Excellar Audit

April 2024

By CoinFabrik

v202311

Executive Summary **3**
Scope **3**
Methodology **3**
Findings **4**
Severity Classification 5
Issues Status 5
Critical Severity Issues 6
CR-01 Current Balance Ignored in Rewards 6
High Severity Issues 6
Medium Severity Issues 6
ME-01 Stealthy Token Minting 6
Minor Severity Issues 7
MI-01 Unbound Instance Storage 7
MI-02 Claiming Rewards Frequently Leads to Better Rewards 7
MI-03 Unbound AMM Depositors 7
MI-04 Extra Rewards When Going Back to not AMM 8
MI-05 Transfers not Considered for Rewards 9
Enhancements 9
EN-01 Tests Not Running 9
EN-02 AMM Operations 9
Other Considerations **10**
Upgrades 10
Centralization 10
Privileged Roles 10
excellar-token-contract 11
Rewards Mechanism 11
Changelog **12**

Executive Summary

CoinFabrik was asked to audit the contracts for the Excellar project.

During this audit we found one critical issue, one medium issue and several minor issues. Also, two enhancements were proposed. The critical and medium issues were resolved and some of the minor issues as well. The other minor issues were acknowledged. One of the enhancements was implemented.

Scope

The audited files are from the git repository located at https://github.com/excellar-labs/excellar-contracts.git. The audit is based on the commit `17c344f2ea30646c761dcd6590cdb99c88a44bfe`. Fixes were checked on commit `5944a89dc024cfb857d69d4c5a7780943324c5e2`.

The scope for this audit includes and is limited to the following files:

- `token/src/admin.rs`: Utility functions for admin operations.
- `token/src/allowance.rs`: Utility functions to manage the token allowances.
- `token/src/amm.rs`: Utility functions to manage automated market makers interactions.
- `token/src/balance.rs`: Utility functions to operate with the account balances and total token supply.
- `token/src/contract.rs`: `excellar-token-contract` implementation.
- `token/src/event.rs`: Utility functions to publish events.
- `token/src/lib.rs`: Library entry point.
- `token/src/metadata.rs`: Utility functions to interact with the token's metadata.
- `token/src/reward.rs`: Utility functions to manage rewards.
- `token/src/storage_types.rs`: Type definitions for stored data.

No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

Methodology

CoinFabrik was provided with the source code, including automated tests that define the expected behavior. Our auditors spent one and a half weeks auditing the source code provided, which includes understanding the context of use, analyzing the boundaries of the expected behavior of each contract and function, understanding the implementation by the development team (including dependencies beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. Without being limited to them, the audit process included the following analyses.

- Arithmetic errors
- Race conditions
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

After delivering a report with our findings, the development team had the opportunity to comment on every finding and fix the issues they considered convenient. Once fixed and/or commented, our team ran a second review process to verify that the changes to the code effectively solve the issues found and do not unintentionally add new ones. This report includes the final status after the second review.

Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below. This table does not include the enhancements we suggest to implement, which are described in a specific section after the security issues.

ID	Title	Severity	Status
CR-01	Current Balance Ignored in Rewards	Critical	Resolved
ME-01	Stealthy Token Minting	Medium	Resolved
MI-01	Unbound Instance Storage	Minor	Resolved
MI-02	Claiming Rewards Frequently Leads to Better Rewards	Minor	Acknowledged
MI-03	Unbound AMM Depositors	Minor	Acknowledged

ID	Title	Severity	Status
MI-04	Extra Rewards When Going Back to not AMM	Minor	Resolved
MI-05	Transfers not Considered for Rewards	Minor	Acknowledged

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. Blocking bugs are also included in this category. They must be fixed **immediately**.
- **High:** These refer to a vulnerability that, if exploited, could have a substantial impact, but requires a more extensive setup or effort compared to critical issues. These pose a significant risk and **demand immediate attention**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of, but might be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit has one of the following statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code, but is a result of an intentional decision. The reported risk is accepted by the development team.
- **Resolved**: Adjusted program implementation to eliminate the risk.
- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk.

Critical Severity Issues

CR-01 Current Balance Ignored in Rewards

In the audited contract, the current balance is ignored, even if it was held for a long time by the user, for the purpose of gaining rewards. For example, an account that receives 1000 tokens will get 0 rewards even after 2 days of holding the position.

