MultiversX Guild

MultiversX smart contract - Security audit by Arda

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

Issues

The issues below have been raised over several 5 consecutive reviews, and appear from the most recent review (R5) to the oldest (R1).

A-R5.1. Not Solved Medium If the maximum number of guilds is reached, a guild can be deployed but it can be activated, in the sense that it can't receive rewards. However, it is possible that users have staked some tokens. In this case, even if the guild is not yet active, users would need to wait for the unbonding period before they can withdraw.

Normal users also have the option to migrate to another guild which is active, but the guild master is forced to wait the 20 days unbonding period.

A-R4.1. Solved Medium Users can stake in guilds where the unbond token is not yet issued. This would prevent them from withdrawing their funds. Indeed, once a guild is deployed, it is possible to immediately stake in it (as there is no state mechanism), even if the unbond token has not been issued.

Solution: In the endpoint `stake_farm_endpoint`, we suggest verifying that the unbond token has been issued.

A-R4.2. Solved Medium In the endpoint `upgrade`, the method `update_all` is called. This is problematic because this method updates storages that impact rewards, e.g. the rewards per block and users' tiers, but does so without first aggregating rewards. Therefore, changes would impact past rewards, possibly making users lose past rewards.

Solution: In the endpoint `upgrade`, we suggest not calling the method `update_all`.

A-R4.3. Solved Minor The field `_unused_contract_state` of the struct `StorageCache` is never used.

Solution: We recommend deleting the field `_unused_contract_state`.

A-R3.1. Solved Major The constraint that a user must call `unbond_farm` with all his unbond tokens is unnecessary and has the following consequence: if a user has burned some of his unbond tokens, even 1 unit, **then he can't ever withdraw his UTK**. Indeed, `unbond_farm` and `cancel_unbond` can be called only with the full amount of unbond tokens.

Solution: We suggest removing the constraint in `unbond_farm` and `cancel_unbond` that the user must send all his unbond tokens.

A-R3.2. Solved Minor The function `withdraw_rewards_common` is useless and unnecessarily complex (~10 lines of code). Indeed, it is only used in the following 2 lines of `close_guild`

```
None
let remaining_rewards = rewards_capacity - accumulated_rewards;
self.withdraw_rewards_common(&remaining_rewards);
```

And these 2 lines could simply be replaced by:

```
None
self.reward_capacity().set(accumulated_rewards);
```

Solution: We suggest performing the above simplification in `close_guild` and deleting the function `withdraw_rewards_common`.

A-R3.3. Solved Minor The storage name "guildMaster" is a prefix of other storage names: "guildMasterTiers", "guildMasterTokens" and "guildMasterRps". This is an error-prone practice, see MultiversX doc.

Solution: We suggest renaming "guildMaster" e.g. into "guildMasterAddress".

A-R2.1. Solved Critical **Due to wrong calculations of users' rewards, an attacker can drain all the UTK of all guilds, in particular preventing other users from withdrawing.**

This is because, when a user claims or compounds rewards for several farm tokens, the rewards per share (RPS) is only updated to the current RPS for the first farm token, and then is averaged with all other RPS. What should be done instead is to define the RPS of the merged position as the current RPS.

Therefore the merged farm token has a too low RPS, i.e. it can be used to claim rewards again which should not be claimable by this farm position. Thus the user would earn more UTK than it should, and prevent others from claiming and withdrawing their farm tokens.

The consequence of the above is that an attacker can claim rewards by providing 1 atomic unit of farm token as 1st payment, and the rest of his farm tokens as 2nd payment. After claiming his rewards, his merged farm token would have an unchanged RPS (or increased by +1). The attacker can then repeat this as many times as he wants to steal all rewards from other users. He can also perform the attack over multiple guilds.

Solution: When a user claims or compounds rewards, we suggest defining the RPS of the new farm token as the current RPS.

A-R2.2. Solved Critical **An attacker can make his share grow arbitrarily big, e.g. worth 99% of all the staked UTK, hence making all users lose their UTK rewards.**

This is because, when a user cancel an unbonding, the whole farm amount (which includes compounded rewards) is added back to storages which are supposed to count only base stake (no compounded stake), i.e. the user base staked tokens and the total amount of base staked tokens (`total_staked_tokens`).

