



Vinci Security Review

May 17th, 2023

Report Prepared By:

Georgi Georgiev (Gogo), Independent Security Researcher

Table of contents

Disclaimer

Risk classification

- Impact
- Likelihood
- Actions required by severity level

Executive summary

- Summary
- Scope

Findings

- High severity
 - Super stakers will receive inflated rewards from penalty pot due to unaccounted debt.
 - Stakers will receive twice their rewards when they cross a checkpoint after a missed period.
 - Penalization of penalty pot rewards can be circumvented.
 - A portion of the funds for staking rewards will be locked in the LP staking contract.
 - Super staker rewards will be incorrectly sent to users that are not yet super stakers.
- Medium severity
 - Instant Vinci rewards are calculated with the wrong precision.
 - Weekly LP staking rewards are distributed from the wrong starting date.
- Low severity
 - Users can maintain a stake of just 1 wei to keep the super staker status.
 - The Vinci token operator has control over the whole project value.
 - Insufficient input validation for vesting parameters.
 - Wrong accounting for penalty pot rewards.
 - An unbounded loop may cause DoS for Vinci token vestings.
 - The LP staking operator can block staking functionality.
 - Wrong value is used for weekly rewards in sanity check.
- Informational
 - Storage variables can be marked constant or immutable.
 - Missing zero address checks.
 - Public functions can be marked external.
 - Unstaking 0 amount should not be allowed.
 - Function incorrect as per spec.
 - Unsafe ERC20 operations.
 - Thresholds order is not validated.
 - Typographical mistakes.

Disclaimer

Audits are a time, resource and expertise bound effort, where trained experts evaluate smart contracts using a combination of automated and manual techniques to find as many vulnerabilities as possible. Audits can show the presence of vulnerabilities but not their absence

Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost, or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behaviour, but there are no funds at risk.

Likelihood

- High - there is a direct attack vector, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only conditionally possible incentivized attack vector, but still relatively likely.
- Low - depends on too many or too unlikely assumptions or requires a huge stake by an attacker with little or no incentive.

Actions required by severity level

- Critical - client must fix the issue.
- High - client must fix the issue.
- Medium - client should fix the issue.
- Low - client could fix the issue.

Executive summary

Summary

Protocol name	Vinci
Repository	https://github.com/The-Digital-Renaissance/solidity
Commit hash	21f6946856c7b1d3c49e6c308df5a418b21e0834
Methods	Manual review

Scope

- smartcontract-stakingpools/contracts/LPstaking.sol
- smartcontract-stakingpools/contracts/vinciStaking.sol
- smartcontract-stakingpools/contracts/vinciToken.sol
- smartcontract-stakingpools/contracts/inheritables/checkpoints.sol
- smartcontract-stakingpools/contracts/inheritables/penaltyPot.sol
- smartcontract-stakingpools/contracts/inheritables/tiers.sol

Findings

High severity

H.01. Super stakers will receive inflated rewards from penalty pot due to unaccounted debt.

Description

The VinciStakingV1 contract inherits from PenaltyPot, which is responsible for managing the distribution of additional VINCI rewards from penalizations (`penaltyPot`) collected when users perform an `unstake`.

`allocationPerStakedVinci` is used to track the distributed rewards, similar to `rewardPerTokenStored` in Synthetix's StakingRewards contract. The `_bufferPenaltyPotAllocation` function calculates the rewards that a staker accumulates based on their `individualAllocationTracker` and the current amount of tokens staked:

```
uint256 allocation = (_stakingBalance *  
    (allocationPerStakedVinci - individualAllocationTracker[user])) /  
    PENALTYPOT_ROUNDING_FACTOR;  
  
individualAllocationTracker[user] = allocationPerStakedVinci;
```

The vulnerability here arises from the fact that the `individualAllocationTracker` is not initialized for users when they become super stakers. Therefore, when each staker claims their first reward from the penalty pot, it will be significantly higher than expected.

Recommended mitigation steps

To fix the issue, the following line could be added in VinciStakingV1's `_crossCheckpoint` to initialize the `individualAllocationTracker` of super stakers before their stake is added to the supply eligible for allocation:

```
function _crossCheckpoint(address user) internal {  
    ...  
    if (!_isSuperstaker(user)) {  
        // @audit initialize individualAllocationTracker  
        _bufferPenaltyPotAllocation(user, 0);  
        _addToElegibleSupplyForPenaltyPot(activeStake);  
    }  
    ...  
}
```

H.02. Stakers will receive twice their rewards when they cross a checkpoint after a missed period.

