Stake2Care MSFP audit Report

This audit report provides a comprehensive analysis of the Stake2Care MSFP protocol, conducted by HHK, focusing on identifying potential security vulnerabilities, gas optimizations and assessing overall code quality.

Table of Contents

Project Presentation

Stake2Care MSFP

Stake2Care MSFP provides an update to the initial Stake2Care protocol by adding an incentive to users if they lock their `steth` for a certain duration. Donors choosing to lock their tokens will accrue an influx of “MSF-points”, a utility token, which can be used within the Stake2Care ecosystem.

Auditor presentation

HHK is a freelance smart contract developer and security researcher with a strong track record. He Often rank in the top 10 during audit contests and has conducted over 15+ audits with yAudit. Specializing in smart contract security since early 2023, HHK's audit portfolio is available here. For inquiries, please contact hhk.contact@proton.me.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
- CharityEscrow.sol
- MSFPoint.sol
```

The contracts of the Stake2Care MSFP Repo were reviewed over X days. The repository was under active development during the review, but the review was limited to the latest commit at the start of the review. This was commit d32d35aa3609d1ffbaac68de53faff9079946e00 for the Stake2Care MSFP repo.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability. By deploying or using the code, Stake2Care MSFP and users of the contracts agree to use the code at their own risk.

Findings Severity

Impact/Likelyhood	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Informational

Suggestions to enhance performance (gas consumption) and architecture are provided as gas and informational findings.

Note that the assessment of findings severity is subjective and may vary depending on individual perspectives, system configurations, and specific use cases. What one user considers a critical issue may be deemed less severe by another based on different criteria and priorities.

Findings

Critical Findings

None.

High Findings

None.

Medium Findings

None.

Low Findings

1. Low - Users may not be able to decrease their lock if the rate increases

After locking their impact vault tokens, users are able to call `decreaseLock()` to reduce their lock against a points earned earlier with an additional 10% fee. However if the earn structure changed and rate went up, the calculation may result in more points that the user actually have and revert.

Technical Details

In the function `decreaseLock()` users can reduce their lock by any amount as long as they still have their tokens initially received while locking and can afford the 10% penalty fee.

To determine the amount of points the user needs to burn to reduce its lock, the function call the internal function `computePointDecrease()`. This function will calculate the original amount of points received using the current reward rate and then remove the points earned from the time spent as well as the points for the new lock value. Points left will receive a 10% boost and will be burned from the user balance.

However as the natspec points out: `investor may have a net negative point PNL if the earn rates have increased since he locked`. Because the function uses the current rate, if the rate increased since the lock was created, the function will assume that the user has more points than he actually has which will make the function revert.

Impact

Low. Users might not be able to decrease their lock if the rate increased.

Recommendation

Consider saving the original rate each user locked at and use it to calculate the amount of token to burn.

By saving the original rate, you could also add a new function that would allow users to reset their rate while keeping the same lock: burn all MSFP according to the old rate then mint new MSFP using the new rate. Making it more fair for users that locked when rate where lower.

Developer Response

Acknowledged. Done on purpose, it's a design choice: 1- Users may otherwise arbitrage the rate increase by unlocking and then relocking with the higher rate. 2- Note that if the rate decreases, users may unlock at a lower point cost so this symmetry does not make it unfair for the user. The financial parallel is buying a

bond at a fixed rate. If rates increase (resp. decrease), the fair value of the bonds decreases (resp. increases), furthermore part of the disproportionate (quadratic) time reward element is here to compensate the user against the reward rate risk he takes. 3- This breaks the symmetry between users, with different users having different unlock costs at the same time. Importantly, points received at lock time may have already been used to access specific opportunities (like private NFT sales) whose value varies over time - using historical rates would unfairly favor users who accessed high-value opportunities early while paying less to unlock. 4- Points can always be bought from the contract, so users are never formally trapped in.

Gas Saving Findings

None.

Informational Findings

1. Informational - Users may fully unlock instead of reducing their lock

Technical Details

The function `decreaseLock()` can be called by a user to reduce the duration and/or amount of its lock.

It takes the duration and amount decrease as parameter. The function will check these two parameters against the `minimumLockDuration` and `minimumLockAmount` to make sure the new lock is at greater or equal to these values.

The issue is that the check will not revert but instead modify the parameter entered by the user. If the new lock is too small, the function will set the value to 0, remove the lock and unlock the tokens.

So if a user reduces both duration and amount but one of the two is result in an amount to small, it will result in a full unlock instead of a decrease of the lock.

Impact

Informational. User may end up fully unlocking instead of decreasing his lock.

Recommendation

Consider reverting if the new lock value is smaller than the minimum value.

Developer Response

Fixed in cd2daae7796f9f8144d709c2cd73f74762e04eda.

2. Informational - `totalSupply()` may return incorrect value

Technical Details

The function `totalSupply()` uses the `impactVault.balanceOf(address(this))` to get the amount of impact vault token locked on the contract.

However donations to the contract may make this value incorrect as the amount of token on the contract will be greater than the actual balance locked.

Impact

Informational.

Recommendation

Consider adding a storage variable and modify it on lock/unlock.

Developer Response

Acknowledged - not done for gas optimization reasons. It is not worth the extra gas cost.

3. Informational - `increaseAllowance()` and `decreaseAllowance()` are not overridden

Technical Details

The contract `CharityEscrow` overrides main ERC20 functions like `transferFrom()` or `approve()` to disable them.

However it doesn't override the public functions `increaseAllowance()` and `decreaseAllowance()` which means they could be called to modify the allowance of a user. While this doesn't result in any direct issue it seems that it would be better to disable them since `approve()` is disabled.

Impact

Informational. Allowance can be modified.

Recommendation

Disable the 2 functions or disable the internal function `_approve()` that is used by both of them and the public `approve()` function.

Developer Response

Fixed in a9dccae596066c90359bad7da494d26bd8942f58.

Conclusion

The Stake2Care MSFP update will introduce new utility to the project. The codebase is well-structured and thoroughly tested. While minor concerns were raised regarding potential rate changes affecting withdrawals, the team provided a clear explanation of the reasoning behind.