Recommendation

Take into consideration the current position to calculate the awarded rewards. In order to do so, call `checkpoint_reward` in the `claim_reward` and `admin_claim_reward` functions.

Status

Resolved. The `checkpoint_reward` call was added.

High Severity Issues

No issues found.

Medium Severity Issues

ME-01 Stealthy Token Minting

Location:

- `token/src/contract.rs: 67, 81, 92`
- `token/src/balance.rs: 28-32`

When rewards are claimed via the `claim_reward` or `admin_claim_reward` functions, defined in `token/src/contract.rs`, new tokens are generated and the total supply is increased but no event is emitted to account for those, unlike when new tokens are generated via the mint function also defined in `token/src/contract.rs`.

Recommendation

Emit mint events every time the total supply of tokens increases. To do so, extract a single function that increases the user balance, increases the total supply accordingly and emits the corresponding mint event, and use this function in all the situations where the total token supply is increased.

Status

Resolved. Token minting events were added to the reward claiming functions.

Minor Severity Issues

MI-01 Unbound Instance Storage

Location:

- `token/src/admin.rs: 22, 47, 60`

KYC-approved addresses, blacklisted addresses and AMM addresses are stored in the instance storage. So when each address is added an extra cost is incurred on every interaction with the `excellar-token-contract` contract, and if the required storage exceeds 64Kb it may lead to a denial of service.

The severity of this issue was lowered because only the admin can trigger it.

Recommendation

Store each entry on its own persistent storage slot.

Status

Resolved. Now entries are stored as recommended.

MI-02 Claiming Rewards Frequently Leads to Better Rewards

Location:

- `token/src/contract.rs: 70`

Claiming rewards frequently increases the total obtained rewards, as the funds obtained in the early reward claims are used to calculate the latter reward claims.

Please note that in the current code funds need to be transferred between accounts to workaround CR-01 Current Balance Ignored in Rewards.

Recommendation

Take into account the nature of compound interest and its geometric nature when designing rewards.

Status

Acknowledged.

MI-03 Unbound AMM Depositors

Location:

- `token/src/amm.rs: 69`

Each time a new account makes a deposit to an AMM account its address is added to the depositors of this AMM account. The number of depositors is unbounded so it may exceed the maximum of 64Kb allowed for a storage entry. And each time a deposit is made into the AMM account, the operations for this account get more expensive and the price for the storage gets more expensive as well.

Recommendation

Keep the deposited amount for each `(depositor, amm)` pair on a different persistent-storage entry and modify the logic accordingly. Give the depositor a mechanism to claim the rewards corresponding to each AMM individually, to avoid iteration in a single transaction.

Status

Acknowledged.

MI-04 Extra Rewards When Going Back to not AMM

Sometimes, when transfers are made to AMM accounts and then the accounts are moved back to non-AMM accounts extra rewards appear.

We managed to reproduce by doing the following steps:

1. Set the reward rate as 30%.
2. Mark the accounts `u1`, `u2` as kyc-passed.
3. Mark the accounts `a1` and `a2` as AMMs.
4. Fund account `u1` with 1000 tokens.
5. Transfer the 1000 tokens from `u1` to `a1`.
6. Wait one day.[1]
7. Transfer the 1000 tokens from `a1` to `a2`.
8. In the same block, also transfer the tokens from `a2` to `u2`.
9. Claim rewards for `u1`. As expected the account will have 300 tokens.
10. Mark the `a1` account as not an AMM, and kyc-passed.
11. Claim the rewards for `a1`. The account will also have 300 tokens, generating 600 tokens of rewards even when only 300 tokens are expected to be generated in a day with these settings and amounts used.

Recommendation

AMM accounts should be zero-balanced when set as such and when set back to non-AM and not accumulate any rewards for themselves.

Status

Resolved. The test stated above passes now.

[1] This is made synthetically by advancing the sequence number.

MI-05 Transfers not Considered for Rewards

Location:

- `token/src/contract.rs: 245-249, 266-270`

When funds are transferred away from an AMM to another AMM in the audited contract this fund release is not taken into account to decide the rewards distribution between all the non-AMM accounts that funded the AMM.

Recommendation

Take into consideration transfers made from an AMM account to another AMM account.

Status

Acknowledged.

