MultiversX xExchange Fees Collector (2)

MultiversX smart contract - Security audit by Arda

Repository: *https://github.com/multiversx/mx-exchange-sc/*
Smart contract path: *energy-integration/fees-collector*
Initial commit: *dbd9093bd1d9dcd8a85a7ef88fe283b6636fafe8*
Final commit: *36e8e8dfc4101ccf9e855c0a8d44971fea1b2c31*

Issues

1. Available tokens highly overestimated in get_token_available_amount and prevent some users from claiming

Status	Solved
Severity	Critical
Commit (if not initial)	
Location file (optional)	
Additional note (optional)	

Description

The method `get_token_available_amount` is supposed to compute the amount of tokens in the contract which are not allocated as rewards for past 4 weeks as well as the current week: `current_week - 4, current_week - 3, …, current_week`.

For this, it subtracts the amounts allocated for these weeks to the balance of the smart contract. However, it wrongly computes the amount allocated for these weeks. Indeed, it considers that the amount allocated to a week is in the storage `accumulated_fees(week)`, although it should be the sum of `accumulated_fees(week)` and the amount of tokens in the array `total_rewards_for_week(week)`.

Namely, each time a week passes, at the 1st interaction in the next week (`week + 1`), the method `collect_and_get_rewards_for_week` clears `accumulated_fees(week)` and adds the reward token to an array `total_rewards_for_week(week)`. Therefore, the allocated

rewards for that week are in the array `total_rewards_for_week(week)`, not in `accumulated_fees(week)` anymore. So at all times, the amount of reward token allocated for a week is the sum of `accumulated_fees(week)` and of the amount of that token in `total_rewards_for_week(week)`.

Consequence: In practice, there is at least 1 user interaction per week with the Fees Collector, therefore rewards allocated for weeks `current_week - 4, …, current_week - 1` would be in the array `total_rewards_for_week`, not in `accumulated_fees`. In turn, the method, `get_token_available_amount` does not subtract all allocated rewards for these last 4 weeks, and thus it overestimates the available rewards.

The endpoints `swap_token_to_base_token` and `redistribute_rewards` which call this method would therefore allocate too many rewards to the current week.

This would make **users claim too many rewards, thereby preventing some other users to claim their rewards** once the smart contract balance will be empty. This would be **an irreversible problem since rewards would have been claimed already** (hence withdrawn from the contract), and **this issue would materialize on mainnet as soon as the contract is upgraded**.

Here is a unit test that reproduces the issue.

Unit test issue_available_amount_computation

```
None
#[test]
fn issue_available_amount_computation() {
    let rust_zero = rust_biguint!(0);
    let mut fc_setup =
        FeesCollectorSetup::new(fees_collector::contract_obj,
energy_factory::contract_obj);
    let mut router_setup = RouterSetup::new(
        fc_setup.b_mock.clone(),
        router::contract_obj,
        pair::contract_obj,
    );
    // Setup router pairs and liquidity
    router_setup.add_liquidity();

    // Create users that will claim rewards
    let first_user =
fc_setup.b_mock.borrow_mut().create_user_account(&rust_zero);
```

```
    let second_user =
fc_setup.b_mock.borrow_mut().create_user_account(&rust_zero);
    fc_setup.set_energy(&first_user, 10, 5_000);
    fc_setup.set_energy(&second_user, 10, 5_000);

    // Setup router and add all tokens to reward_tokens list
    let router_address = router_setup.router_wrapper.address_ref().clone();
    fc_setup
        .b_mock
        .borrow_mut()
        .execute_tx(
            &fc_setup.owner_address,
            &fc_setup.fc_wrapper,
            &rust_zero,
            |sc| {
                sc.set_router_address(managed_address!(&router_address));

                // Only add the extra tokens, as BASE_ASSET_TOKEN_ID was added
at deployment
                let mut tokens = MultiValueEncoded::new();
                tokens.push(managed_token_id!(USDC_TOKEN_ID));
                sc.add_reward_tokens(tokens);
            },
        )
        .assert_ok();


    // Go to week 4 otherwise get_token_available_amount always returns 0
    fc_setup.advance_week(); // current_week = 2
    fc_setup.advance_week(); // current_week = 3
    fc_setup.advance_week(); // current_week = 4

    // Register users for rewards in week 4
    fc_setup.claim(&first_user).assert_ok();
    fc_setup.claim(&second_user).assert_ok();

    let usdc_token_weekly_amount = 500u64;
    let current_week = fc_setup.get_current_week();
    fc_setup.simulate_increase_accumulated_fees(
        current_week,
        USDC_TOKEN_ID,
        usdc_token_weekly_amount,
    );
```

```
    fc_setup.advance_week(); // current_week = 5
    fc_setup.claim(&first_user).assert_ok();

    let wegld_mex_pair_addr =
router_setup.wegld_mex_pair_wrapper.address_ref().clone();
    let wegld_usdc_pair_addr =
router_setup.wegld_usdc_pair_wrapper.address_ref().clone();
    fc_setup
        .b_mock
        .borrow_mut()
        .execute_tx(
            &fc_setup.owner_address,
            &fc_setup.fc_wrapper,
            &rust_zero,
            |sc| {
                let mut swap_operations = MultiValueEncoded::new();
                swap_operations.push(
                    (
                        managed_address!(&wegld_usdc_pair_addr),
                        managed_buffer!(SWAP_TOKENS_FIXED_INPUT_FUNC_NAME),
                        managed_token_id!(WEGLD_TOKEN_ID),
                        managed_biguint!(1),
                    )
                        .into(),
                );
                swap_operations.push(
                    (
                        managed_address!(&wegld_mex_pair_addr),
                        managed_buffer!(SWAP_TOKENS_FIXED_INPUT_FUNC_NAME),
                        managed_token_id!(BASE_ASSET_TOKEN_ID),
                        managed_biguint!(1),
                    )
                        .into(),
                );

                sc.swap_token_to_base_token(managed_token_id!(USDC_TOKEN_ID),
swap_operations);
            },
        )
        .assert_ok();

    // ISSUE: THE USER CAN'T CLAIM HIS REWARDS BECAUSE NO USDC LEFT IN THE
CONTRACT
    fc_setup.claim(&second_user).assert_error(10, "insufficient funds");
```

```
}
```