Description

VinciStakingV1.`_crossCheckpoint` is called by stakers after a staking period has passed. However, a user may not call this function for more than one period, which means that it should also account for the rewards missed by the user during the previous periods. To handle this case, `_crossCheckpoint` calculates the rewards for the missed periods and adds them directly to the claimable rewards balance since the staking periods are already in the past:

```
function _crossCheckpoint(address user) internal {
    ...
    (uint256 missedPeriod, uint256 newCheckpoint) = _postponeCheckpoint(user);

    if (missedPeriod > 0) {
        uint256 missedRewards = _estimatePeriodRewards(activeStake, missedPeriod);
        ...
        claimableAddition += missedRewards;
        ...
    }

    if (claimableAddition > 0) {
        claimableBalance[user] += claimableAddition;
    }
    ...
}
```

The problem is that, even though the `missedRewards` are accounted for and moved directly to the `claimableBalance`, a few lines below they are accounted for a second time, as the rewards for the current (next) period are calculated using the difference between the newly updated checkpoint and the initially cached one:

```
uint256 rewards = _estimatePeriodRewards(activeStake, newCheckpoint - previousNextCheckpoint);
```

This means that stakers who wait more than one period before they cross their checkpoint will receive twice their `missedRewards` at the end of the next period.

Recommended mitigation steps

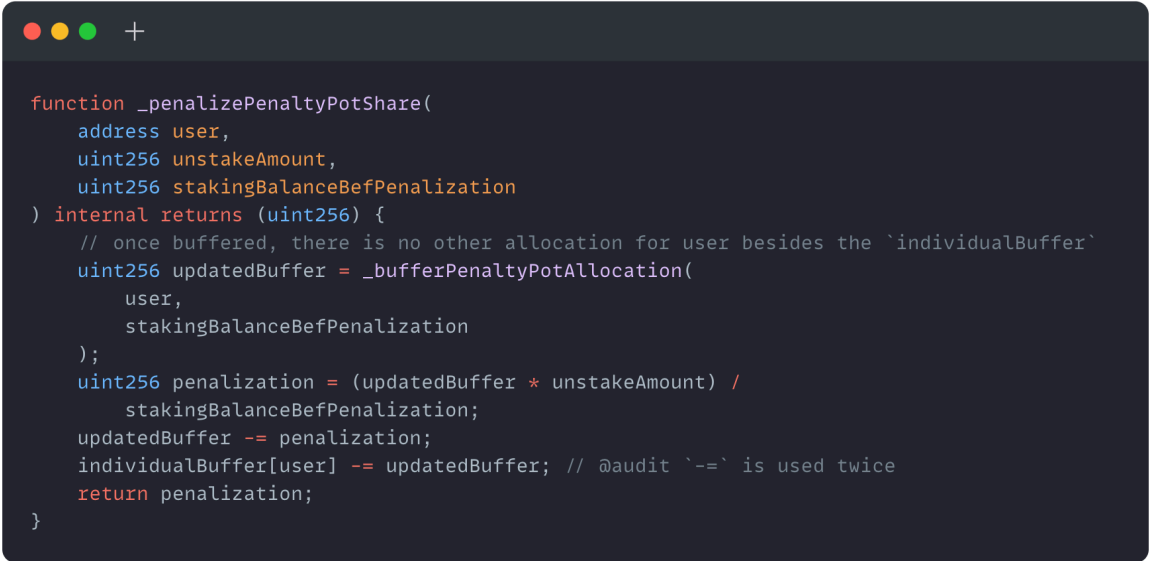
Consider returning the previous checkpoint along with `missedPeriod` from `_postponeCheckpoint` and use it instead of `previousNextCheckpoint` when calling `_estimatePeriodRewards`.

H.03. Penalization of penalty pot rewards can be circumvented.

Description

PenaltyPot. `_penalizePenaltyPotShare` takes a penalty from the penalty shares a staker owns from the pot when they decide to `unstake` a certain amount of their VINCI tokens. The penalty taken from their shares is calculate using the ratio of the unstaked amount to the total staked amount.

