naming-conventions.

## L16 - Validate Variable Setters

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | TokenEmissionsController.sol#L65<br>RewardDistributorV1LZ.sol#L135,136 |
| **Status** | Acknowledged |

### Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
withdrawingAdmin = _withdrawingAdmin
tokenAdmin = _tokenAdmin
feeReceiver = _feeReceiver
```

### Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

## L19 - Stable Compiler Version

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | interfaces/ITokenProxy.sol#L3 |
| **Status** | Acknowledged |

### Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.22;
```

### Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

# Functions Analysis

| Contract | Type | Bases | | |
|---|---|---|---|---|
| | Function Name | Visibility | Mutability | Modifiers |
| | | | | |
| **UniV3IncentivesController** | Implementation | ERC721Holder, Adminable | | |
| | | Public | ✓ | Ownable |
| | deposit | External | ✓ | - |
| | withdraw | External | ✓ | - |
| | getReward | External | ✓ | - |
| | getAllUserNfts | External | | - |
| | lastTimeRewardApplicable | Public | | - |
| | claimableRewards | External | | - |
| | changePositionRanges | External | ✓ | onlyOwner |
| | addReward | External | ✓ | onlyAdmin |
| | notifyReward | External | ✓ | onlyAdmin |
| | _getReward | Internal | ✓ | |
| | _rewardPerToken | Internal | | |
| | _earned | Internal | | |
| | _addReward | Internal | ✓ | |
| | _notifyReward | Internal | ✓ | |
| | _updateReward | Internal | ✓ | |

| | | | | |
|---|---|---|---|---|
| | | | | |
| **TokenProxy** | Implementation | ERC20, Ownable, ITokenProxy | | |
| | | Public | ✓ | ERC20 |
| | deposit | External | ✓ | - |
| | withdraw | External | ✓ | - |
| | setController | External | ✓ | onlyOwner |
| | | | | |
| **TokenEmissionsController** | Implementation | ITokenEmissionsController, Adminable | | |
| | | Public | ✓ | Adminable Ownable |
| | startEmissions | External | ✓ | onlyOwner |
| | deposit | External | ✓ | - |
| | withdraw | External | ✓ | - |
| | stakeNFT | External | ✓ | - |
| | unstakeNFT | External | ✓ | - |
| | getReward | External | ✓ | - |
| | lastTimeRewardApplicable | Public | | - |
| | claimableRewards | External | | - |
| | addReward | External | ✓ | onlyAdmin |
| | notifyReward | External | ✓ | onlyAdmin |
| | _setRewardsDuration | Internal | ✓ | |
| | _getReward | Internal | ✓ | |

| | | | | |
|---|---|---|---|---|
| | _rewardPerToken | Internal | | |
| | _earned | Internal | | |
| | _addReward | Internal | ✓ | |
| | _notifyReward | Internal | ✓ | |
| | _updateReward | Internal | ✓ | |
| | _getMultiplier | Internal | | |
| | | | | |
| **RewardableLZ** | Implementation | OFT, ERC20Permit, ERC20Burnable | | |
| | | Public | ✓ | OFT ERC20Permit Ownable |
| | | | | |
| **IOFTEndpointGetter** | Interface | | | |
| | endpoint | External | | - |
| | | | | |
| **RewardDistributorV1LZ** | Implementation | UUPSUpgradeable, OwnableUpgradeable | | |
| | | Public | ✓ | - |
| | initialize | Public | ✓ | initializer |
| | createTaskWithPermit | External | ✓ | - |
| | createTask | Public | ✓ | - |
| | topUpTaskWithPermit | External | ✓ | - |
| | topUpTask | Public | ✓ | - |

