InfraredVault Security Vulnerabilities Report

Vulnerability #1: Delisted Reward Tokens Can Still Earn Incentives

MEDIUM SEVERITY

Summary

The addIncentives function in InfraredVault.sol checks reward tokens by confirming if they were previously added through rewardData, but it does not verify their current status on the whitelist. This oversight means that tokens that have been delisted can still receive incentives, which could put users at risk from compromised or vulnerable tokens that were deliberately removed from the protocol.

Finding Description

The addIncentives function in InfraredVault.sol has a vulnerability in its token validation process that could enable malicious users to bypass the protocol's token whitelisting security measures. The function only verifies if the token was previously included as a reward token via rewardData, but it does not check whether the token is currently whitelisted using isWhitelisted.

Impact Explanation

> 1. The whitelisting mechanism is bypassed as a result of this issue

Proof of Concept

A coded function in the InfraredVaultTest:

```
function testAddIncentivesWithDelistedRewardToken() public {
    // First whitelist and add a reward token
    address rewardToken = address(new MockERC20("Reward", "RWD", 18));
    uint256 rewardAmount = 100 ether;
    uint256 rewardsDuration = 7 days;

    vm.startPrank(infraredGovernance);
    infrared.updateWhiteListedRewardTokens(rewardToken, true);
    infrared.addReward(address(wbera), rewardToken, rewardsDuration);
    vm.stopPrank();

    // Verify reward token is properly set up
    (, uint256 duration,,,,) = infraredVault.rewardData(rewardToken);
    assertEq(duration, rewardsDuration, "Reward duration should be set");

    // Delist the reward token
    vm.prank(infraredGovernance);
    infrared.updateWhiteListedRewardTokens(rewardToken, false);

    // Demonstrate we can still add incentives despite token being delisted
    deal(rewardToken, address(this), rewardAmount);
    ERC20(rewardToken).approve(address(infrared), rewardAmount);

    // This should revert but doesn't
    infrared.addIncentives(address(wbera), rewardToken, rewardAmount);

    // Verify incentives were added despite token not being whitelisted
    uint256 vaultBalance = ERC20(rewardToken).balanceOf(address(infraredVault))
    assertEq(
        vaultBalance, rewardAmount, "Incentives added for delisted token"
    );
}
```

Recommendation

1. Add a whitelist check in addIncentives function before moving forward with token reward processing

> 2. Add a pause mechanism in case of emergencies

Vulnerability #2: Inflexible Token Limit Leads to Tokens Locked in Reward System

LOW SEVERITY

Summary

The InfraredVault contract has a strict cap of 10 reward tokens (MAX_NUM_REWARD_TOKENS) for each vault. When this cap is hit, no additional reward tokens can be added or taken away, which effectively locks the reward system in its existing state. This limitation hinders the protocol's ability to adjust to new reward strategies or token integrations without the need to deploy new vaults.

Finding Description

In InfraredVault.sol, there is a limitation relating reward token management that could result in token being stuck within the protocol. The contract has a constant called MAX_NUM_REWARD_TOKENS set to 10, and once this cap is reached, there is no way to remove or replace the existing reward tokens.

There is no mechanism to remove reward tokens. Since tokens can only be added until the maximum is reached, and there is no removal option, once the limit is hit, the protocol loses all ability to adapt to new needs or changes in tokens.

```
function addReward(address _rewardsToken, uint256 _rewardsDuration)
    external
    onlyInfrared
{
    if (_rewardsToken == address(0)) revert Errors.ZeroAddress();
    if (_rewardsDuration == 0) revert Errors.ZeroAmount();
    if (rewardTokens.length == MAX_NUM_REWARD_TOKENS) {
```

```
            revert Errors.MaxNumberOfRewards();
        }
        _addReward(_rewardsToken, infrared, _rewardsDuration);
    }
```

Impact Explanation

1. If a reward token is added but later becomes an issue, the protocol is stuck with it since there's no removal mechanism
2. Unusable tokens take up storage space and resources since they are part of the reward token

Recommendation

1. Add a method to remove reward tokens