However, a mistake has been made when updating the `individualBuffer` value in storage, as `-=` is used instead of just `=`:

A screenshot of a code editor window with a dark theme. The window has three colored window control buttons (red, yellow, green) and a plus sign in the top-left corner. The code is a Solidity function named `_penalizePenaltyPotShare` that takes three arguments: `address user`, `uint256 unstakeAmount`, and `uint256 stakingBalanceBefPenalization`. The function is marked as `internal` and returns a `uint256`. It contains a comment: `// once buffered, there is no other allocation for user besides the 'individualBuffer'`. The logic calculates an `updatedBuffer` using `_bufferPenaltyPotAllocation`, then calculates a `penalization` as `(updatedBuffer * unstakeAmount) / stakingBalanceBefPenalization`. It then updates `updatedBuffer` and `individualBuffer[user]` using the `-=` operator. A comment at the end of the function states: `// @audit '-=' is used twice`.

```
function _penalizePenaltyPotShare(
    address user,
    uint256 unstakeAmount,
    uint256 stakingBalanceBefPenalization
) internal returns (uint256) {
    // once buffered, there is no other allocation for user besides the `individualBuffer`
    uint256 updatedBuffer = _bufferPenaltyPotAllocation(
        user,
        stakingBalanceBefPenalization
    );
    uint256 penalization = (updatedBuffer * unstakeAmount) /
        stakingBalanceBefPenalization;
    updatedBuffer -= penalization;
    individualBuffer[user] -= updatedBuffer; // @audit '-=' is used twice
    return penalization;
}
```

Therefore, the ratio is actually inverted, meaning that stakers who unstake a small amount of their tokens will be penalized less than those who unstake their whole VINCI stake.

Recommended mitigation steps

Use `=` instead of `-=`.

H.04. A portion of the funds for staking rewards will be locked in the LP staking contract.

Description

The VinciLPStaking contract pays two types of rewards to stakers - instant and staking rewards. Therefore, two storage variables are used to track the amounts of VINCI tokens funded for both types - `fundsForStakingRewards` and `fundsForInstantPayouts`.

When `distributeWeeklyAPR` is called, `fundsForStakingRewards` is decreased since a certain amount of tokens has been distributed among the `totalStaked` amount, and `vinciRewardsPerLP` is updated to account for the increase in distributed rewards:

```
function distributeWeeklyAPR() external {
    ...
    uint256 weeklyVinciDistribution = WEEKLY_VINCI_REWARDS + bufferedDecimals;
    uint256 rewardsPerLPToken = weeklyVinciDistribution / totalStaked;
    ...
    vinciRewardsPerLP += rewardsPerLPToken;
    // keep track of spent funds in rewards
    fundsForStakingRewards -= rewardsPerLPToken * totalStaked;
    ...
}
```

The problem is that the even though `fundsForStakingRewards` is decreased here to keep track of the spent funds for rewards, `fundsForStakingRewards` is also decreased in `_sendStakingRewards` where the `amount` has already been subtracted from the `fundsForStakingRewards` in `distributeWeeklyAPR` as part of the `rewardsPerLPToken * totalStaked`:

```
function _sendStakingRewards(address to, uint256 amount) internal {
    fundsForStakingRewards -= amount;
    require(vinciToken.transfer(to, amount));
}
```

Recommended mitigation steps

Do not decrease `fundsForStakingRewards` in `_sendStakingRewards`.

H.05. Super staker rewards will be incorrectly sent to users that are not yet super stakers.


Description

In VinciStakingV1. `_crossCheckpoint`, penalty pot rewards are redeemed for the staking periods that have passed by calling the inherited method `_redeemPenaltyPot`, which multiplies the user's stake by the corresponding allocation tracker.

However, the penalty pot rewards are only distributed among super stakers. A user becomes a super staker after at least one checkpoint has been crossed. Therefore, the penalty pot rewards are wrongfully accounted for users who call `_crossCheckpoint` for the first time since their stake is not part of the supply eligible for penalty pot allocations.

Recommended mitigation steps

Do not account for penalty rewards if the user is not yet a super staker:



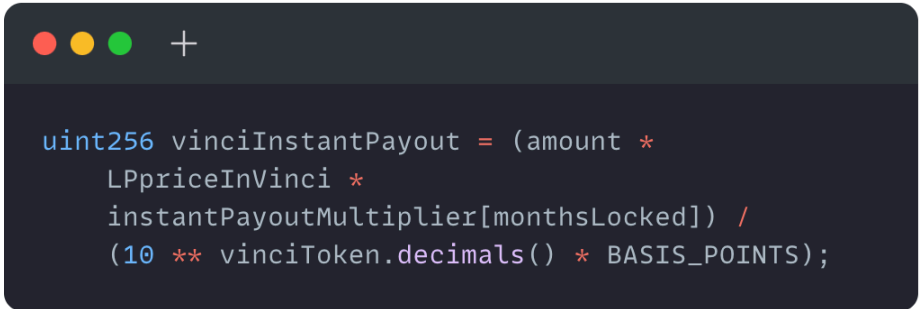
```
uint256 penaltyPotShare = _isSuperstaker(user)
    ? _redeemPenaltyPot(user, activeStake)
    : 0;
```

Medium severity

M.01. Instant Vinci rewards are calculated with the wrong precision.