Recommendation

In the method `get_token_available_amount`, we recommend computing correctly the amounts of rewards allocated to past weeks `current_week - 4, …, current_week`, i.e. as the sum of tokens found in `accumulated_fees(week)` and in `total_rewards_for_week(week)`.

Moreover, we would fix the test `redistribute_rewards_test` so that it correctly computes the amount of rewards that should be redistributed.

2. Anyone can send XMEX to deposit_swap_fees to be distributed as rewards, and the energy of that XMEX will be counted twice

Status	Solved
Severity	Major
Commit (if not initial)	
Location file (optional)	
Additional note (optional)	

Description

The endpoint `deposit_swap_fees` is now open for anyone to send any token. However, if XMEX is sent (which is possible since the Fees Collector has transfer role for XMEX), then the energy of this XMEX is not deducted from the sender's energy, but this energy will be added to the energy of the users who claim it.

As a result, the total energy of all users will be inflated, and will not correspond to the XMEX in circulation.

This will make it much harder for the xExchange team when debugging problems to make energy computations and make them match with the XMEX in circulation.

Example: All users send their XMEX to the Fees Collector, and so their total energy after claiming the XMEX back as rewards will be 2x bigger than it should. If they repeat this operation, their energy would be 3x bigger than it should, and so on.

Recommendation

In the endpoint `deposit_swap_fees`, if the received token is XMEX, we recommend verifying that the caller is the *Token Unstake* smart contract (or a whitelisted address), as this is the only address supposed to send XMEX to the Fees Collector for rewards distribution.

If this solution does not fit with the team, we suggest reaching out to the auditor.

3. Available tokens are underestimated if redistribution is not performed 1st in a week (i.e. before users claim)

Status	Solved
Severity	Medium
Commit (if not initial)	
Location file (optional)	
Additional note (optional)	

Description

The method `get_token_available_amount` is supposed to compute the amount of tokens in the contract which are not allocated as rewards for past 4 weeks as well as the current week: `current_week - 4, current_week - 3, …, current_week`.

For this, it subtracts the amounts allocated for these weeks to the balance of the smart contract. However, if some users already claimed rewards for these past weeks before the method is called, then these rewards would have been withdrawn from the contract, making the SC balance smaller, hence resulting in an underestimated amount of available rewards to redistribute.

Consequences: The amount of rewards to redistribute or swap to MEX would be underestimated, making users earn less than they should.

Example: Alice is the only participant in the Fees Collector.

- At week 1, 500 MEX are allocated as rewards.
- At week 5, 500 MEX are allocated as rewards.

- At week 6, Alice claims, so she earns 500 MEX (since she can't claim for week 1).
- At week 6, MEX rewards are redistributed: since the smart contract balance is 500, `get_token_available_amount` returns `500 - 500 = 0`.

Therefore, no rewards were redistributed although 500 MEX should have been redistributed.