Enhancements

These items do not represent a security risk. They are best practices that we suggest implementing.

ID	Title	Status
EN-01	Tests Not Running	Implemented
EN-02	AMM Operations	Not implemented

EN-01 Tests Not Running

When `cargo test` is run in the shell a compilation error happens.

Recommendation

Make the tests run.

Status

Implemented. Now the tests run when `cargo test` is executed as documented in the `README.md` file.

EN-02 AMM Operations

As documented in the Rewards Mechanism section, the rewards awarded to an AMM account are expected to be distributed to the account depositors. This mechanism will award rewards to both accounts that provide liquidity to the AMMs and those that transfer funds to it as a normal part of its operation. It also awards these rewards forever. It also prevents the AMM from using the rewards, or part of it, to fund its operations.

Recommendation

Decouple AMM strategies from the token. Give the rewards to the account irrespectively of the AMM status. Distinguish deposits and withdrawals of liquidity from normal AMM operations and only distribute funds to the accounts that provide liquidity for the AMM. This may include other AMMs. Allow users to withdraw funds given to provide liquidity. Each AMM should have its shares handled separately. You may choose to implement part of the AMMs as smart contracts, or do them fully off-chain.

Status

Not implemented.

Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest to other stakeholders of the project, including users of the audited contracts, token holders or project investors.

Upgrades

The admin can upgrade the contract code of the `excellar-token-contract` via the `upgrade` function.

Centralization

Besides changing all the code of the `excellar-token-contract` its admin can take other actions that affect the rest of the system:

- It can make accounts not eligible to receive funds via the `blacklist` function.
- It can deny rewards by marking an account as not passing the KYC. This also prevents the account from burning tokens, but not transferring them.
- It can deny rewards by marking an account as being an AMM, this is different from the item above, as new rewards are awarded to its depositors as explained in the Rewards Mechanism section.
- It can change the reward calculation by changing either the reward tick or the reward rate.

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

excellar-token-contract

Admin

An account with the admin role can:

1. Claim a reward for another account via the `admin_claim_reward` function.
2. Mark an account as failing KYC via the `fail_kyc` function.
3. Mark an account as passing KYC via the `pass_kyc` function.
4. Blacklist an account to receive funds or transfer funds on behalf of other accounts via the `blacklist` function.
5. Allow an account to receive funds or transfer funds on behalf of other accounts via the `whitelist` function.
6. Mark an account as an AMM (automated market maker) via the `add_amm_address` function.
7. Mark an account as not being an AMM (automated market maker) via the `remove_amm_address` function.
8. Mint tokens for an account via the `mint` function.
9. Claim rewards for another account via the `admin_claim_reward` function.
10. Set another account as admin via the `set_admin` function.
11. Upgrade the contract via the `upgrade` function.
12. Set the reward rate via the `set_reward_rate` function.
13. Set the reward tick via the `set_reward_tick` function. The initial reward tick is 28800, which is expected to be roughly the number of ledger advances in a day.

The initial admin is passed as a parameter to the `initialize` function, which is expected to be run once to initially configure the contract.

Rewards Mechanism

The rewards mechanism was explained to us by the development team as follows:

Rewards work depending on whether an account is identified as AMM or not:

1. In the case when it is not an AMM, the account accumulates `reward_rate * balance * number_of_blocks_held/reward_tick`, which means that every `reward_tick` the account will receive the full `reward_rate` of their balance, provided that its balance has not changed.
2. In the case where the address holding the tokens is an AMM, because the admin flagged it as such, then the reward that would otherwise have been accumulated and distribute will instead be distributed to all the participants in the AMM pro-rata of their total deposit, i.e. instead of the reward being given to the AMM contract, it would be given to the depositors in the pool.

In both cases for the reward to become part of their token balance, the user needs to manually claim, or the admin needs to claim on their behalf.

Besides that, it must be noted that the rewards are calculated when a transfer, burn or mint occurs and uses the reward rate and reward tick settings at the time of the calculation. Also see CR-01 Current Balance Ignored in Rewards.

Changelog

- 2024-04-12 – Initial report based on commit `17c344f2ea30646c761dcd6590cdb99c88a44bfe`.
- 2024-04-23 – Check fixes on commit `5944a89dc024cfb857d69d4c5a7780943324c5e2`.