Security Audit Report

QoWatt cards-staking
smart contract

by ARDA

on May 6, 2024

Table of Content

Disclaimer

The report makes no statements or warranties, either expressed or implied, regarding the security of the code, the information herein or its usage. It also cannot be considered as a sufficient assessment regarding the utility, safety and bugfree status of the code, or any other statements.

This report does not constitute legal or investment advice. It is for informational purposes only and is provided on an "as-is" basis. You acknowledge that any use of this report and the information contained herein is at your own risk. The authors of this report shall not be liable to you or any third parties for any acts or omissions undertaken by you or any third parties based on the information contained herein.

Terminology

Inherent risk: A risk for users that comes from a behavior inherent to the smart contract design.

Inherent risks only represent the risks inherent to the smart contract design, which are a subset of all the possible risks. **No inherent risk doesn't mean no risk.** It only means that no risk inherent to the smart contract design has been identified. Other kind of risks could still be present. For example, the issues not fixed incur risks for the users, or the smart contracts deployed as upgradeable also incur risks for the users.

Issue: A behavior unexpected by the users or by the project, or a practice that increases the chances of unexpected behaviors to appear.

Critical issue: An issue intolerable for the users or the project, that must be addressed.

Major issue: An issue undesirable for the users or the project, that we strongly recommend to address.

Medium issue: An issue uncomfortable for the users or the project, that we recommend to address.

Minor issue: An issue imperceptible for the users or the project, that we advise to address for the overall project security.

Audit Summary

Scope of initial audit

- **Repository:** https://github.com/QoWattEcosystem/qowatt-sc
- **Commit:** `eef18d20f118619a39829b835da8144383ce67d4`
- **Path to Smart contract:** `./contracts/staking-cards/`

Scope of final audit

- **Repository:** https://github.com/QoWattEcosystem/qowatt-sc
- **Commit:** `173600ed0aee5fadf8036203eeee2eb044f7ee59`
- **Path to Smart contract:** `./contracts/staking-cards/`

Report objectives

1. Reporting all **inherent risks** of the smart contract.
2. Reporting all **issues** in the smart contract **code**.
3. Reporting all **issues** in the smart contract **test**.
4. Reporting all **issues** in the **other** parts of the smart contract.
5. Proposing **recommendations** to address all issues reported.

3 inherent risks in the final commit

2 issues in the final commit

43 issues reported from the initial commit and 2 remaining in the final commit:

Severity	Reported			Remaining		
	Code	Test	Other	Code	Test	Other
Critical	2	0	0	0	0	0
Major	7	0	0	0	0	0
Medium	13	1	0	2	0	0
Minor	20	0	0	0	0	0

Inherent Risks

R1: Users are not guaranteed to be able to claim boosted rewards.

This is because boosted rewards are in QoWatt coins, and can be claimed only if coins are available in the contract, which is not predictable. Indeed, it depends on whether users mint cards, as it is the only source of coins in the Staking Cards contract. However, even if boosted rewards cannot be claimed in case there are no coins in the contract, users can still use these unclaimed boosted rewards to mint new cards or upgrade their cards.

R2: Users have no guarantee on the rewards which are distributed per block in each pool.

This is because at any time the owner can stop producing rewards, reduce the rewards reserve, change the rewards factor, or change the distribution of rewards between pools.

R3: Users can earn less rewards than they should.

This is because the total number of shares in the Staking Cards can be overestimated, hence the rewards per share (RPS) is underestimated, for the following reasons:

- When a user interacts with the contract, the RPS is updated before removing the shares which have expired in previous epochs. Therefore the RPS increase and the RPS history are computed using an overestimated total number of shares.
- The bonus shares of a user who is inactive contributes to the total number of shares for `14` days, which is the maximum time before a bonus share can

expire, even if the accurate expiry of the user's bonus shares is shorter, e.g. only `2` days.

Code Issues & Recommendations

Since the smart contract code is not open-source, only the remaining issues are published.

C10: Some rewards might never be distributed to users if an entire epoch passes without rewards aggregation

Severity: Medium **Status:** Won't fix

Location

`contracts/staking-cards/src/share.rs`

Description

Current behavior: When rewards are not aggregated for at least an entire epoch `epoch`, a portion of the rewards aggregated at the next user interaction will neither be distributed to users nor reimbursed to the Staking Pool Rewards.

This is because, when QWT rewards are subsequently aggregated, the new rewards per share (RPS) `new_rps` is computed using the previously recorded total supply, which is overestimated by the amount of bonus shares which expired at the said epoch: `bonus_share_expiration_by_epoch(epoch, pool, level)`.

Indeed, these bonus shares expire at the past epoch `epoch`, thus they are ineligible to claim rewards between the RPS for this epoch (`history_rewards_per_share_by_epoch(epoch, pool, level)`) and the new RPS (`new_rps`). Therefore, the following amount will never be distributed to users or be added back to the reserve `rewards_reserve` of the Staking Pool Rewards:

```
(new_rps - history_rewards_per_share_by_epoch(epoch, pool, level))
  * bonus_share_expiration_by_epoch(epoch, pool, level)
```

Consequently, this amount of rewards will forever stay in the Staking Pool Rewards because they are considered distributed (i.e. removed from the storage `rewards_reserve`), but no users will ever be able to claim them

Moreover, the problem exacerbates as more epochs pass without aggregating rewards, as then the amount of expiring bonus shares increases, and so does the amount of rewards that will never be distributed or reimbursed.

