# Htp Stacking Protocol Audit

April 24, 2024

By CoinFabrik

v202311

# Executive Summary

CoinFabrik was asked to audit the contracts for the Htp Stacking project.

During this audit we found one high issue and two medium issues. All issues were fixed by the development team.

# Scope

The audited files are from the git repository located at https://gitlab.com/onpulse/htp-staking/htp-staking-protocol. The audit is based on the commit `2ec302210087d6513f5a0702ed2cbefab434e0ed`. Fixes were checked on commit `669d3de75fadaf550b0d7869dea753b9165e97c5.`

The scope for this audit includes and is limited to the following files:

- `StackingBase.sol`: Base staking contract
- `StakingRewards.sol`: Top staking protocol class
- `IWPLS.sol`: WPLS token interface

No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

# Methodology

CoinFabrik was provided with the source code. Our auditors spent one week auditing the source code provided, which includes understanding the context of use, analyzing the boundaries of the expected behavior of each contract and function, understanding the implementation by the development team (including dependencies beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. Without being limited to them, the audit process included the following analyses.

- Arithmetic errors
- Outdated version of Solidity compiler
- Race conditions
- Reentrancy attacks
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers

- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

After delivering a report with our findings, the development team had the opportunity to comment on every finding and fix the issues they considered convenient. Once fixed and/or commented, our team ran a second review process to verify that the changes to the code effectively solve the issues found and do not unintentionally add new ones. This report includes the final status after the second review.

# Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below. This table does not include the enhancements we suggest to implement, which are described in a specific section after the security issues.

| ID | Title | Severity | Status |
|---|---|---|---|
| HI-01 | Integer Overflow Prevents Upgrade Rewards, Staking | High | Resolved |
| ME-01 | Reward Token Can Grow Indefinitely | Medium | Resolved |
| ME-02 | Unreachable Reward Token ID | Medium | Resolved |

## Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. Blocking bugs are also included in this category. They must be fixed **immediately**.

- **High:** These refer to a vulnerability that, if exploited, could have a substantial impact, but requires a more extensive setup or effort compared to critical issues. These pose a significant risk and **demand immediate attention**.

- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of, but might be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

# Issues Status

An issue detected by this audit has one of the following statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code, but is a result of an intentional decision. The reported risk is accepted by the development team.
- **Resolved**: Adjusted program implementation to eliminate the risk.
- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk.

# Critical Severity Issues

No issues found.

# High Severity Issues

## HI-01 Integer Overflow Prevents Upgrade Rewards, Staking

**Location**:

- `StackingBase.sol:93,111`

**Classification**:

- CWE-190: Integer Overflow or Wraparound[1]

The modifiers `UpdateRewards()` and `NextTick()` traverse the `rewardToken[]` array to recalculate rewards for each token. But if the calculation of a single token reverts due to an Integer Overflow, the function that used the modifiers (`stake()` or `withdraw()`) will be reverted.

[1] https://cwe.mitre.org/data/definitions/190.html

It is possible to cause an Integer Overflow in the modifiers as they call the function `_earned()` that in line 347 it contains this line:

```
        //Get reward rate for the current tick
        earnedTotal += balance * (rate - lastRate) / 1e18;
```

In this code, if `lastRate` is superior to the current `rate`, the `_earned()` function will revert, causing the reversion of the complete transaction.

### Recommendation

Handle all possible integer overflows so they don't cancel the transaction. Specifically check for all subtraction operations like in StackingBase.sol: 343, 347 and 353.

### Status

**Resolved.** A Check was added to _earned() function so no underflow can be caused in any scenario.

## Medium Severity Issues

## ME-01 Reward Token Can Grow Indefinitely

**Location:**

- `StackingBase.sol:130,91,108`

**Classification**:

- CWE-770: Allocation of Resources Without Limits or Throttling[2]

The function `addRewardToken()` adds a token to the `rewardToken[]` array so it can generate earnings for the stakers. But there is no upper limit for the amount of reward tokens that can exist. As this unlimited array is traversed in loops (for loops at lines 91 and 108), the loop may revert due to gas exhaustion, preventing all staking and withdrawing operations.

**Note:** Severity of this issue is diminished because of the low likelihood of the contract Owner adding enough reward tokens to trigger the issue.

### Recommendation

Implement an upper limit to the amount of reward tokens.

[2]https://cwe.mitre.org/data/definitions/770.html

## Status

**Resolved.** Upper limit implemented at `addRewardToken()`.

### ME-02 Unreachable Reward Token ID

**Location:**

- `StackingBase.sol:91,108,144,194,220,236,232,246,258,314`

**Classification**:

- CWE-129: Improper Validation of Array Index[3]

The `rewardToken[]` array storing the reward tokens can grow without limit (see ME-01) but the variable used to index this array is declared as `uint8`, with a limit of 256. The result is that any reward token with an ID over 256 is unreachable, and cannot be removed.

**Note:** Severity of this issue is diminished because of the low likelihood of the Contract Owner adding enough reward tokens to trigger the issue.

#### Recommendation

Implement an upper limit to the amount of reward tokens, that is less than 256, or increase the index variable integer size.

#### Status

**Resolved.** Upper limit implemented at `addRewardToken()`.

## Minor Severity Issues

No issues found.

# Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest to other stakeholders of the project, including users of the audited contracts, token holders or project investors.

## Centralization

No important centralization is detected on the contract, except for necessary contract ownership to manage reward tokens.

[3]https://cwe.mitre.org/data/definitions/129.html

# Upgrades

No upgrade mechanism is implemented in the provided contracts.

# Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

## StackingBase.sol

### Owner

The owner of the contract can add and remove reward tokens.

# Staking tick mechanism

Unlike other staking protocols, Htp Stacking does not rely on block height nor other external mechanism to measure time but it uses ticks, a monotonic increasing variable that is updated every time any user calls `stake()` or `withdraw()`. This means that earnings depend not on time but on contract activity.

# Changelog

- 2024-04-24 – Initial report based on commit `2ec302210087d6513f5a0702ed2cbefab434e0ed.`
- `2024-05-17` – Reaudit report based on the fixes in commit `669d3de75fadaf550b0d7869dea753b9165e97c5.`

**Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the HTP Stacking project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.**