Therefore, by repeatedly calling `unstake_farm` and `cancel_unbond`, the user would make his share of the total staked UTK grow arbitrarily big.

Example: Initially, there are 100 UTK staked in the guild, and a user Alice has 10 UTK staked: 5 base stake, 5 compounded stake.
Step 1: Alice calls `unstake_farm` and `cancel_unbond`: Storages now indicate that there are 105 UTK staked in the guild, and that Alice has 15 UTK staked: 10 base stake, 5 compounded stake.
Step 2: Alice calls `unstake_farm` and `cancel_unbond`: Storages now indicate that there are 110 UTK staked in the guild, and that Alice has 20 UTK staked: 15 base stake, 5 compounded stake.
…
Step 2000: Alice calls `unstake_farm` and `cancel_unbond`: Storages now indicate that there are 10100 UTK staked in the guild, and that Alice has 10010 UTK staked: 10005 base stake, 5 compounded stake.
Therefore Alice started with 10% of the UTK staked and ended up with 99.1% of the UTK staked. In turn, other users will earn 10x less rewards than they should.

Solution: This issue has arisen as a result of complex calculations for distinguishing between the base and compounded stake of a user. We suggest following the recommendation to issue A-R1.3 to resolve it.

A-R2.3. Solved Critical **Users will receive unexpectedly low rewards: the guild master will earn close to 0 rewards while other users will receive a bit less than they should.** This is because in `generate_aggregated_rewards`, when increasing the guild's master rewards per share (RPS) and the users' RPS, in both case the total number of shares is being overestimated: it is `storage_cache.farm_token_supply`, the sum of the guild's master staked tokens and of the users' staked tokens.

Solution: We suggest correcting the total number of shares used for increasing the RPS in `generate_aggregated_rewards`. That is:

- For the guild's master RPS, it should be the UTK staked by the guild's master (base and compounded).
- For the users' RPS, it should be the UTK staked (base and compounded) by all users except the guild's master.

A-R2.4. Solved Major `calculate_rewards_for_given_position` **calculates wrong rewards per share (RPS) for the guild master, hence the front-end will display wrong rewards.**

This is because the function calls `calculate_rewards` which computes rewards using the users' RPS instead of the guild master's RPS.

Solution: We suggest adding a `user` argument to `calculate_rewards_for_given_position`, which would then be forwarded to `calculate_rewards`, so that rewards can be calculated correctly for all users, in particular for the guild master.

A-R2.5. Solved Major In `unstake_farm_common_no_unbond_token_mint`, the calculations of base and compounded UTK from a farm position can have rounding errors. Indeed, the compounded rewards are computed by a rule of three on the farm token's attributes, which performs an integer division, hence the compounded rewards can be underestimated, and in turn the base farm amount can be overestimated:

```
None
let original_attributes = exit_result.original_token_attributes.clone();
let base_tokens_removed =
  original_attributes.current_farm_amount -
original_attributes.compounded_reward;
self.remove_total_staked_tokens(&base_tokens_removed);
self.remove_tokens(
  original_caller,
  TotalTokens::new(base_tokens_removed,o riginal_attributes.compounded_reward),
);
```

This means that too many tokens are removed from the total base staked UTK (`total_staked_tokens`) and the user's base staked UTK. This might lead to failure if these storages become negative, preventing users from withdrawing their UTK.
Also this might prevent the user from withdrawing all his tokens, thus in particular he would have to keep more than 100k UTK staked.

Solution: This issue has arisen as a result of complex calculations for distinguishing between the base and compounded stake of a user. We suggest following the recommendation to issue A-R1.3 to resolve it.

A-R2.6. Solved Major Guilds won't work, i.e. it will be impossible to stake in guilds. This is because the 1st deposit made by the guild master will fail as `mint_per_block_rewards` attempts to read an empty storage `guild_master_tokens`.

Solution: We suggest implementing a default decoding for empty struct `TotalTokens`, which has 0 fields.

A-R2.7. Solved Medium There is no check that the guilds are not globally paused when a user merges farm tokens or cancels his unstake.

Solution: We suggest calling `require_not_globally_paused` in the endpoints for merging and canceling an unstake.

A-R2.8. Solved Medium In `generate_aggregated_rewards`, the updates made by `update_all` (updates of internal rewards per block, seconds per block, internal tiers and internal UTK supply) are not done when there are no UTK staked (`storage_cache.farm_token_supply == 0`) or when there are not enough rewards to be distributed (`total_reward > remaining_rewards`).

