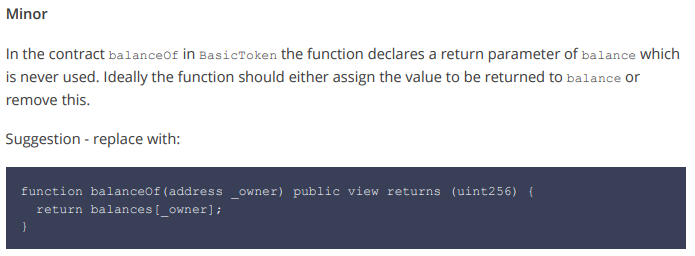
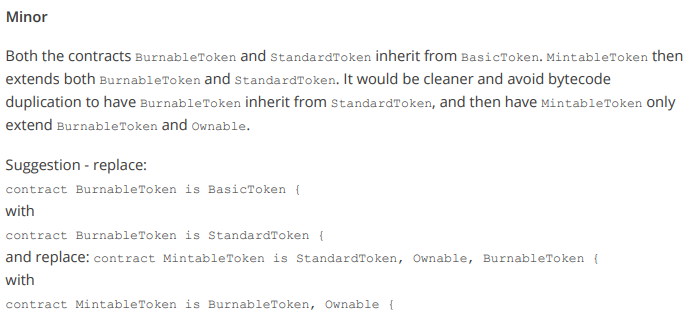
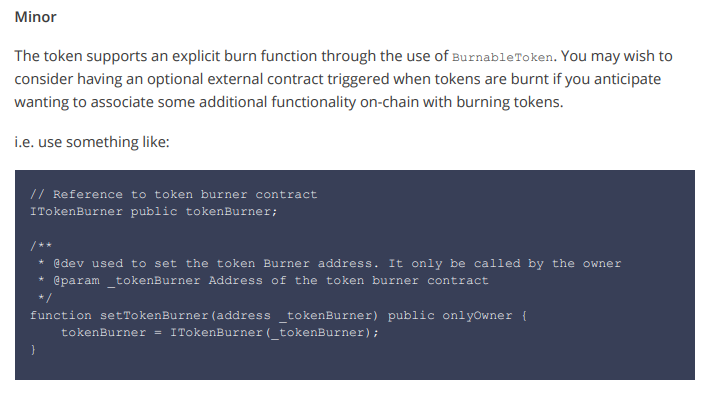
**ElepogToken.sol**



**Suggestion used**

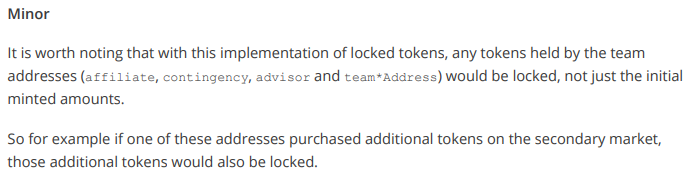


**Suggestion used**



**Suggestion not implemented:**

Although a good idea, this is out of the scope for what we needed the burn function to do.



**Suggestion not implemented:**

The wallet addresses teamY1, teamY2, teamY3, teamY4 are the only addresses that are locked (please see checkPermissions function). These addresses will only be used for the vesting program and not store any additional tokens.

function checkPermissions(address \_from) internal view returns (bool) {

// team vesting, a wallet gets unlocked each year.

if (\_from == teamY1 && now < unlockY1Time) {

return false;

} else if (\_from == teamY2 && now < unlockY2Time) {

return false;

} else if (\_from == teamY3 && now < unlockY3Time) {

return false;

} else if (\_from == teamY4 && now < unlockY4Time) {

return false;

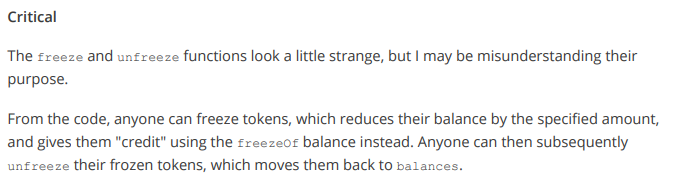
} else {

//all other addresses are not locked

return true;

