Tally Reward Notifier Review

January 25, 2025

Prepared for DappHero Corp

Conducted by:

Richie Humphrey (devtooligan)

About the Tally Reward Notifier Review

Tally is a blockchain-based governance platform that enables organizations to manage decision-making through transparent, decentralized processes. Using the OpenZeppelin Governor framework, it provides infrastructure for onchain voting and proposal systems while ensuring community participation and preventing centralized control.

This Reward Notifier updates add new functionality to the `Staker` contract, a contract which allows users to delegate the voting power of their staked tokens.

About Offbeat Security

Offbeat Security is a boutique security company providing unique security solutions for complex and novel crypto projects. Our mission is to elevate the blockchain security landscape through invention and collaboration.

Summary & Scope

The following folders and files from src/ were reviewed at commit 51c66fe:

- src/notifiers/
    - RewardTokenNotifierBase.sol
    - TransferRewardNotifier.sol
    - TransferFromRewardNotifier.sol
    - MintRewardNotifier.sol
- src/calculators/IdentityEarningPowerCalculator.sol

The project implements reward notification functionality through a base notifier contract inherited by three variants that use different mechanisms to transfer the rewards: direct transfers, transfers from a designated source, and minting new tokens. The design allows for different reward distribution mechanisms while enforcing consistent rules around notification intervals and amounts.

Also in scope is a simple earning power calculator used by the staker for basic scenarios that don't use any conditional logic or apply any multipliers to the staked amount.

Identifier	Title	Severity	Fixed
M-01	SafeERC20 library imported but not used, bypassing critical safety checks	Medium	PR#105
L-01	Missing input validation in RewardTokenNotifier contracts	Low	PR#107
L-02	Reward notifier can be irrevocably disabled by setting large reward interval	Low	PR#108

Detailed Findings

Medium Findings

[M-01] SafeERC20 library imported but not used, bypassing safety checks

While the reward notifier contracts import OpenZeppelin's SafeERC20 library, they bypass its safety mechanisms by making direct ERC20 calls. This circumvents essential return value checks and safety validations designed to prevent failed transfers.

All three reward notifier contracts reference `SafeERC20` but do not utilize any of its functions, making direct IERC20 calls instead:

TransferRewardNotifier uses direct `transfer`:

```
function _sendTokensToReceiver() internal virtual override {
    TOKEN.transfer(address(RECEIVER), rewardAmount);
}
```

TransferFromRewardNotifier uses direct `transferFrom`:

```
function _sendTokensToReceiver() internal virtual override {
    TOKEN.transferFrom(rewardSource, address(RECEIVER), rewardAmount);
}
```

TransferRewardNotifier uses direct `approve`:

```
function approve(address _spender, uint256 _amount) external {
    _checkOwner();
    TOKEN.approve(_spender, _amount);
    emit Approved(_spender, _amount);
}
```

Severity Discussion

The impact from this could potentially be high because bypassing SafeERC20's safety checks means failed transfers may not be detected breaking core protocol functionality, as rewards that appear to be distributed may actually fail without the contracts being aware of the failure.

The likelihood is low since the DAO must first select a non-standard token as the reward token, or the token contract must be upgraded to a non-standard implementation. However, since these contracts are designed to be used by anyone with a variety of use cases, this issue deserves consideration.

Based on the impact and likelihood analysis, this finding has a severity of Medium.

Recommendation

Replace all direct ERC20 calls with their SafeERC20 equivalents to ensure failed transfers are properly detected and handled:

```
// In TransferRewardNotifier
function _sendTokensToReceiver() internal virtual override {
-   TOKEN.transfer(address(RECEIVER), rewardAmount);
+   TOKEN.safeTransfer(address(RECEIVER), rewardAmount);
}

// In TransferFromRewardNotifier
function _sendTokensToReceiver() internal virtual override {
```

```diff
-    TOKEN.transferFrom(rewardSource, address(RECEIVER), rewardAmount);
+    TOKEN.safeTransferFrom(rewardSource, address(RECEIVER), rewardAmount);
}

// In TransferRewardNotifier
function approve(address _spender, uint256 _amount) external {
    _checkOwner();
-    TOKEN.approve(_spender, _amount);
+    TOKEN.safeApprove(_spender, _amount);
    emit Approved(_spender, _amount);
}
```

Low Findings

[L-01] Missing input validations in RewardTokenNotifier contracts

The `RewardTokenNotifier` contracts lack several sanity checks:

1. `rewardAmount` is not checked to be greater than 0 in `RewardTokenNotifierBase._setRewardAmount`. This could result in unnecessary calls to the notify function which would waste gas and spam events.