Recommendation

We recommend introducing a storage that stores the total amount of rewards already claimed by users for a given week.

```
None
fn rewards_claimed_for_week(token_id: EsdtTokenIdentifier, week: Week)
  -> SingleValueMapper<BigUint>;
```

It is increased each time a user claims rewards, for each week of claimed rewards, and in `get_token_available_amount`, the rewards claimed in the last 4 weeks are added to the smart contract balance.

As a result, after the upgrade is done:

- For the first 4 weeks, `get_token_available_amount` would underestimate the non-MEX tokens that can be swapped to MEX, because `rewards_claimed_for_week` would be 0 for the 4 weeks preceding the upgrade, resulting in an overestimation of the unclaimed rewards for these weeks. However once 4 weeks will pass, the underestimation would disappear and rewards would be swappable to MEX
- From week 5 and onwards, `get_token_available_amount` would return the exact amount of non-claimable rewards.

Finally, we recommend adding a unit test showing that the right amount of rewards is being redistributed even if some users claim before the redistribution occurs.

4. MEX and XMEX can be removed from the list of claimable tokens

Status	Solved
Severity	Medium
Commit (if not initial)	
Location file (optional)	

Additional note (optional)	

Description

Through the endpoint `remove_reward_tokens`, the admin can remove MEX and XMEX from the list `reward_tokens`. From then on, this would prevent users from claiming MEX and XMEX rewards.

However, it is expected that users should always be able to claim these tokens.

Recommendation

In the endpoint `remove_reward_tokens`, we suggest verifying that the tokens being removed from the list `reward_tokens` are distinct from MEX and XMEX.

```
None
fn remove_reward_tokens(token_ids: MultiValueEncoded<TokenIdentifier>) {
  let locked_token_id = self.get_locked_token_id();
  let base_token_id = self.get_base_token_id();
  for token_id in token_ids {
      require!(token_id != locked_token_id && token_id != base_token_id);
  }
...
}
```

5. Removing a reward token can make some rewards unclaimable for four weeks

Status	Solved
Severity	Medium
Commit (if not initial)	
Location file (optional)	
Additional note (optional)	

Description

When the admin deletes a token from the list `reward_tokens`, the tokens which were deposited so far in the current week won't be swappable to MEX for the next 4 weeks, and in turn users won't be able to claim these rewards for the next 4 weeks.

This is because, once a token is removed from `reward_tokens`, it is no longer being moved at the 1st interaction of the next week in `collect_rewards_for_week` from the storage `accumulated_fees` to the storage `total_rewards_for_week`. Indeed, this method only iterates over the list `reward_tokens`.Consequently:

- Users can't claim these rewards, because they can claim only rewards recorded in `total_rewards_for_week`.
- These tokens can't be swapped for MEX for 4 weeks, because during these 4 weeks these tokens won't be considered as available for swapping as they are present in the storage `accumulated_fees`.

Recommendation

In the endpoint `remove_reward_tokens`, we suggest clearing the storage `accumulated_fees` for the current week. This ensures that the tokens deposited for the current week can immediately be swapped to MEX and claimed by users.

6. Deleting locked_token_id from code without deleting storage "lockedTokenId" increases future risk of backward compatibility

Status	Solved
Severity	Medium
Commit (if not initial)	
Location file (optional)	
Additional note (optional)	

Description

The storage function `locked_token_id` has been deleted from the code, however the underlying storage *"lockedTokenId"* will still be present on mainnet. Therefore, if in the future, a storage is introduced with a key sharing a common prefix with *"lockedTokenId"*, this might cause backward compatibility issues.

Recommendation

In the `upgrade` endpoint, we recommend deleting the storage *"lockedTokenId"*, as already done for all other storages which are removed from the code.

7. List “reward_tokens” could theoretically be too big and make claims fail

Status	Solved
Severity	Medium
Commit (if not initial)	
Location file (optional)	
Additional note (optional)	Sub-issue of a previously raised issue, now easily solvable.

Description

At the 1st user interaction of each week, the method `collect_rewards_for_week` is called, which iterates over the entire list `reward_tokens`. In theory, this list could be arbitrarily big, and if too big it would make the transaction exceed limits of gas or built-in functions’ calls.

In practice, this should never occur since `reward_tokens` is not expected to grow: it should be kept to the tokens currently whitelisted on mainnet (around 10 tokens), and might be reduced to contain only MEX and XMEX in the future. Still, nothing prevents the problem from happening in theory, e.g. if later in time some new tokens are added in the list `reward_tokens`.

Recommendation

As detailed in another issue of this report, we can simply delete the endpoint `add_reward_tokens`, which resolves this issue.

Alternatively, if another approach is preferred: in the admin endpoint `add_reward_tokens`, we can verify that `reward_tokens` does not exceed a maximum size `MAX_REWARD_TOKENS` of 20 tokens.

