Centre Token Minting Contracts Audit

OPENZEPPELIN SECURITY | APRIL 17, 2019

Security Audits

The Circle and Coinbase teams asked us to review and audit the minting contracts of the Centre Token. We looked at the code and now publish our results.

The code is located in the centre-tokens repository, and the files audited are `Controller.sol`, `MintController.sol`, and `MasterMinter.sol` in the `contracts/minting` directory. In particular, we have not audited other files in the Centre Token project or how the

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

Update: *Circle and Coinbase teams made some fixes based on our recommendations. We address below the fixes introduced up to commit* `fbb6cfeb503ece6719cd456f82e6ef1602145db3`*. This update covers the three originally audited files plus the new* `MinterManagementInterface.sol` *file, which was split from* `MintController.sol` *following our recommendation.*

Critical Severity

None.

High Severity

None.

Medium Severity

Missing zero address checks

In several places in the code, addresses are passed as parameters to functions. In many of these instances, the functions do not validate that the passed address is not the address `0`. In the `Controller` contract, this happens in functions `configureController` (which only checks `_worker`) and `removeController`.

In the `MintController` contract, on the other hand, it might be expected that the constructor sets the `minterManager` variable to `0`, but the `setMinterManager` function does not prevent setting it to `0`.

While this does not currently pose a security risk, consider adding checks for the passed addresses being nonzero to prevent unexpected behavior where required, or documenting the fact that a zero address is indeed a valid parameter.

Update: *functions in* `Controller` *now check all passed arguments to prevent zero addresses from being set. No changes were introduced in this respect in* `MintController`.

No way to decrement an allowance without reconfiguring minter

minter. This mechanism is implemented in the `incrementMinterAllowance` function, which in turn calls `minterManager.isMinter` to perform the required check.

The contract, however, provides no analogous mechanism for *decreasing* a minter's allowance, meaning that this can only be achieved by "resetting" the minter through the `configureMinter` function.

It might well be that this is intended design and that decrementing minter allowances does not make sense in context; if this is not the case, however, consider providing this complementary functionality.

Update: *the* `decrementMinterAllowance` *function was introduced to handle allowance decrements without risking undesired minter reactivation. Its implementation is such that, if the value by which to decrement the allowance is greater than the current allowance, the latter is set to zero. Consider documenting this behavior inline.*

Note that no checks are here performed as to whether allowance decrements are taking place after *tokens were already minted, since the actual minting logic is handled elsewhere. The interaction between minting and allowance management is outside the scope of this audit, consider running a full security audit on the corresponding contracts as well, if not done already.*

Low Severity

Experimental version of Ownable used

`Controller.sol` imports a custom version of `Ownable`, which was taken from the `labs` repository of ZeppelinOS and later edited. This is experimental code, and, as stated in the README, not meant for production. As a side note, the import is made with `import './../Ownable.sol'`, which has a superfluous leading `./`.

Given that the project is already using OpenZeppelin, consider using its well-tested version of `Ownable`.

Lack of check might lead to gas burn

Consider checking that `allowanceIncrement > 0` to avoid unnecessary gas costs.

***Update:** the `incrementMinterAllowance` function now requires that its (renamed) `_allowanceIncrement` parameter is greater than `0`.*

Missing error messages in require statements

There are several `require` statements that provide no error messages (lines 43 and 57 in `Controller.sol`, and line 94 in `MinterController.sol`). Consider including specific and informative error messages in all `require` statements.

***Update:** all `require` statements now provide messages for the reverts.*

Unaudited version of OpenZeppelin used

The repository in which the audited contracts are held is using *openzeppelin-solidity* `v1.11.0`. This is an outdated release, and was never audited. At the time of writing the latest stable version is `v2.0.0`, which has gone through an external security audit.

Consider updating the project to the newer, audited version.

Room for improvement in contracts documentation

The Centre Token minting contracts are in general well documented, and the documentation adheres to the NatSpec format. There is, however, room for improvement. First, `MasterMinter.sol` has no documentation at all. Second, while all contracts have a short description of each function, all comments make use of only the `@dev` tag among the many available in NatSpec.

