MultiversX Guild Config

MultiversX smart contract - Security audit by Arda

Repository: *https://github.com/multiversx/sc-guilds-rs/*
Smart contract path: *guild-config*
Initial commit: *7a394909b25514c329e19ce9b36d12c634cec464*
Final commit: *5583fe6e2bf12fdbf74170cfd9d05f8d0fd96c43*

Issues

C-R1.1. Solved Major **The following specification is not implemented: *"The APRs and Tiers could be changed in the future. We should have an option to do this."***

Indeed, the APRs can be changed, but not the tiers' criteria, both for users and the guild master, since `add_user_tiers` and `add_guild_master_tiers` fail when there are already tiers set.

Solution: We suggest removing the check that there are no tiers already set in `add_user_tiers` and `add_guild_master_tiers`. However, changes of tiers should not impact non-accumulated rewards for past blocks in guilds. To avoid such negative impact, we proceed as follows:

- In each guild, we store an internal version of user's tiers and guild master's tiers, and this storage is used when aggregating rewards.
- After rewards are aggregated, the internal tiers are updated using the tiers from the Guild Config.

C-R1.2. Solved Major **When attempting to change the APR of the i-th tier, then it is the APR of the (i-1)-th tier which is changed, in particular the APR of the 1st tier can't be changed.**

This is because in `set_apr`:

1. An iteration is made to find the tier's index to change, indices ranging from 0 to the length of the mapper minus 1:

```
None
for (i, tier) in mapper.iter().enumerate() {
   if tier.is_equal(&reward_tier) {
      opt_found_index = Some(i);
      break;
```

```
    }
}
```

2. Then, the mapper's entry at index `opt_found_index` is changed. However the mapper is a VecMapper, hence its indices range from 1 to the length of the mapper. Therefore it is the wrong entry which is being updated.

Solution: We suggest correcting the index mismatch in `set_apr`. We also suggest having a small unit test witnessing that the change of APR works well.

C-R1.3. Solved Major **Users have no protection against increases of the unbond period** `min_unbond_epochs_user`. Similarly for the guild-master. This is because the unbond period can be changed at any time by the owner, and the change takes place immediately. In particular, a significant increase might dissatisfy users, i.e. some of them would have preferred not staking at all given the new unbond period.

Solution: Since the unbond period is not supposed to change, we can remove the ability to change their values.
(Old: We suggest introducing a timelock that will ensure that a duration `UNBOND_PERIOD_CHANGE_TIMELOCK` of 1 day will pass before a change of unbond epoch takes place.

- *So, in the endpoint `set_min_unbond_epochs_user` to change the unbond period, we store the future value of the parameter and the timestamp at which the change can take effect, which is `current_timestamp + UNBOND_PERIOD_CHANGE_TIMELOCK`, in a new storage `future_unbond_period`.*
- *Whenever we need to read the unbond period, we use a method `update_and_get_unbond_period` that updates the value of the unbond period by checking if the timestamp at which the change can take effect is in the past, then setting the unbond period to the stored future value, then clearing `future_unbond_period`, and finally returning the unbond period being stored in `min_unbond_epochs_user`. Note that we can make `update_and_get_unbond_period` into a public endpoint that anyone can call.)*

The same approach can be used for the guild master unbond period.

C-R1.4. Solved Medium If there is a change of `total_staking_token_minted`, or of the user's APRs, or of the guild master's APR, then this will modify all rewards for past blocks which have not yet been aggregated in all the guilds. For example, if `total_staking_token_minted` is increased, this might reduce the user tier for all guilds, reducing the APR of all users for past blocks.

This is because, when rewards will be aggregated in each individual guild for past blocks, the new value of `total_staking_token_minted` instead of the old one.

Solution: We propose a solution so that a change of `total_staking_token_minted` does not impact past rewards:

- In each guild, we store an internal version of `total_staking_token_minted`, and this storage is used to aggregate rewards.
- After rewards are aggregated, the guild's storage `total_staking_token_minted` is updated using the storage from the guild config.

We can do the exact same thing for APRs, i.e. in each guild we store an internal version of the user's APR and of the guild master's APR, and proceed as above to aggregate rewards and then update these internal storages.

C-R1.5. Solved Medium The APR of a higher tier might be smaller than the APR of a lower tier, because there is no enforcement that APRs are increasing in `set_guild_master_tiers_common`, `set_user_tiers_common`, `set_user_tier_apr` and `set_guild_master_tier_apr`.

Solution: We recommend checking that APRs are increasing in `set_user_tier_apr` and `set_guild_master_tier_apr`. In practice, if one APR is changed, we should check that it is bigger than the APR of the previous tier, and smaller than the APR of the next tier.

C-R1.6. Solved Medium An arbitrary number of tiers (for users or for the guild master) can be added. However if there are too many, this will make iterations fail when computing rewards, making all users' interactions fail.

Solution: When setting tiers, we suggest checking that the number of tiers is smaller than a fixed number, e.g. 5.

C-R1.7. Solved Medium If the total amount staked in guilds exceeds `total_staking_token_minted`, e.g. because a mistake was made when setting `total_staking_token_minted`, then rewards aggregation will fail in guilds when trying to get the users' tier, since the percentage staked will exceed 100%, and so no tier will be found. This in turn prevents users from claiming rewards and withdrawing their funds.

Solution: When computing the users' tier, in case the total staked amount exceeds the threshold of all tiers except the highest tier, then we suggest returning the highest tier. A similar approach can be followed for computing the guild master's tier for consistency.

C-R1.8. Solved Medium There is no endpoint to change `max_staked_tokens`. However, the team wishes to have the ability to change this storage, depending on the feedback they get from the community after launching the guilds.

Solution: We suggest adding an owner endpoint to change `max_staked_tokens`. Moreover, in order to not get into issue C-R1.8. if the new value of `max_staked_tokens` gets below the amount currently staked in a guild, we also suggest following the recommendation to issue C-R1.8.

Finally, the value of `max_staked_tokens` should be consistent with ongoing tiers, i.e. bigger than the max threshold of all tiers. Since the last tier has no threshold (see recommendation to issue C-R1.8.), it is enough to check that the threshold of the penultimate tier is smaller than the new value of `max_staked_tokens`.

C-R1.9. Solved Minor It is checked that the criterion to enter a tier is below or equal to the criterion to enter the next tier, however the equality case makes no sense since if two consecutive tiers have the same criterion, then the 1st tier will be obsolete, i.e. never used.

Solution: In `add_tier`, we suggest checking that tiers' criteria are strictly increasing, not just non-decreasing.

Disclaimer