| | | | | |
|---|---|---|---|---|
| | withdrawRewards | External | Payable | - |
| | quoteSend | External | | - |
| | getSameChainLZSendParam | External | | - |
| | setSponsorAdmin | External | ✓ | onlyOwner |
| | setTokenAdmin | External | ✓ | onlyOwner |
| | setFeeReceiver | External | ✓ | onlyOwner |
| | setBurnFee | External | ✓ | onlyOwner |
| | setPlatformFee | External | ✓ | onlyOwner |
| | setMaxDailyWithdrawal | External | ✓ | onlyOwner |
| | setMinWithdrawalAmount | External | ✓ | onlyOwner |
| | burnFees | External | ✓ | onlyOwner |
| | recoverFees | External | ✓ | onlyOwner |
| | _processFees | Internal | ✓ | |
| | _authorizeUpgrade | Internal | ✓ | onlyOwner |
| | _addressToBytes32 | Internal | | |
| | | | | |
| **Adminable** | Implementation | Ownable | | |
| | _checkAdmin | Internal | | |
| | addAdmin | External | ✓ | onlyOwner |
| | removeAdmin | External | ✓ | onlyOwner |
| | isAdmin | External | | - |
| | | | | |

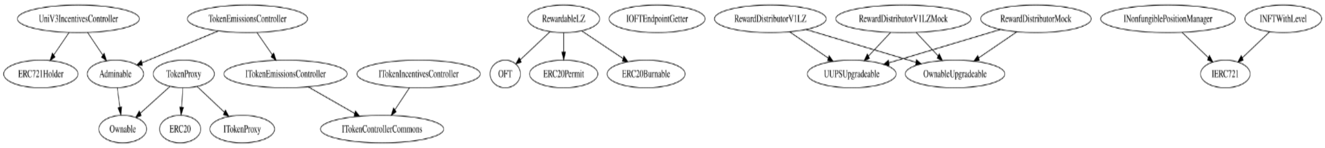
| | | | | |
|---|---|---|---|---|
| **IOFTEndpointGetter** | Interface | | | |
| | endpoint | External | | - |
| | | | | |
| **RewardDistributorV1LZMock** | Implementation | UUPSUpgradeable, OwnableUpgradeable | | |
| | | Public | ✓ | - |
| | initialize | Public | ✓ | initializer |
| | createTaskWithPermit | External | ✓ | - |
| | createTask | Public | ✓ | - |
| | topUpTaskWithPermit | External | ✓ | - |
| | topUpTask | Public | ✓ | - |
| | withdrawRewards | External | Payable | - |
| | quoteSend | External | | - |
| | getSameChainLZSendParam | External | | - |
| | setSponsorAdmin | External | ✓ | onlyOwner |
| | setTokenAdmin | External | ✓ | onlyOwner |
| | setFeeReceiver | External | ✓ | onlyOwner |
| | setBurnFee | External | ✓ | onlyOwner |
| | setPlatformFee | External | ✓ | onlyOwner |
| | setMaxDailyWithdrawal | External | ✓ | onlyOwner |
| | addToTest | External | ✓ | onlyOwner |
| | setMinWithdrawalAmount | External | ✓ | onlyOwner |

| | | | | |
|---|---|---|---|---|
| | burnFees | External | ✓ | onlyOwner |
| | recoverFees | External | ✓ | onlyOwner |
| | _processFees | Internal | ✓ | |
| | _authorizeUpgrade | Internal | ✓ | onlyOwner |
| | _addressToBytes32 | Internal | | |
| | | | | |
| **RewardDistributorMock** | Implementation | UUPSUpgradeable, OwnableUpgradeable | | |
| | | Public | ✓ | - |
| | initialize | Public | ✓ | initializer |
| | createTaskWithPermit | External | ✓ | - |
| | createTask | Public | ✓ | - |
| | topUpTaskWithPermit | External | ✓ | - |
| | topUpTask | Public | ✓ | - |
| | withdrawRewards | External | ✓ | - |
| | setTokenAdmin | External | ✓ | onlyOwner |
| | setFeeReceiver | External | ✓ | onlyOwner |
| | setBurnFee | External | ✓ | onlyOwner |
| | setPlatformFee | External | ✓ | onlyOwner |
| | setMaxDailyWithdrawal | External | ✓ | onlyOwner |
| | addToTest | External | ✓ | onlyOwner |
| | setMinWithdrawalAmount | External | ✓ | onlyOwner |
| | burnFees | External | ✓ | onlyOwner |