Yet another instance where documentation can be improved is the following comment in `Controller.sol`:*"set the controller of a particular _worker".* This might lead the user to think that each worker has a single controller, while the system in fact allows it to have more than one.

Finally, the comment *"allows control of configure/remove minter…"* in `MintController` seems to refer to the `configureMinter` and `removeMinter` functions, but these are not and functions, and having comments more clearly and faithfully represent the functionality of the contracts.

Update: *the inline documentation has been greatly improved, covering in particular all points reported above. The documentation refers in two occasions to a* `MinterManagerInterface`*, whose correct name is* `MinterManagementInterface`.

Notes & Additional Information:

- `Controller.sol`, `MintController.sol` and `MasterMinter.sol` are using an outdated Solidity release: `v0.4.24`. Consider using the latest version of the Solidity compiler (`v0.5.2` at the time of writing) throughout the code. `removeController` does not check if the controller is set before removal, allowing for subsequent calls to the function, which will confusingly emit repeated `ControllerRemoved` events with the same address. Consider whether this check should be included in order to avoid this behavior.
  Update: *a check was added to prevent this behavior.*
- `Controller` explicitly defines an empty constructor, which, according to the Solidity docs, is exactly what the contract would assume if this piece of code was absent. Consider removing the empty constructor for clarity.
  Update: *the empty constructor was removed.*
- Several functions (`configureController` and `removeController` in `Controller`, and `setMinterManager` in `MintController`) return hardcoded boolean values. Consider documenting this if it arises from the need of conforming to an interface.
  Update: *these functions no longer return a boolean.*
- Several functions in the code (`configureController` and `removeController` in `Controller`, and `setMinterManager`, `removeMinter`, `configureMinter`, and `incrementMinterAllowance` in `MintController.sol`) are marked as `public`, but are never called from within the contract hierarchy. Consider making these functions `external` if they are only to be called from other contracts.
- Consider moving `MinterManagementInterface` to an individual file and importing it from `MinterController.sol`, thus allowing it to be cleanly imported from other files in

`internal` or `private` and providing the required getter functions as a good encapsulation practice.

Update: *these variables are now* `internal`*, and the corresponding getters are in place.*

- In line 61 of `MintController`, the `minterManager` variable is implicitly converted to the `address` type. Starting from Solidity version `v0.5.0`, implicit address conversions issue an error. Consider explicitly casting `minterManager` to the `address` type in case the decision is later made to use a newer version of Solidity.

  Update: *the casting is now done explicitly.*

- Repeatedly in the code (`Controller.sol` lines 56, 66; `MintController.sol` lines 60, 71, 80, 92), the `public` keyword appears after the custom modifiers. According to the Solidity docs on Function declaration, however, *"The visibility modifier for a function should come before other modifiers".*

  Update: *the ordering of the modifiers now conforms to the suggested one.*

- The line wrapping is inconsistent in the codebase. Comments appear to be wrapped at around 86 characters but some parts of the code extend beyond that limit (e.g. line 49 of `MintController.sol`, which goes to 128). Consider adhering to a single limit and applying it throughout. The recommended width is 80 columns.

  Update: *line wrapping is now consistent throughout.*

- There is an inconsistency in function parameter naming. In `Controller` they all start with an underscore, while in `MintController` only those of `constructor` and `setMinterManager` do. Consider using underscores in all parameter names.

  Update: *all parameter names now start with underscores, except only for those of the new* `MinterAllowanceDecremented` *event. Apart from this, there is an extra blank line between the declaration of this new event and the preceding ones.*

- All `import` statements use single quotes, double quotes are recommended in the Solidity Style Guide.

  Update: `import` *statements now use double quotes.*

- There is a double space in line 52 of `MintController.sol`.

  Update: *the double space was removed.*

- Functions `incrementMinterAllowance` and `internal_setMinterAllowance` in `MintController` are not correctly indented.

  Update: *these indentation issues were resolved.*

practices and reduce the 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 Centre Token minting contracts. 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...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

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

Security Audits

OpenZeppelin