# Qiibee Token Audit

OPENZEPPELIN SECURITY | DECEMBER 19, 2017

Security Audits

The Qiibee team asked us to review and audit their QBX Token contracts. We looked at the code and now, publish our results.

The audited code is located in the qiibee/qb-contracts repository. The version used for this report is commit `d40368c9a7a536572a5bb03cb031d658ccb34f24`. We restricted our audit only to the following contracts: `MigrationAgent`, `QiibeePresale` and `QiibeeToken`.

Here is our assessment and recommendations, in order of importance.

***Update:*** *The Qiibee team has followed the majority of our recommendations and updated the contracts. The new version is at commit* `25efdbf5bc29de12a724450c540218f6c8e59129`.

## Critical Severity

### Tokens burning breaks MigrationAgent contract

Tokens burning after the deployment of the `MigrationAgent` creates a discrepancy between `tokenSupply` defined in the token and the migration contract, which will irreversibly break the `MigrationAgent`. Once the `MigrationAgent` contract is deployed, it defines `tokenSupply` to match the current state of `qbxSourceToken`. However, when the supply of the source token decreases after burning, the `tokenSupply` variable is not updated causing

Consider restricting the access to the `burn` function solely to the `QiibeeToken` owner. If this solution cannot be implemented, we suggest updating `tokenSupply` variable of after every `burn` operation as an alternative.

***Update**: Fixed in this commit by both restricting the access to the `burn` function and counting the `burntTokens`, so the `MigrationAgent` can `updateSupply`.*

#### Migration bypasses vesting restrictions

All of the users are allowed to `migrate` their tokens to a new contract regardless of whether their assets are fully transferable or locked in a vesting scheme. Moreover, once they migrate the tokens any vesting restrictions are removed.

We recommend only allowing `transferableTokens` to be available for migrations. An alternative solution is to implement a new `migateVestedTokens` function that will copy the vesting configuration to the migration target contract.

***Update:** Fixed in this commit by checking if the amount of tokens is within the `transferableTokens` limit at the moment of migration.*

## High Severity

#### Token minting is not synchronized with MigrationAgent

Token minting leads to an inconsistent state as the `MigrationAgent` tokenSupply is never increased after being set in the constructor. The discrepancy between the `QiibeeToken` and the `MigrationAgent` states, breaks the `safetyInvariantCheck` and corrupts the migration process.

We suggest enforcing that the token has finalized the minitingby adding the check `require(_qbxSourceToken.mintingFinished)` in line 21 of the `MigrationAgent`. If the team wants to continue the minting after the migration is deployed we recommend increasing the `tokenSupply` by the amount of newly created tokens.

***Update:** Fixed in this commit by tracking `newTokens` and updating the `MigrationAgent` state in the `updateSupply` function.*

There are three unchecked math operations inside the migration function in lines 114–116 . It's always better to be safe and perform operations with correctness assertions.

Consider rigorously checking for under and overflows for all of the arithmetic operations. We recommend using the `SafeMath` library from OpenZeppelin.

***Update:*** *Fixed in this commit.*

#### Constant names incompatible with the ERC20 standard

`QiibeeToken` declares the obligatory `ERC20` standard parameters as uppercase constants: `SYMBOL`, `NAME`, `DECIMALS`. This conflicts with the official specification that requires the names to be lowercase: `symbol`, `name`, `decimals`.

Consider renaming constants to lowercase, so they are compliant with the official `ERC20` standard.

***Update:*** *Fixed in this commit.*

### Low Severity

#### Vesting logic implemented directly in the token contract

The vesting logic is currently implemented in the `QiibeeToken` in the form of a base `VestedToken` contract. These features are going to be used by a limited number of buyers and for a restricted amount of time only. Having complex logic included directly in the token contract may not only cause compatibility issues with blockchain explorers and exchanges, but it may also increase the potential attack surface.

We would suggest extracting the vesting logic into a separate contract as implemented in the OpenZeppelin pull request.

***Update:*** *The Qiibee team explained, that because of legal matters, they prefer to keep both functionalities within a single contract that controls token issuance.*

#### Validate MigrationAgent setting

corrupt the migration process.

We recommend checking the precondition `require(_agent.qbxSourceToken == address(this))` in line 102 to avoid being in an inconsistent state.

***Update:*** *Fixed in this commit.*

### Missing full sanity checks on adding accredited investors

It is possible to `addAccreditedInvestor` with `minInvest` being greater than `maxCumulativeInvest`. Although this configuration will be recorded properly, it will throw an exception on line 92 of the `buyTokens` function, therefore preventing the investor to take a part into the sale.

We recommend adding an extra precondition `require( minInvest &lt;= maxCumulativeInvest)` in line 119 so any potential errors are detected as early as possible.

***Update:*** *Fixed in this commit.*

## Notes & Additional Information

- We suggest adding timezone information to the date description: *start time for vested tokens (equiv. to 30/06/2018)*. (***Update:*** *fixed in this commit.*)
- Consider emitting an event similar to the `NewTokenGrant` from the `_VestedToken` base contract, once the tokens are minted for vesting. (***Update:*** *fixed in this commit.*)
- There is a `TODO` comment left in the code: "*is there any way to properly check this?*. Please make sure that all of the outstanding questions are resolved before deploying the code in the production mode. (***Update:*** *fixed in this commit.*)
- `QiibeeToken` inherits from a `BurnableToken` which provides a `burn` function allowing anyone to burn a certain token amount, destroying them and reducing `totalSupply`. Please make sure this is expected, or consider making the `burn` function only callable from the `Qiibee` address. Note that any ERC20 token allows owners to burn tokens by transferring them to the zero address or any random one, but this does not modify the `totalSupply`. (***Update:*** *fixed in this commit.*)

- Consider using the [illegible] function instead of duplicating the logic in `afterFundraising` modifier body. (***Update:*** *fixed in this commit.*)
- We recommend setting the token address in the `QiibeePresale` constructor ,mitigating the risk that the `setToken` function may not be called before the sale starts. (***Update:*** *fixed in this commit.*)

## Conclusion

Two critical severity issues and one high severity issue were found and explained, along with recommendations on how to fix them. Some changes were proposed to follow best practices and reduce potential attack surface.

*Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the QBX Tokencontracts. We have not reviewed the related Qiibee project. The above should not be construed as investment advice. For general information about smart contract security, check out our thoughts here.*

## Related Posts


### Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...


### OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust


### Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

OpenZeppelin