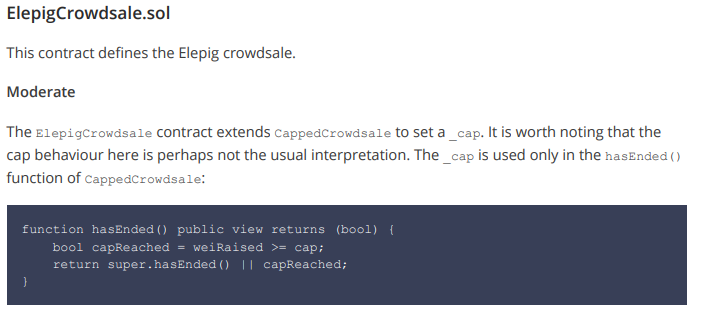
}

}

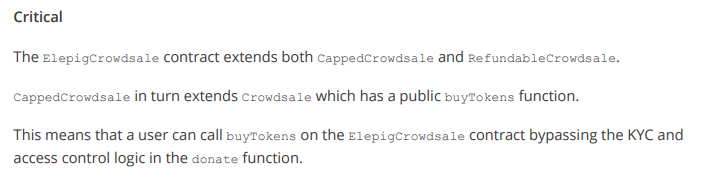


**Suggestion used**: removed the functions as they were decided to not be needed.

**ElepigCrowdsale.sol**

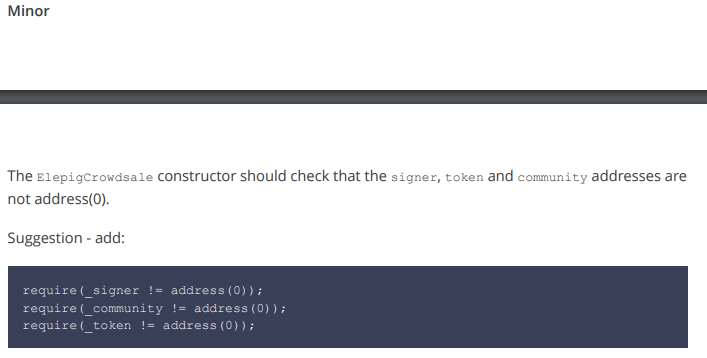


Useful insight, however this is still a hardcap as the cap set will be the total amount of tokens available in the sale (150MM) during deployment, so unless this amount is reached, we will have to wait for the time to run out. If the time does run out, and there are tokens left to sell, they will be minted into the community wallet. The cap will always be 150MM tokens. This isn’t clear in the contract and might be worth hard coding.

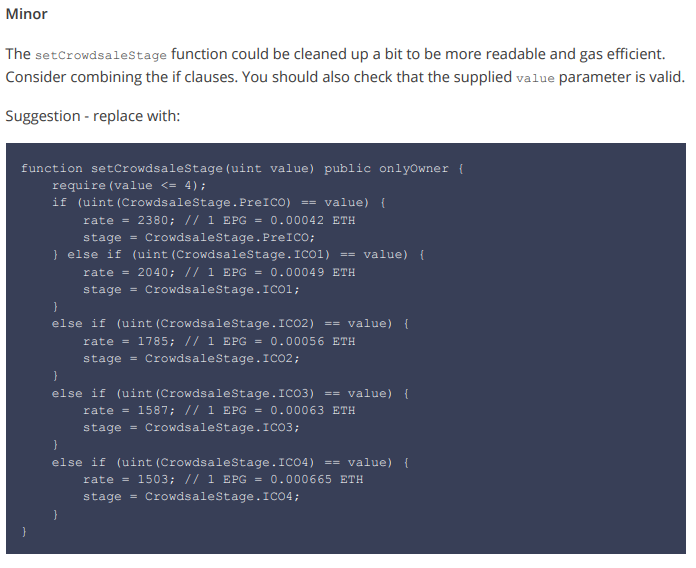


**Suggestion used:**

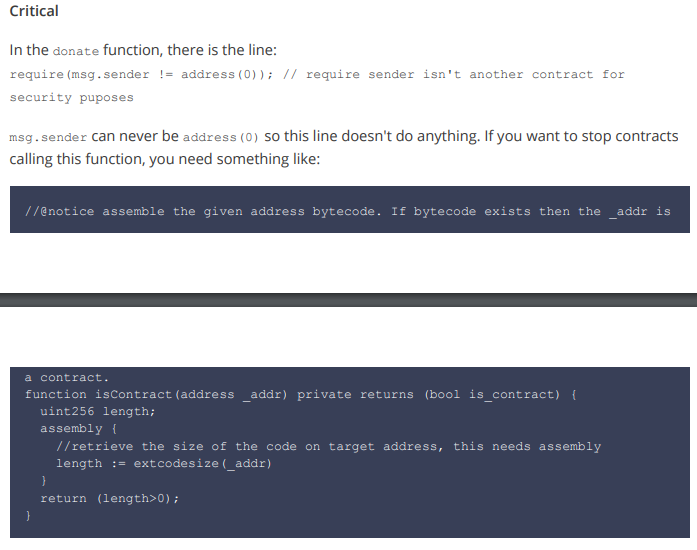
Removed the buyTokens function from the Crowdsale Token and implemented the functionality within the donate function.