Description

When users deposit LP tokens to the VinciLPStaking contract, there is an instant reward paid in VINCI, which is a certain percentage of the LP token's value based on the stake duration.



```
uint256 vinciInstantPayout = (amount *
    LPpriceInVinci *
    instantPayoutMultiplier[monthsLocked]) /
    (10 ** vinciToken.decimals() * BASIS_POINTS);
```

The variables' precisions in the above equations are as follows:

$$(1eX * 1e18 * 1e2) / (1e18 * 1e2)$$

X is the number of decimals the deposited LP token has. Therefore, the result of the equation stored in ``vinciInstantPayout`` will not certainly be in ``vinciToken`` decimals, but in the decimals of the LP token.

Fortunately, the contract is intended to be used only with known LP tokens from DEX-es like SushiSwap, where the contract decimals are 18, the same as the VINCI token, meaning that the result will be in the correct precision. However, if another staking token is used, the result might be off by a lot, causing either too low rewards to be paid or too high, depending on the token decimals.

Recommended mitigation steps

Consider using ``10 ** lpToken.decimals()`` instead of ``10 ** viniToken.decimals()``.

M.02. Weekly LP staking rewards are distributed from the wrong starting date.

Description

The weekly rewards are set to account the start of the distribution from ``rewardsReferenceStartingTime`` which is hardcoded to 1685491200 - 31 May 2023 02:00:00 GMT+02:00, the project launch date.

However, if the contract is deployed (or reused) after this date or the first deposit happens after this date, additional staking rewards will be distributed even though there were no stakes for a certain period of time.

Recommended mitigation steps

Consider using the time of the first deposit as the ``rewardsReferenceStartingTime`` instead of the hardcoded 1685491200.

Low severity

L.01. Users can maintain a stake of just 1 wei to keep the super staker status.

Description

The super staker status is given to users who stake an amount of VINCI tokens for at least 6 months and do not unstake all of it. If the user unstakes all their tokens, they lose their super staker status, which means they will no longer be eligible for penalty pot rewards.

The flaw is that users are not required to stake a certain minimum amount of tokens to keep their super staker status. A user can stake 1 wei of VINCI and then wait 6 months until they gain the super staker status, and stake their balance afterward, effectively circumventing the requirement of a 6+ month stake. Users can also decide to never withdraw their full amount of staked VINCI tokens but always unstake `activeStaking - 1` wei to keep their super staker role, rendering the checkpoint reset useless.

Recommendation


Consider implementing a threshold mechanism similar to the one with tier levels to prevent the above scenarios.

L.02. The Vinci token operator has control over the whole project value.

Description

Users should be aware that the deployer of the Vinci token is given the `CONTRACT_OPERATOR_ROLE`, which means a privileged account has access to all minted but currently unvested VINCI tokens.

This can be abused to rug pull the entire project value from DEX-es.



```
// @dev This withdraws VINCI tokens that have not been allocated to vestings (but have been minted)
function withdraw(address recipient, uint256 amount) public onlyRole(CONTRACT_OPERATOR_ROLE) {
    require(amount <= freeSupply, "amount exceeds free supply");
    _transfer(address(this), recipient, amount);
    freeSupply -= amount;
}
```

Recommendation

Make sure users are aware of the above risk. Use a multi-signature wallet for the account given the `CONTRACT_OPERATOR_ROLE`. Even if the above method is removed, the contract operator can still perform the same action by simply creating a vesting with 0 duration and the desired amount to a desired recipient, so the risk cannot be completely mitigated.

L.03. Insufficient input validation for vesting parameters.

Description

When the Vinci token contract operator creates a vesting schedule for a given account, they provide a struct with three properties: `amount`, `releaseTime`, and `claimed` boolean flag.

No input validation is present, which means the operator can pass wrong values leading to unexpected behaviour.

Recommendation

Consider adding the following checks:

```
if (
    vestings[i].amount == 0 ||
    vestings[i].claimed == true ||
    vestings[i].releaseTime < block.timestamp
) revert InvalidVestingParameters();
```

L.04. Wrong accounting for penalty pot rewards.