Therefore, in such cases, this information would be outdated. For example, if a new guild is deployed, and the rewards per block (RPB) is globally decreased, and a user later first stakes in the guild, he will enjoy the outdated bigger RPB until the next interaction in the guild.

Solution: In `generate_aggregated_rewards`, we recommend calling `update_all` even when there are no UTK staked (`storage_cache.farm_token_supply == 0`) or when there are not enough rewards to be distributed (`total_reward > remaining_rewards`).

A-R2.9. Solved Minor The function `id_to_human_readable` to convert a number into its string representation is unnecessary and complex. Indeed, MultiversX Rust framework already provides the helper function `sc_format` to make such conversions.

Solution: We suggest deleting the function `id_to_human_readable` and instead using the helper function `sc_format` from the MultiversX Rust framework.

A-R2.10. Solved Minor In `base_farm_init`, there is unnecessarily complex logic to define the permissions, including an obsolete mention to backward compatibility:

```
None
let caller = self.blockchain().get_caller();
if admins.is_empty() {
  // backwards compatibility
  let all_permissions = Permissions::OWNER | Permissions::ADMIN |
Permissions::PAUSE;
  self.add_permissions(caller, all_permissions);
} else {
  self.add_permissions(caller, Permissions::OWNER | Permissions::PAUSE);
  self.add_permissions_for_all(admins, Permissions::ADMIN);
};
```

Solution: In the context of the guild, `admins` is never empty and the caller is the owner, so we suggest simplifying the above logic to define the permissions as follows:

```
None
let caller = self.blockchain().get_caller();
self.add_permissions(caller, Permissions::OWNER | Permissions::PAUSE);
self.add_permissions_for_all(admins, Permissions::ADMIN);
```

A-R2.11. Solved Minor In `withdraw_rewards_common` and `close_guild`, it is unnecessary to load a storage cache and call `generate_aggregated_rewards`. Indeed, in both cases, the rewards aggregation was already made before in `multi_unstake` (called within `close_guild`).

Solution: In `withdraw_rewards_common` and `close_guild`, we recommend not loading the storage cache and not calling `generate_aggregated_rewards`.

A-R2.12. Solved Minor The storage name `total_staked_tokens` is used for base staked tokens only, so it is confusing.

Solution: We suggest following the solution to A-R2.3. Otherwise, we can rename the storage e.g. into `total_base_staked_tokens`.

A-R1.1. Solved Critical **An attacker can steal all the staked UTK from all guilds by exploiting a weakness of the endpoint `merge_farm_tokens_endpoint`.**

Namely, this merge endpoint does not explicitly check that the received tokens are farm tokens, and here are the constraints which are implicitly enforced:

- The 1st token should be a farm token, as in `merge_from_payments_and_burn` it is explicit that a farm token of the given nonce should be burnt,
- For all the nonces provided by the caller, in `get_attributes_as_part_of_fixed_supply`, it is required that the attributes of the farm token with the given nonce can be read. Thus it is required that these nonces are present in the Guild, but there is no constraint on the amount.

Therefore, an attacker could proceed as follows:

- As 1st payment, he provides an atomic unit of a genuine farm token, say with nonce 10.
- As additional payment, he provides a fake farm token with nonce 10. The amount is huge, i.e. equal to the whole farm supply in the Guild.

He would then receive a merged position whose amount would be the total farm supply, hence he can unstake all the UTK from the Guild, and repeat this operation in other guilds.

Solution: In `merge_farm_tokens_endpoint`, we suggest explicitly checking that all payments are farm tokens.

A-R1.2. Solved Critical **Once a guild is closed, users can't simply withdraw their positions, instead they are forced to migrate their positions to another guild. In particular, users might be unable to ever withdraw their funds, as it may be impossible to stake funds in other guilds.** This would typically happen if there are no active guilds left, or if all active guilds are full.

Solution: Users should be able to unstake and withdraw their funds normally even after a guild is closed. So we suggest letting users call the endpoints for unstaking and unbonding even after a guild is closed.