2. Zero address checks are missing for contract parameters. An address inadvertently set to the zero address may result in unexpected behavior requiring redeployment or making them unusable.
    - `RECEIVER` in base contract constructor
    - `TOKEN` in base contract constructor
    - `rewardSource` in `TransferFromRewardNotifier._setRewardSource`
    - `minter` in `MintRewardNotifier._setMinter`

Recommendation

Add input validation checks where appropriate.

[L-02] Reward notifier can be irrevocably disabled by setting large reward interval

The reward notifier contract has two related issues with its interval configuration that could lead to extended delays in reward distributions:

1. The contract uses an absolute `nextRewardTime` timestamp rather than tracking relative to the last distribution. This means that if a large interval is accidentally set

and `notify()` is called, the contract remains locked to that schedule even if the interval is later corrected.

2. The reward interval can be set to any value without an upper bound.

Example scenario:

1. Admin accidentally sets interval to 300 months instead of 3 months
2. `notify()` is called, setting `nextRewardTime` far in the future
3. Admin corrects interval but `nextRewardTime` remains locked
4. Rewards are blocked until the 300 month period elapses

The impact of this is heightened by the fact that the mistake might not be caught until it's too late, since setting the reward interval does not affect the very next reward distribution, but the time after that.

We also noted that these issues are somewhat mitigated by several factors:

- This contract is intended to be used with a governance system that would have to approve calls to `setRewardInterval`.
- Due to the permissioned `approve` function, no funds would be lost and a new contract could be deployed if needed.
- The use of the explicit `nextRewardTime` could be seen as a feature as it makes the schedule immutable and transparent.

Recommendation

Consider implementing the following improvements:

1. Track the last reward time instead of next reward time:

```
abstract contract RewardTokenNotifierBase is Ownable {
-   uint256 public nextRewardTime = 0;
+   uint256 public lastRewardTime = 0;

    function notify() external virtual {
-       if (block.timestamp < nextRewardTime) {
+       uint256 nextRewardTime = lastRewardTime + rewardInterval;
+       if (block.timestamp < nextRewardTime) {
            revert RewardTokenNotifierBase__RewardIntervalNotElapsed();
        }
-       nextRewardTime = block.timestamp + rewardInterval;
+       lastRewardTime = block.timestamp;
        _sendTokensToReceiver();
        RECEIVER.notifyRewardAmount(rewardAmount);
        emit Notified(rewardAmount, nextRewardTime);
    }
}
```

2. Add a maximum interval limit:

```
+ error RewardTokenNotifierBase__InvalidRewardInterval(uint256 interval);

+ // Maximum reward interval of 1 year
+ uint256 public constant MAX_REWARD_INTERVAL = 365 days;

function _setRewardInterval(uint256 _newRewardInterval) internal {
+    if (_newRewardInterval > MAX_REWARD_INTERVAL) {
+        revert RewardTokenNotifierBase__InvalidRewardInterval(_newRewardInterval
+    }
    emit RewardIntervalSet(rewardInterval, _newRewardInterval);
    rewardInterval = _newRewardInterval;
}
```

3. At a minimum, in the current design which uses `nextRewardTime`, add documentation clearly stating that once set it is set in `notify()` it cannot be reset. This helps alert integrators that a misconfigured interval could disable the contract.

Appendix A. Code Quality Recommendations

1. `TransferRewardNotifier` emits an unnecessary `Approved` event since the ERC20 token contract will already emit this event.
2. Consider using OpenZeppelin's `Ownable2Step` pattern instead of `Ownable` for safer ownership transfers.