Description

The following function is used to remove a certain amount of super borrower's stake from the eligible supply for the penalty pot when they call `unstake`:

```
function _removeFromElegibleSupplyForPenaltyPot(uint256 amount) internal {
    uint256 amountToRemove = amount + bufferedDecimalsInSupplyRemovals;
    supplyElegibleForAllocation -= (amountToRemove / PENALTYPOT_ROUNDING_FACTOR);
    // overwriting is intentional, as the old decimals are included in `amountToRemove`
    bufferedDecimalsInSupplyRemovals = amountToRemove % PENALTYPOT_ROUNDING_FACTOR;
}
```

Consider the following example:

- `totalVinciStaked` = 1.0009e18 VINCI (all eligible for penalty pot allocation)
- `supplyElegibleForAllocation` = 1.0009e18 / 1e15 = 1e3
- `bufferedDecimalsInSupplyAdditions` = 0.9e15
- `penaltyPool` = 100e18 VINCI
- `_distributePenaltyPot()` is called
- `allocationPerStakedVinci` = 100e18 / 1e3 = 100e15

Now when the user crosses a checkpoint and claims their rewards from the penalty pot, it will be calculated as $1.0009e18 * 100e15 / 1e15 = 100.09$ VINCI in `_bufferPenaltyPotAllocation`.

However, since the `penaltyPool` was only 100e18 VINCI, the user cannot actually claim their rewards.

Recommendation

A fix is non-trivial with the current implementation.

L.05. An unbounded loop may cause DoS for Vinci token vestings.

Description

Users' vesting is stored in an array of `Timelock` structs. This array is then iterated when users claim their tokens. This can result in a DoS if too vestings are added meaning tokens would be lost.

Recommendation

Consider using a mapping instead of an array.

L.06. The LP staking operator can block staking functionality.

Description

The VinciLPStaking contract operator can set the `LPpriceInVinci` storage variable to any arbitrary value through `setLPPriceInVinci`.

It is used in `newStake` to calculate the instant payout. The operator can block the latter function by setting the `LPpriceInVinci` to `type(uint256).max`.

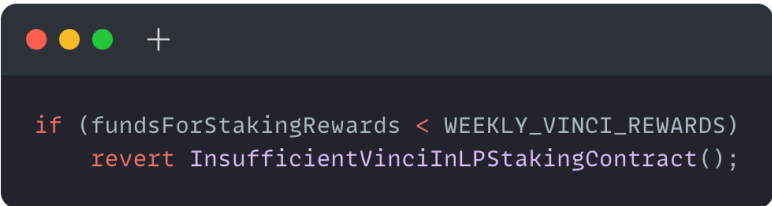
Recommendation

Consider adding an upper bound for `LPpriceInVinci`.

L.07. Wrong value is used for weekly rewards in sanity check.

Description

VinciLPStaking.`distributeWeeklyAPR` performs the following safety check:



```
if (fundsForStakingRewards < WEEKLY_VINCI_REWARDS)
    revert InsufficientVinciInLPStakingContract();
```

If there is any precision loss from previous distributions, the lost decimals will be stored in ``bufferedDecimals``. Then, in the next distribution, ``bufferedDecimals`` will be added to ``WEEKLY_VINCI_REWARDS`` and the actual distributed amount would be ``WEEKLY_VINCI_REWARDS + bufferedDecimals``.

Recommendation

Consider either removing the above check or using ``WEEKLY_VINCI_REWARDS + bufferedDecimals`` instead of only ``WEEKLY_VINCI_REWARDS``.

Informational

I.01. Storage variables can be marked constant or immutable.

The following storage variables could be marked as ``constant`` or ``immutable``:

- VinciLPStaking - ``vinciToken`` (immutable) , ``lpToken`` (immutable), ``rewardsReferenceStartingTime`` (constant)
- VinciStakingV1 - ``vinciToken`` (immutable)

I.02. Missing zero address checks.

An ``address(0)`` check could be added in ``_stake``.

I.03. Public functions can be marked external.

Multiple functions are marked as ``public`` but are not called internally. Consider marking them as ``external``.

I.04. Unstaking 0 amount should not be allowed.

Revert if a user tries to unstake 0 amount in VinciStakingV1.``unstake``.

I.05. Function incorrect as per spec.

In the documentation, it is mentioned that “Any wallet is allowed to fund the contract with Vinci tokens”. However, ``fundContractWithVinciForRewards`` in VinciStakingV1 is only callable by the contract operator.

I.06. Unsafe ERC20 operations.

Consider using the SafeERC20 library from OpenZeppelin for token transfers.

I.07. Thresholds order is not validated.

Thresholds in the TierManager should be ordered in ascending order. Consider adding a check in ``_updateTierThresholds``.

I.08. Typographical mistakes.

Multiple typographical mistakes are made around the codebase - spelling mistakes, unused functions, unused structs, incorrect or forgotten old comments. Consider fixing those issues.