A-R1.3. Solved Critical When a user compounds rewards, the new amount of farm tokens is bigger as it comprises the compounded UTK. This increase is not recorded in the user's total base stake as it is marked as user's compounded stake, however when the user will unstake the position, this bigger farm amount will be removed from the user's total base stake. In summary, a bigger amount is removed from the user's base stake than what was initially added. This error has the following consequences:

- **All UTK rewards compounded by users can't be withdrawn.** Indeed, as soon as the recorded user's base stake will become negative due to the calculation error, the unstake transactions will fail. Thus if the initial base stake of the user is 10 and he compounds 2 UTK, then he can only withdraw 10 UTK, the remaining 2 UTK can't be withdrawn.
- **A guild master can close his guild without withdrawing all his stake.** This is because he can exploit the calculation error to withdraw less than his total stake while still withdrawing the amounts of base stake recorded in the Guild.
- **The Guild APR can become too small.** As when users unstake all their positions, a bigger amount can be removed from the total Guild's stake than what they deposited. Since the user APR depends on how much UTK is staked over all in Guilds, and that this total staked amount would be underestimated, this would reduce the users' APR.

Solution: We propose a solution which (i) takes into account other related issues and (ii) takes good care of rounding errors, because tiny rounding errors can have a big impact on the constraint that when users unstake, they should either leave **0** tokens in the contract or more than the minimal staked amount.

The solution is as follows:

1. **For each user, we just record his total stake: sum of base and compounded stake.**

Indeed, on a user basis, we don't need to make any distinction between both. We do the same for the guild master. So when a user stakes, unstakes or compounds, we can simply update his total stake based on the new farm amount.

2. **For the whole Guild, we just keep `farm_token_supply` (sum of base and compounded stake) and record the total base stake: `total_base_staked_tokens`.**

In particular, we can remove `total_compounded_tokens and total_staked_tokens`.
When users unstake positions, we decrease `total_base_staked_tokens` by: the user farm amount minus the amount of rewards compounded in the position, which can be obtained from the attributes.
For safety, **we never decrease `total_base_staked_tokens` below 0, i.e. we cap the decrease by the current value of `total_base_staked_tokens`.**

Finally, we suggest adding a unit test where, after a user enters, and later compounds some rewards, he can exit all his position and his total stake is back to 0.

A-R1.4. Solved Critical **The APRs for users and guild master are not following the specifications, in particular users and the guild master have the same APR.** This is because, even though rewards are aggregated using the right tiers' rules, at claim time, there is no difference between the rewards computation for users and for the guild master. The final APR they receive is a weighted average between the expected guild master's APR and the users' APR.

Solution: We suggest having different rewards computations for the guild master and for users. For this, we can introduce a new rewards per share (RPS) for the guild master, `guild_master_reward_per_share`, and when the guild master claims, we use this RPS instead of the users' RPS. Moreover, when aggregating rewards, we increase the users' RPS and the guild master's RPS separately, based on the respective tiers' rules.

A-R1.5. Solved Major **The APR for the guild master is strongly underestimated**, because the guild master's tier is computed by using the amount of UTK staked by the guild master, although from the specifications it should be computed using the amount of UTK staked by all users in the guild.

Solution: We suggest, as in the specifications, computing the guild master's tier and APR based on the total UTK staked in the guild.

A-R1.6. Solved Major **The guild master can't directly compound his rewards.** This is because in `compound_rewards`, the storage `user_tokens(guild_master)` is being read although it is empty, instead of the storage `guild_master_tokens(guild_master)`, and this will fail.

```
None
self.user_tokens(&caller).update(|tokens_per_tier| {
   tokens_per_tier.compounded += &compound_result.compounded_rewards
});
```

Therefore, he would have to claim his rewards and stake them. This might even be impossible in case the maximum amount of UTK staked in a guild has been reached.

Solution: We suggest updating the guild master's storage `guild_master_tokens` instead of `user_tokens`.

A-R1.7. Solved Major When the guild master closes his guild, all his unstaked tokens are not removed from the total UTK staked over all guilds: `total_staking_token_staked`. Therefore, `total_staking_token_staked` will be overestimated, and **the tier for users' APR can be overestimated, making users earn more rewards than they should.**

Solution: In `close_guild`, we suggest calling `decrease_staked_tokens`.

A-R1.8. Solved Medium Users with non-zero stake should have over 100 UTK staked, however a user with 99 base UTK staked and 2 compounded UTK staked will not be accepted, because `require_over_min_stake` only counts the base stake. However, after discussing with the project team, the compounded stake should also be counted.

