WNCG Staking Protocol Security Audit

: Final Report

June 30th, 2022

Rev 1.0

Theori

Table of Contents

Executive Summary

Theori reviewed Planetarium's WNCG stacking contracts. These contracts allow users to stake Balancer liquidity pool tokens and earn rewards. The contracts are based on the AAVE safety module staking contracts and the Convex Finance rewards pool contract. We found one critical issue that could result in denial of service and loss of reward tokens.

Project Overview

Scope

Name	WNCG Staking
Target / Version	• Github Repository: https://github.com/planetarium/wncg-staking-contracts • **wncg-staking-contracts** Rev 1.0: 85e3f2f05abe9b0935f3b33a62d1156ec80b22a5
Application Type	Staking protocol
Lang. / Platforms	Smart contracts (Solidity)

Staking contracts

In order to use the Planetarium WNCG staking contracts, users must first provide liquidity to the Balancer WNCG/ETH 80/20 Pool. Typically, users earn BAL rewards by depositing these LP tokens into the Balancer gauge. Instead, users can deposit the LP tokens into the Planetarium staking contracts to earn both BAL and WNCG rewards.

Users call the `stake()` function in the `StakingRewards` contract to deposit Balancer LP tokens, which then get deposited to the Balancer gauge. This allows the staking contract to earn BAL rewards on behalf of users. The `StakingRewards` contract tracks the amount that each user deposits in order to fairly distribute the BAL rewards it collects from the gauge. Additional WNCG rewards are distributed to users based on their share of the staking pool. The WNCG rewards are initially provided by Planetarium and the amount of rewards available to be distributed is refreshed occasionally. Users can collect their rewards as they accumulate, but they can only unstake their LP tokens after a cooldown period.

Severity Categories

Severity	Description
Critical	The attack cost is low (not requiring much time or effort to succeed in the actual attack), and the vulnerability causes a high-impact issue. (e.g. Effect on service availability, Attacker taking financial gain)
High	An attacker can succeed in an attack which clearly causes problems in the service's operation. Even when the attack cost is high, the severity of the issue is considered "high" if the impact of the attack is remarkably high.
Medium	An attacker may perform an unintended action in the service, and the action may impact service operation. However, there are some restrictions for the actual attack to succeed.
Low	An attacker can perform an unintended action in the service, but the action does not cause significant impact or the success rate of the attack is remarkably low.

Issue Breakdown by Severity

Category	Count	Issues
Critical	1	• THE-WNCG-001
High	0	
Medium	0	
Low	0	

Findings

We found one critical issue that may lead to denial of service and improper distribution of rewards to an attacker.

Summary

ID	Summary	Severity	Status
THE-WNCG-001	An attacker can withdraw any users' deposit tokens from the reward pool in the `withdrawFor()` function resulting in possible denial of service and loss of reward tokens.	**Critical**	**Fixed** *(June 20, 2022)*

Issue #1 Lack of sender validation during withdraw (*withdrawFor*) of deposit tokens from reward pool

ID	Summary	Severity
THE-WNCG-001	An attacker can withdraw any users' deposit tokens from the reward pool in the *withdrawFor()* function resulting in possible denial of service and loss of reward tokens.	**Critical**

Description

The *StakingRewards* contract is the public facing smart contract for the WNCG staking protocol. Users call the *stake()* function to deposit Balancer liquidity pool tokens, and call the *withdraw()* function to withdraw the balancer liquidity pool tokens. While their tokens are staked, users accumulate reward tokens based on the number of staked tokens.

Internally, when a user calls the *stake()* function, the *StakingRewards* contract mints deposit tokens which are staked in the *BALRewardPool* contract using the *stakeFor()* function. When a user calls *withdraw()*, the deposit tokens are withdrawn from the *BALRewardPool* contract using the *withdrawFor()* function and burned. *BALRewardPool* is responsible for tracking the accumulated rewards for each user based on the number of deposit tokens staked.

However, the *withdrawFor()* function in *BALRewardPool* is missing validation of *msg.sender*. This allows anybody, including an attacker, to call the *withdrawFor()* function and withdraw deposit tokens for any user.

```
function withdrawFor(address _for, uint256 _amount) public nonReentrant
updateReward(_for) returns(bool) {
    require(_amount > 0, 'RewardPool : Cannot withdraw 0');

    _totalSupply = _totalSupply.sub(_amount);
    _balances[_for] = _balances[_for].sub(_amount);

    stakingToken.safeTransfer(msg.sender, _amount);
```

```
    emit Withdrawn(_for, _amount);

    return true;
}
```

BALRewardPool.sol#withdrawFor

While this is a critical issue, it does not allow an attacker to steal the staked Balancer liquidity pool tokens as the *StakingRewards* contract tracks the balance of each user internally. This issue does lead to denial of service as users will not be able to withdraw their staked tokens due to *withdrawFor()* reverting. Additionally, after the attacker steals the deposit tokens, they may be able to accrue rewards from the stolen deposit tokens.

Recommendations

- Validate in *withdrawFor* that *msg.sender* is trusted

Fix

Planetarium implemented the recommendation by adding a check that *msg.sender* is the *operator* address. They added the check to both *stakeFor* and *withdrawFor* as only the *StakingRewards* contract should be calling those functions.

```
function withdrawFor(address _for, uint256 _amount) public nonReentrant
updateReward(_for) returns(bool) {
    require(msg.sender == operator, "!authorized");
    require(_amount > 0, 'RewardPool : Cannot withdraw 0');

    ...
}
```

BALRewardPool.sol#withdrawFor

Revisions

Revision	Date	History
1.0	Jun 30, 2022	• Initial document.