Prophet Bot Revenue Sharing Security Review

Reviewers
Reviewer 1, Nirlin

July 5, 2024

1 Executive Summary

Total Issues

Critical Risk	0
High Risk	1
Medium Risk	1
Low Risk	2
Gas Optimizations and Informational	0

Contents

2 Nirlin Security

Nirlin is a smart contract auditor with over a year of experience in reviewing DeFi protocols. He has identified numerous critical vulnerabilities before the code goes into production.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of brink according to the specific commit by a three person team. Any modifications to the code will require a new security review.

3 Findings

3.1 High Risk

3.1.1 Reward Duration can never be changed.

Severity: High

Description:

There is setter function `setRewardDuration` that changes the `rewardsDuration` initially set to 2 days, in original synthetic implementation `rewardsDuration` function is public and owner controlled so owner can change the epoch length but in current implementation it is private which means owner will never be able to change it to some other value in future.

```
    function setRewardsDuration(uint256 _rewardsDuration) private {
        if (block.timestamp < periodFinish) {
            revert CannotUpdateRewardsDuration();
        }
        rewardsDuration = _rewardsDuration;
        emit RewardsDurationUpdated(rewardsDuration);
    }
```

Recommendation: Change the function as following

```
    function setRewardsDuration(uint256 _rewardsDuration) external onlyOwner {
        if (block.timestamp < periodFinish) {
            revert CannotUpdateRewardsDuration();
        }
        rewardsDuration = _rewardsDuration;
        emit RewardsDurationUpdated(rewardsDuration);
    }
```

3.2 Medium Risk

3.2.1 Rewards for the initial period are lost

Severity: Medium

Description: After the epoch is started and funds have been sent to the contract, if their is no staker for the initial period, staking tokens will be lost for that period and no one will be able to claim it.

in original sunthetix implementation the work around for this was to recover those tokens using the following function

```
    function recoverERC20(
        address tokenAddress,
        uint256 tokenAmount
    ) external onlyOwner {
        if (tokenAddress == address(stakingToken)) {
            revert TokenCannotBeRewardToken();
        }
        IERC20(tokenAddress).safeTransfer(owner(), tokenAmount);
        emit Recovered(tokenAddress, tokenAmount);
    }
```

but in original implementation the reward token is erc20 instead in prophet bots it's ether, which means ether can never be recovered using this function hence those are permanently stuck in the contract.

You can read more about this bug here:

https://0xmacro.com/blog/synthetix-staking-rewards-issue-inefficient-reward-distribution/

Recommendation: you can either implement the solution given by macro that is a bit complex IMO, or simply add function to also recover ether from the contract.

3.3 Low Risk

3.3.1 Permanent DOS of adding rewards during the epoch.

Severity: Low

Description: `notifyRewardAmount` function do two things, add the rewards after the epoch is ended or just add it while an epoch is running extend it making it addition time the start time. Look at the following snippet

```
  function notifyRewardAmount(
      uint256 reward
   ) private updateReward(address(0)) {
      if (block.timestamp >= periodFinish) {
          rewardRate = reward / rewardsDuration;
      } else {
          uint256 remaining = periodFinish - block.timestamp;
          uint256 leftover = remaining * rewardRate;
          rewardRate = (reward + leftover) / rewardsDuration;
      }

      uint256 balance = (address(this).balance);
      if(rewardRate > balance / rewardsDuration) revert RewardRateTooHigh();

      lastUpdateTime = block.timestamp;
      periodFinish = block.timestamp + rewardsDuration;
      emit RewardAdded(reward);
   }
```

For the new epoch when the previous have been finished following if block is triggered:

```
  if (block.timestamp >= periodFinish) {
          rewardRate = reward / rewardsDuration;
      }
```

but if the epoch is not finished and owner adds the rewards following else block is triggered:

```
else {
          uint256 remaining = periodFinish - block.timestamp;
          uint256 leftover = remaining * rewardRate;
          rewardRate = (reward + leftover) / rewardsDuration;
      }
```

where the leftover is calculated and considering it and new reward new reward rate is calculated and from here on new epoch start. But in our code, this function is triggered in `receive` function as follow

```
receive() external payable {
     if (msg.sender != IProphetRouter(prophetRouter).owner()) {
         revert OnlyProphetRouterOwnerCanSendEther();
     }
     setRewardsDuration(2 days);
     notifyRewardAmount(msg.value);
 }
```

Another thing is `notifyRewardAmount` is private so it can't be called directly and in `setRewardsDuration` function there are following checks :

```
function setRewardsDuration(uint256 _rewardsDuration) private {
     if (block.timestamp < periodFinish) {
         revert CannotUpdateRewardsDuration();
     }
     rewardsDuration = _rewardsDuration;
     emit RewardsDurationUpdated(rewardsDuration);
 }
```

which means it can't be called before the finish period which means the else block discussed above can never be triggered because that is called before the finish period but the revert happens in `setRewardsDuration` for that condition.

Add the following in test and we are not able to add between epoch as it reverts when `setRewardDuration` is called

```
 function test_myreceive() public {
     warpTime(1 days);
     vm.startPrank(prophetRouterOwner);
     (bool sent, ) = address(revShare).call{value: 2.5 ether}("");
     require(sent, "Failed to send Ether");
     vm.stopPrank();
 }
```

changing 1 days to 2 days and the test case passes.

Recommendation: Don't call `setRewardDuration` in receive and it will enable the original syntehtix behavior.

3.4 Low Risk

3.4.1 Stake function have no 0 amount checks.

Severity: Low

Description:

In stake function amount is passed by the user, for the first stake this function ensure there is no zero input

```
      if (_balances[msg.sender] + stakeAmount < MINIMUM_STAKE) {
            revert InsufficientStakeAmount();
        }
```

but for all the subsequent deposits this check will still pass for the same user even if he pass amount as zero.

Recommendation: modify the function as follow

```
    function stake(
        uint256 amount
    ) external nonReentrant updateReward(msg.sender) {
        require(amount != 0, "amount is zero duh");
    -----
    -----
```

4 Appendix