Solution: We suggest, in `require_over_min_stake`, to also count the user's compounded tokens.

A-R1.9. Solved Medium **A guild can't be closed if it has no rewards remaining**, because in `close_guild`, when a guild is closed there is an ESDT transfer made for the remaining rewards, which would fail if there are 0 rewards to transfer.

Solution: In `close_guild`, we suggest not making a ESDT transfer of pending rewards if there are 0 rewards left.

A-R1.10. Solved Medium In `stake_farm_common`, a specific logic should be performed if the user staking UTK (`original_caller`) is different from the guild master, but instead the logic is performed for the caller (`caller`):

```
None
if caller != guild_master {
  require!(
    !self.guild_master_tokens().is_empty(),
    "Guild master must stake first"
  );
}
```

Solution: We suggest performing the logic for the user staking UTK (`original_caller`) instead of `caller`.

A-R1.11. Solved Medium In `stake_farm_common`, a wrong event is emitted, namely in case a user is migrating a position from another guild, then the event will indicate that it is the guild factory who is staking tokens, instead of the user. This is because the event indicates that the user who is staking is `caller` instead of `original_caller`:

```
None
self.emit_enter_farm_event(
  &caller,
  enter_result.context.farming_token_payment,
  enter_result.new_farm_token,
  enter_result.created_with_merge,
  enter_result.storage_cache,
);
```

Solution: We suggest emitting the event with `original_caller` instead of `caller`.

A-R1.12. Solved Medium It is desired by the project to have endpoints for withdrawing, claiming and compounding multiple positions, but this is not possible currently: endpoints can perform such operations for at most one position at a time.

Solution: We suggest making it possible to handle several positions in endpoints for unstaking, unbonding, claiming and compounding.

A-R1.13. Solved Medium All rewards sent to `top_up_rewards` after a guild is closed are lost, because there is no way to retrieve them in the Guild Factory.

Solution: In `top_up_rewards` we suggest verifying that the guild is not closed.

A-R1.14. Solved Medium Rewards are not aggregated in `top_up_rewards`. Therefore, the deposited rewards will contribute to past blocks, even if it is intended that they would contribute to future blocks only.

Solution: In `top_up_rewards` we suggest first aggregating rewards, before increasing the rewards capacity.

A-R1.15. Solved Medium All user endpoints except the exit endpoints (migration, unbond) are supposed to fail once a guild is paused. However, `merge_farm_tokens_endpoint` is an unintended exception to the rule.

Solution: We suggest making `merge_farm_tokens_endpoint` fail if the guild is closed.

A-R1.16. Solved Medium The owner endpoint `withdraw_rewards` is payable (although it should not) and unused (since the Guild Factory has no endpoint to call it).

Solution: We suggest removing the endpoint `withdraw_rewards`.

A-R1.17. Solved Medium `BLOCKS_IN_YEAR` is a hardcoded constant but the number of blocks in a year will change in the future, when block duration will change. This will therefore result in a wrong conversion of APR into rewards per block in `bound_amount_by_apr`.

Solution: We suggest introducing a storage `blocks_in_year` in the Guild Config, that only the Guild Factory can change, and to read this storage in guilds whenever needed. However, to avoid changes of `blocks_in_year` from impacting non-accumulated rewards for past blocks, we suggest proceeding as follows:

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

A-R1.18. Solved Medium The file *guild-sc/src/farm_base_impl/base_traits_impl.rs* contains mostly redundant or unused functions.

- The unused functions are `get_exit_penalty`, `apply_penalty`, `mint_rewards`,
- Other functions, apart from `calculate_rewards`, are redundant, i.e. the correct implementations are in *guild-sc/src/base_impl_wrapper.rs.* It is moreover introducing a confusion as we have twice functions with the same name.

Solution: We suggest removing all the file *guild-sc/src/farm_base_impl/base_traits_impl.rs.*

A-R1.19. Solved Medium The backend would need an easy way to obtain the total amount of staked tokens for each user, but right now there is no view function for it.

Solution: We recommend introducing a view function that returns the user's total amount of staked tokens, internally handling whether the user is the guild master or not.

A-R1.20. Solved Minor The struct `FarmTokenAttributes` is useless, as it is always converted to another struct for attributes of farm tokens and unbond tokens. Moreover, the associated helper functions `from` and `into` are useless.