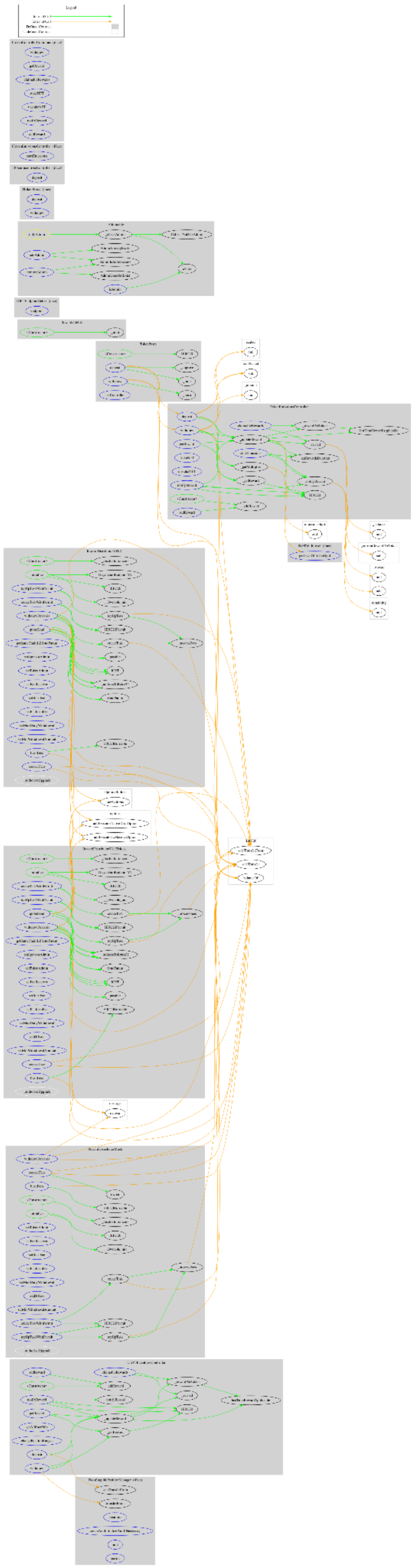
| | | | | |
|---|---|---|---|---|
| | recoverFees | External | ✓ | onlyOwner |
| | _processFees | Internal | ✓ | |
| | _authorizeUpgrade | Internal | ✓ | onlyOwner |
| | | | | |
| **ITokenProxy** | Interface | | | |
| | deposit | External | ✓ | - |
| | withdraw | External | ✓ | - |
| | | | | |
| **ITokenIncentivesController** | Interface | ITokenControllerCommons | | |
| | deposit | External | ✓ | - |
| | | | | |
| **ITokenEmissionsController** | Interface | ITokenControllerCommons | | |
| | deposit | External | ✓ | - |
| | startEmissions | External | ✓ | - |
| | | | | |
| **ITokenControllerCommons** | Interface | | | |
| | withdraw | External | ✓ | - |
| | getReward | External | ✓ | - |
| | claimableRewards | External | | - |
| | stakeNFT | External | ✓ | - |
| | unstakeNFT | External | ✓ | - |

| | | | | |
|---|---|---|---|---|
| | notifyReward | External | ✓ | - |
| | addReward | External | ✓ | - |
| | | | | |
| **INonfungiblePositionManager** | Interface | IERC721 | | |
| | positions | External | | - |
| | createAndInitializePoolIfNecessary | External | ✓ | - |
| | mint | External | Payable | - |
| | permit | External | Payable | - |
| | | | | |
| **INFTWithLevel** | Interface | IERC721 | | |
| | getLevelOfTokenById | External | | - |

# Inheritance Graph



# Flow Graph



# Summary

Rewardable is an interesting project that has a friendly and growing community. Its contracts implement a staking mechanism with automated reward distribution. The Smart Contract analysis reported no compiler error or critical issues. This audit investigates security issues, business logic concerns and potential improvements. The team has acknowledged the findings.

# Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

# About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

**The Cyberscope team**

cyberscope.io