Expected behavior: It should be possible to eventually distribute to users all the rewards deposited by the QoWatt team in the Staking Pool Rewards smart contract. Therefore, when some of the aggregated rewards can't be claimed, e.g. due to an overestimation of the total supply, these rewards should be re-added the Staking Pool Rewards so that they can later be distributed.

Worst consequence: 1 month passes without any user interaction, and then rewards are aggregated. Although all users' bonus shares have expired before the middle of the month, they are fully counted in the RPS update. Therefore, more than 50% of the aggregated bonus rewards for the entire month will never be distributed to users.

Recommendation

First, we recommend implementing the solution to Historical RPS of an epoch can be smaller than the real RPS at the beginning of that epoch. It is necessary to ensure that the procedure below properly computes the rewards that should be added back to the reserves.

Then, in `free_expired_share_bonus`, as we remove expired bonus shares from the total supply, we would sum together the amount of rewards that will never be distributed because they were assigned to expired bonus shares.

For each pair `(pool, level)` and past epoch `last_epoch` since `last_epoch_bonus_share_expiration`, this amount can be computed as follows:

```
compute_rewards_for_a_given_position(
  bonus_share_expiration_by_epoch(last_epoch, pool, level).get()
  history_rewards_per_share_by_epoch(last_epoch, pool, level).get(),
  history_rewards_per_share_by_epoch(cur_epoch, pool, level).get(),
);
```

Then, we would call `free_rewards` to add back all these undistributed rewards to the reserve of the Staking Pool Rewards smart contract.

C11: Historical RPS of an epoch can be smaller than the real RPS at the beginning of that epoch

Severity: Medium **Status:** Won't fix

Location

`contracts/staking-cards/src/share.rs`

Description

Current behavior: For a given pair `(pool, level)`, there are two types of rewards per share (RPS) recorded in the smart contract:

- The real RPS.
- For each past epoch, a historical RPS `history_rewards_per_share_by_epoch(epoch, pool, level)`, abbreviated as `history_rps(epoch)` in what follows.

When rewards are aggregated, the real RPS increases, say from `rps_from` to `rps_to`, and the historical RPS are computed for missed epochs: from `last_epoch_rewards_per_share` to the current epoch.

However, if rewards are aggregated after at least one entire epoch `epoch` without aggregation, then the computed historical RPS `history_rps(epoch)` might be *smaller* than `rps_from`, although `epoch` is *higher* than the epoch at which the real RPS `rps_from` was computed.

This is because of the way `write_rewards_per_share_history` computes historical RPS: instead of interpolating between `rps_from` and `rps_to` to fill all the history RPS for missed epochs, it interpolates between `history_rps(last_epoch_rewards_per_share)` and `rps_to`. Therefore, as `history_rps(last_epoch_rewards_per_share)` was computed at the start of the epoch `last_epoch_rewards_per_share` while `rps_from` was computed any later time during the same epoch, `history_rps(last_epoch_rewards_per_share)` is likely strictly smaller than `rps_from`, and thus the newly computed history RPS are smaller than what they should be.

This is problematic because, if two users have the same bonus share expiring at the same epoch, if the 1st user claims daily while the 2nd user claims after his bonus share has expired, then they won't get the same amount of bonus rewards. For the 1st user, rewards will be computed using the real RPS at epoch `epoch`, while for the 2nd user, rewards will be computed using the historical RPS of epoch `epoch + 1`, which can be smaller.

Expected behavior: Historical RPS should be correctly computed, in particular the historical RPS of an epoch should be bigger than any real RPS recorded in previous epochs. This will ensure that users with the same bonus share earn the same amount of bonus rewards.

Example: At the start of epoch `1`, Alice and Bob both have 10 bonus shares, which expire at epoch `2`. The history RPS for epoch `1` is `history_rps(1) = 1`, and the RPS of Alice and Bob is also `1`.

- In the middle of epoch `1`, Alice claims while the real RPS is `rps_from = 3`, so she earns `(3-1) * 10 = 20` bonus rewards.
- Epoch `2` occurs without any rewards aggregation, and rewards are then aggregated at the beginning of epoch `3`, and the new real RPS is `4`. Therefore it is written that the historical RPS for epoch `2` is `history_rps(2) = 1 + (4 - 1)/2 = 2.5`, and we are in the situation where `history_rps(2) < rps_from`.
- At epoch `3`, Alice claims, and earns no bonus rewards because her bonus share expires at epoch `2` and her RPS (`3`) is higher than `history_rps(2) = 2.5`. Then, Bob claims, and since `history_rps(2) = 2.5`, he earns `(2.5 - 1) * 10 = 15` bonus rewards.

Therefore, Alice has earned `5` more bonus rewards as Bob, although they had exactly the same bonus shares.

Recommendation

We suggest correcting the way `write_rewards_per_share_history` computes the historical RPS for missed epochs, i.e. such that it interpolates between the real RPS before aggregating rewards and the real RPS after the aggregation.

Namely, in `write_rewards_per_share_history`, instead of defining the starting RPS `last_rewards_per_share[pool_index][level_index]` as the historical RPS `history_rewards_per_share_by_epoch(last_epoch_rewards_per_share, pool, level)`, we would define it as the real RPS before it was updated, i.e. before `update_qwt_rewards_per_share` was called.

Test Issues & Recommendations

Since the smart contract code is not open-source, only the remaining issues are published.