**Suggestion used**

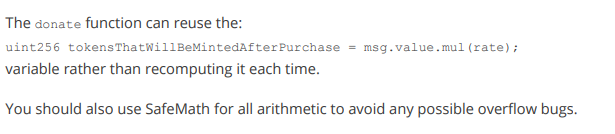


**Suggestion used**

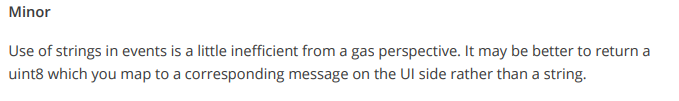


**Suggestion not implemented:**

As this function can only be called by a wallet address that has been through KYC, it can’t be called by another contract, so this issue is handled by the onlyValidAccess modifier.

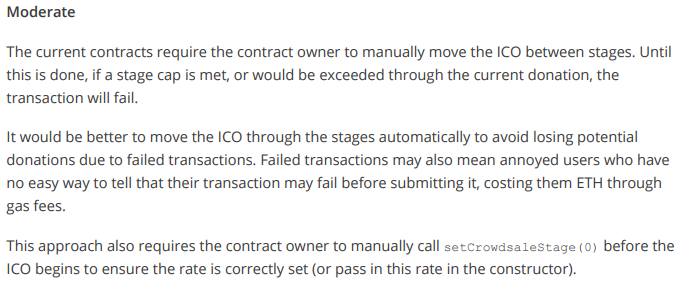


**Suggestion used**



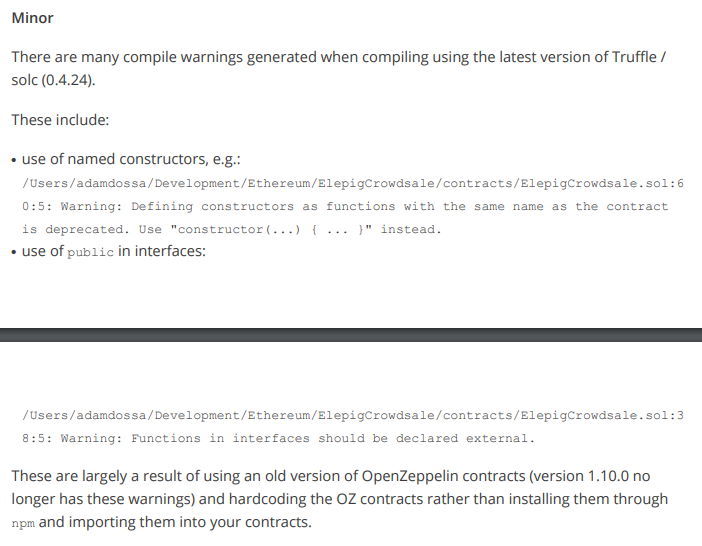
**Suggestion not implemented:**

We would like the events to be meaningful for people looking on etherscan.



**Suggestion not implemented:**

We had originally coded it in to automatically move into the next stage but having to calculate the exchange rates between the rounds was troublesome (ie, calculating ETH that pushes over the new round, calculating the EPG’s before the round change and then after it), doing this also meant we lost control of when the rounds started. The ico-website will display how many EPG’s are left in that round.



**Suggestion not implemented:**

The OpenZeppelin contracts were included within the solidity file for a couple reasons, the first being that it makes it easier to validate the contract on etherscan, and the second being, that we had to edit the standard crowdsale contract to remove the buyTokens function for KYC.

**Compilation and Deployment**

* We will be deploying the Smart Contract via https://remix.ethereum.org/ , the compile.js and deploy.js were just used for development.

**Testing**

* There are now 2 test files, one for the token and one for the crowdsale, with updated tests. These tests can be run using

**npm run test**

(this will run both test files)

* As the majority of the contracts are part of the OpenZeppelin framework, and have been thoroughly tested by the community, we only included tests for the unique functionality