Audit Report

DegenPad Security Review

Auditors

0xKING, Lead Security Researcher

Report prepared by: 0xKING

May 16, 2024

Contents

1 About 0xKing

0xking is an individual auditor who has performed well in various audit contests. Also, I found various vulnerabilities in bug bounties, such as finding vulnerabilities in pancakeswap. I have experience auditing not only solidity, but also vyper and cosmos SDKs.

2 Introduction

DegenPad is a launchpad project that is the first to offer an Initial Tip Offering (ITO) on farcaster. The scope of this audit is the staking contract that allows you to stake DegenPad's token, $DPAD, and manage user's points.

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

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 4 days in total, 0xKING review DegenPad. In this period of time a total of 2 issues were found.

Summary

Project Name	0xKING
Type of Project	Staking Contract
Audit Timeline	May 13th - May 16th
Commit Hash	00af6189efc5d...cb23bee18a216
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	2
Gas Optimizations	0
Informational	0
Total Issues	2

5 Findings

5.1 Low Risk

5.1.1 Unstaking after the snapshot time may cause a revert in getUserStakeInfos.

Context: DpadStaking.sol#L38

Description:

```
    function getUserStakeInfos(uint256 seasonId, address[] calldata _users) external view returns
    ↪  (StakeInfo[] memory){
        StakeInfo[] memory stakeInfos = new StakeInfo[](_users.length);
        SeasonInfo memory season = seasons[seasonId];
        for (uint i=0; i < _users.length; i++) {
            stakeInfos[i] = userStakes[seasonId][_users[i]];
            uint notCalculatedPoint;
            if (season.snapshotTime != 0 && season.snapshotTime < block.timestamp) {
@>              notCalculatedPoint = stakeInfos[i].amount * (season.snapshotTime -
↪   stakeInfos[i].lastUpdateTime);
            } else {
                notCalculatedPoint = stakeInfos[i].amount * (block.timestamp -
                ↪  stakeInfos[i].lastUpdateTime);
            }
            stakeInfos[i].expectedPointStored = stakeInfos[i].currentPointStored + notCalculatedPoint;
        }

        return stakeInfos;
    }
```

getUserStakeInfos is a function that settles and returns the points of each user. If the snapshotTime has passed, use this expression to calculate notCalculatedPoint. `stakeInfos[i].amount * (season.snapshotTime - stakeInfos[i].lastUpdateTime)`

```
    function unstake(uint256 seasonId, uint256 amount) external {
  ...
        userStake.amount -= amount;
@>      userStake.lastUpdateTime = block.timestamp;
        userStake.currentPointStored += calculatedPoint;

        season.totalAmount -= amount;
        season.totalPoint += calculatedPoint;
...
    }
```

Since lastUpdateTime is set to block.timestamp at unstake, it can be set to a larger value than snapshotTime. Therefore, the revert is caused by underflow in the point calculation.

Recommendation: If the snapshotTime is past when unstaking, it is recommended to set it to snapshotTime rather than block.timestamp.

DegenPad: Fixed as recommended.

5.1.2 When stake, the array users may contain duplicates of the same user.

Context: DpadStaking.sol#L118

Description:

```
    function stake(uint256 amount) external {
        SeasonInfo storage season = seasons[lastSeasonId];
        require(season.startTime <= block.timestamp, "DpadStaking: not start season");
        require(season.endTime > block.timestamp, "DpadStaking: already end season");

        StakeInfo storage userStake = userStakes[lastSeasonId][msg.sender];
        require(userStake.amount + amount >= minAmount, "DpadStaking: must be greater than minAmount");

        uint calculatedPoint = userStake.amount * (block.timestamp - userStake.lastUpdateTime);

        userStake.amount += amount;
        userStake.lastUpdateTime = block.timestamp;
        userStake.pointStored += calculatedPoint;

        season.totalAmount += amount;
        season.totalPoint += calculatedPoint;

@>      users[lastSeasonId].push(msg.sender); // @audit After every staking, msg.sender is added to the
↪   users array.
        DPAD.transferFrom(msg.sender, address(this), amount);

        emit Stake(msg.sender, amount, calculatedPoint);
    }
```

`users` is an array managed to hold the addresses of users currently participating in the season. In the current implementation, user addresses are stored in the array for every stake, which can add duplicate data.

Recommendation: It is recommended to manage users as an EnumerableSet.

DegenPad: Fixed as recommended.