8. Can’t swap tokens to MEX for first 4 weeks of Fees Collector

Status	Solved

Severity	Medium
Commit (if not initial)	
Location file (optional)	
Additional note (optional)	

Description

The endpoint `swap_token_to_base_token` that swaps non-MEX tokens to MEX so that users can claim, computes the amount of non-MEX tokens to swap thanks to the method `get_token_available_amount`.

However, the method `get_token_available_amount` returns 0 if the current week is not at least 5.

```
None
if current_week < USER_MAX_CLAIM_WEEK {  return BigUint::zero(); }
```

This means that if a new Fees Collector is deployed and swap fees are deposited in tokens like USDC, EGLD and others, then these tokens can't be swapped to MEX during the first 4 weeks, hence users can't earn rewards from swap fees during the first 4 weeks.

Recommendation

In the method `get_token_available_amount`, we recommend not returning 0 by default for weeks smaller than 5, and rather properly computing the oldest week `start_week` from which allocated rewards should not be considered available.

```
None
let start_week =
  if current_week >= USER_MAX_CLAIM_WEEKS {
     current_week - USER_MAX_CLAIM_WEEKS
  } else {
     0
  }
```

9. Unnecessary endpoint “add_reward_tokens”

Status	Solved
Severity	Minor
Commit (if not initial)	
Location file (optional)	
Additional note (optional)	

Description

The endpoint `add_reward_tokens` is unnecessary because it is not planned to ever add any new reward tokens. Namely, 6 weeks after the upgrade, all reward tokens will be removed except MEX / XMEX.

Recommendation

We recommend deleting the endpoint `add_reward_tokens`.

Unresolved past issues from previous audits

Disclaimer: *Below we list the links to non-resolved issues from previous audit reports, that the team decided not to solve at that time. Therefore, if the team now wants to solve some of these issues, we suggest reaching out to the auditor, since some recommendations might need to be modified to be backward compatible.*

10. Rewards can’t be claimed for a past week without user interaction

Status	Not Solved
Severity	Medium
Commit (if not initial)	
Location file (optional)	
Additional note (optional)	Issue raised in a previous review.

	Note: Solving the issue for XMEX would be relatively easy. Namely, in `accumulate_additional_locked_tokens`, we could compute `new_tokens_amount` as:
	
	None
	`let missed_weeks = if last_update_week == 0 { 1 } else { current_week - last_update_week };`
	`let new_tokens_amount = amount_per_epoch * epochs_in_week * missed_weeks;`

11. Typo opt_existing_claim_progres, should be opt_existing_claim_progress

Status	won't solve
Severity	Minor
Commit (if not initial)	
Location file (optional)	See function `update_user_energy_for_current_week`
Additional note (optional)	

12. Misleading endpoint name claimBoostedRewards

Status	won't solve
Severity	Minor
Commit (if not initial)	
Location file (optional)	
Additional note (optional)	

Additional Feedback

13. Block-to-epoch conversion does not work on all chains

In the upgrade endpoint, the amount of XMEX to mint per block is translated into an amount of XMEX to mint per epoch, and this computation assumes that an epoch lasts 14400 blocks. Although this assumption is true on mainnet, it does not hold on all chains, e.g. on devnet.

```
None
let locked_tokens_per_block_mapper =
  SingleValueMapper::new(StorageKey::new(b"lockedTokensPerBlock"));
let locked_tokens_per_block = locked_tokens_per_block_mapper.take();
let locked_tokens_per_epoch = locked_tokens_per_block * 10u64 * 60u64 * 24u64;
// 14400 blocks per epoch
self.locked_tokens_per_epoch().set_if_empty(locked_tokens_per_epoch);
```

This would not be particularly harmful since the owner can always change `locked_tokens_per_epoch`. But if the team wants a proper way to make the conversion in the upgrade endpoint, they can reach out to the auditor.

Mitigation: The team added an optional parameter `blocks_per_epoch_opt` *to the* `upgrade` *endpoint, to overwrite the value of* `blocks_per_epoch`*. It will not be used when upgrading on mainnet.*

14. Special fee burn will be applied twice to some tokens

Once the upgrade is done, the non-MEX/XMEX tokens in the smart contract which are non-claimable and were partially burnt in the past by a pair, will be partially burnt again when converted to MEX later on.

The team is aware of it and deems the amount affected should be relatively small. However, if they want to prevent the double burn from happening overall, they can reach out to the auditor.

15. Fees Collector needs burn role for MEX

The Fees Collector needs to acquire the burn role for MEX (which it does not have yet, at the time this audit is made), in order to be able to receive MEX in the endpoint

`deposit_swap_fees`, and when converting tokens to MEX in the endpoint `swap_token_to_base_token`.

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.