SOLIDIFIED

Audit Report for RAE. March 28, 2019.

Summary

Audit Report prepared by Solidified for RAE covering the token and minting smart contracts (and their associated components).

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the below token swap. The debrief took place on March 28, 2019 and the final results are presented here.

Audited Files

- RaeToken.sol
- RaeMintContract.sol

The files were audited on commit `83893c9a917926eec2c08183a99e2abbf27ad7db` and solidity compiler version `0.5.0`

Intended Behavior

The purpose of these contracts is to implement an ERC20 token with specific bulk minting functionality

Issues Found

Critical

No critical issues were found.

Major

1. Ineffective access control can lead to inconsistencies in state, possibly freezing the minting function

From `RaeMintContract`:

```
/**
* @dev create minting contract, passing token contract that will be the target of minting.
* On deployment, deployer of RaeToken contract will be given minterRole. With that minterRole the deployer will
* assign minterRole to this contract, and finally the deployer will revoke minterRole from himself so
* that this contract is the only possible minter for RaeToken. Owner of this contract will be the only person
* who can issue mint, bulkMint functions
*/
```

The contract contains a function that allows its owner to add other minters to the token contract. This not only violates the description above, but will also allows direct access to the token contract and this can result in inconsistencies in state, where mintPeriod will differ between the two contracts.

If this happens, the mint functions will always revert when halving occurs (because both contracts will require different `mintAmounts`).

Recommendation
Consider removing the `addMinter` function from `RaeMintContract`. Also take measures to ensure the deployer renounces `minterRole` as soon as the contracts are deployed.

Refer to Note #2 for additional recommendations around the minting functions.

Amended [04.04.2019]
The issue was fixed and is no longer present in commit `3382f4618ac225996a5cd5c741ce18db47ece240`.

Notes

2. Consider grouping minting functions

The RaeTokens minting process is divided into two contracts, `RaeToken` and `RaeMintContract`, but they share a lot of state and functionality. A better approach is to unify both functions in the `RaeMintingContract`, which will call the token's `mint()` function directly, instead of a intermediary function.

Amended [04.04.2019]
RAE has kept the functions separately, but ensured state is not duplicated between contracts, commit `3382f4618ac225996a5cd5c741ce18db47ece240`.

3. Violation of Checks, Effects, Interaction pattern

In `RaeMintContract`, function `bulkMintAggregator`, the variable `_mintPeriods` is set after the external call to the token takes place. This makes the function vulnerable to a reentrancy. In this context, this fact has no impact (since the token contract called is trusted), and therefore the vulnerability is being reported as a note.

Amended [04.04.2019]
The issue was fixed and is no longer present in commit `3382f4618ac225996a5cd5c741ce18db47ece240`.

4. Consider declaring non-changing variables as `constant`

Variables not supposed to change throughout the life of a smart contract, such as `halveEvery`, should be declared as `constant`. This will enforce the value that was present at compile time and will not allow changes to it.

Amended [04.04.2019]
The issue was fixed and is no longer present in commit `3382f4618ac225996a5cd5c741ce18db47ece240`.

5. Update compiler version and lock pragma

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

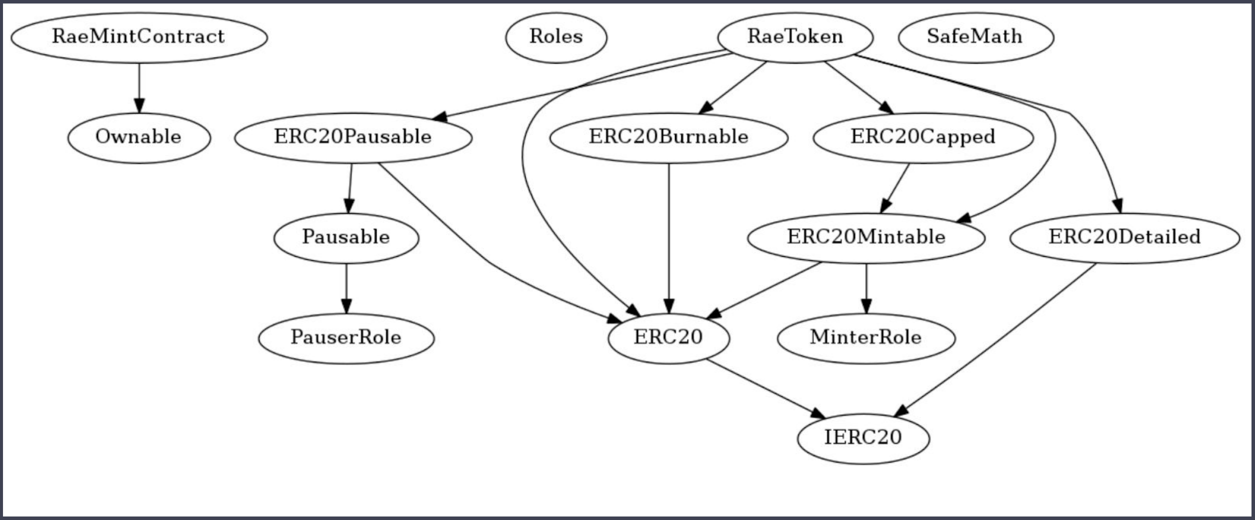
Also take note that the current floating pragma allows for compilation from 0.5.0 and up, consider updating to the latest version to avoid the three known bugs since 0.5.0 (https://solidity.readthedocs.io/en/latest/bugs.html).

Amended [04.04.2019]
The issue was fixed and is no longer present in commit `3382f4618ac225996a5cd5c741ce18db47ece240`.

6. Consider removing unnecessary imports

Solidity supports multiple inheritance, meaning that one contract can inherit several contracts. Multiple inheritance introduces ambiguity called Diamond Problem: if two or more base contracts define the same function, which one should be called in the child contract? Solidity deals with this ambiguity by using reverse C3 Linearization, which sets a priority between base contracts.

That way, base contracts have different priorities, so the order of inheritance matters. Neglecting inheritance order can lead to unexpected behavior.



Recommendation
Remove unnecessary inheritances (`ERC20` and `ERC20Mintable` inherited by `RaeToken` contract).

Amended[04.04.2019]
The issue was fixed and is no longer present in commit `3382f4618ac225996a5cd5c741ce18db47ece240`.

7. Consider declaring functions as `external`

The functions that are not supposed to called within the contracts should be marked as external, both to avoid unwanted calls and to potentially save gas when dealing with large arrays of data.

Amended [04.04.2019]
The issue was fixed and is no longer present in commit `3382f4618ac225996a5cd5c741ce18db47ece240`.

8. Add reasons for `revert`

Since version `0.4.22` of solidity, `require` statements can include an error string. Consider adding appropriate error messages to all require statements.

9. Consider importing interface instead of whole contract

The `RaeMintContract` imports the `RaeToken`, which means the whole token bytecode will be attached in the deployment code of the mint contract, increasing the gas costs. A more economic approach is to define a interface for `RaeToken` and import such interface to the mint contract.

Conclusion

One major issue was found, along with some minor issues that can affect the desired behavior and it's recommended that they're addressed before proceeding to deployment.

Disclaimer