Solution: We suggest removing the struct `FarmTokenAttributes` and associated helper functions.

A-R1.21. Solved Minor The struct `RewardPair` is useless, since there are no boosted rewards involved in guilds.

Solution: We suggest removing the struct `RewardPair`.

A-R1.22. Solved Minor In `unstake_farm_common_no_unbond_token_mint`, there is a duplication of the `into_part` logic to extract the attributes of the exit farm tokens from the attributes of the full amount of farm tokens. Indeed, it is performed once inside the helper function `exit_farm_base`, and once after that in `unstake_farm_common_no_unbond_token_mint`. It is unnecessary to do this twice and as the attributes' computation is critical, making it as simple as possible will remove ambiguity and reduce the likelihood that an issue is introduced in future changes.

Solution: We suggest performing the `into_part` logic only once, e.g. in the helper function `exit_farm_base` and make this function return the attributes so as to not compute them again.

A-R1.23. Solved Minor The imported module `SCWhitelistModule` is unnecessary because the only whitelisted caller we need is the Guild Factory, when it calls the endpoint `stake_farm_endpoint`. Moreover, it increases the number of privileged addresses mixed together: whitelisted addresses, admins, guild master.

Solution: We recommend deleting the module `SCWhitelistModule`, and instead in `stake_farm_endpoint`, if the caller is not the user for which the staking is made, we check that the caller is the owner (i.e. the Guild Factory).

A-R1.24. Solved Minor There is some unused logic around migration nonce, total farm and external claim in *guild-sc/src/config.rs*:

- `DEFAULT_NFT_DEPOSIT_MAX_LEN`
- `DEFAULT_FARM_POSITION_MIGRATION_NONCE`
- `UserTotalFarmPosition`

Solution: We suggest removing the unused logic.

A-R1.25. Solved Minor Unused functions in *tokens/farm_token.rs*: `burn_farm_tokens_from_payments`, `mint_farm_tokens`, `get_farm_token_attributes`, `burn_farm_token_payment`.

Solution: We suggest deleting these methods.

A-R1.26. Solved Minor The following check in `withdraw_rewards_common`:

```
None
require!(
   &rewards_capacity >= withdraw_amount,
   "Not enough rewards to withdraw"
);
```

is superfluous as it comes just after a strictly stronger check:

```
None
let remaining_rewards = &rewards_capacity - &accumulated_rewards;
require!(
   &remaining_rewards >= withdraw_amount,
   "Withdraw amount is higher than the remaining uncollected rewards!"
);
```

Solution: We suggest removing the useless check from `withdraw_rewards_common`.

A-R1.27. Solved Minor It is unnecessary complex to have a separate file *farm_base_impl/base_farm_init.rs* for `base_farm_init`.

Solution: We suggest removing the file *farm_base_impl/base_farm_init.rs* and moving `base_farm_init` in *lib.rs*.

A-R1.28. Solved Minor The function `find_any_user_tier_apr` is unused.

Solution: We suggest deleting `find_any_user_tier_apr`.

A-R1.29. Solved Minor The function `claim_rewards_base` is unused.

Solution: We suggest deleting `claim_rewards_base`.

A-R1.30. Solved Minor The function `get_initial_farming_tokens` is unused.

Solution: If not used in any changes made for other issues, we recommend deleting `get_initial_farming_tokens`.

A-R1.31. Solved Minor In `claim_rewards`, the variable name `virtual_farm_token` is confusing, since the token is not really virtual, indeed it is actually minted.

Solution: We suggest using a more intuitive name like `new_farm_token`.

A-R1.32. Solved Minor In `compound_rewards`, the naming `tokens_per_tier` in the following lines is confusing:

```
None
self.user_tokens(&caller).update(|tokens_per_tier| {
  tokens_per_tier.compounded += &compound_result.compounded_rewards
});
```

Namely, this variable represents the total base stake of users, so the "per_tier" makes no real sense here.

Solution: We suggest renaming `tokens_per_tier` in a more natural way, e.g. `tokens` or `staked_tokens`.

Inherent Risks

1. Users have no guarantee that they will get profits according to the announced APRs, because the UTK supply is limited. Namely, if there are many users staking in the guild, there will not be enough rewards available to maintain the announced APRs.

Disclaimer