# Original audit

# [**https://www.certik.com/projects/spy**](https://www.certik.com/projects/spy)

# **Medium severity issues**

* **missing update aTot after successful transaction**

**In the functions getAirdrop, we are using aTot to make the comparison to aCap at the start of the function but there is not any change in aTot after a successful airdrop transaction.**



Remediation

**Similarly, to the tokenSale function (we make update in sTot), Update aTot (airdrop total amount) after any successful airdrop transaction.**

* **Division before multiplication**

**In the function tokenSale and line 237, dividing is happening before multiplying. Let’s have a look at a simple mathematical example of this:**

**A= (10\*30\*18)/ 30= 180**

**We are now solving the same equation but performing division before multiplication.**

**A= (10/30)\* 30\*18= 179.99999**

**Almost these two terms are quite close but not the same. In the case of a solidity smart contract, it is actually going to yield 179. Therefore, performing multiplication before division mitigates rounding-off error in a smart contract.**



Remediation

**Always do your multiplication before division!**

* **Value Validation check**

**In the functions shown below, the user updates variables related to user transactions. We need to first check the argument in the function and then update the variable.**

**taxFeeUpdate(uint256 amount);**

**burnFeeUpdate(uint256 amount);**

**Text

Description automatically generated**

Remediation

**We need to check the argument in the function before making any changes to the contract. The argument should not be 0 or it should not be more than the maximum.**

**require( \_ amount >= maximumAmountAllowed , “message”);**

**require( \_ amount <= minimumAmountAllowed , “message”);**

# **Low severity issues**

* **Variables That Could Be Declared As Immutable**

**In function startAirdrop and startSale, the owner is initializing some variables, these variables should not get changed in the future. By making the change in these variables, centralization risk can happen.**

Remediation

**We advise using the constructor function and adding these variables as immutable in the contract. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.**

* **Missing Error Messages**

**The require can be used to check for conditions and throw an exception if the condition is not met.**

Remediation

**We advise adding error messages to the linked require statements.**

* **Improper Usage of public and external Type**

**public functions that are never called by the contract could be declared as external. external functions are more efficient than public functions.**

Remediation

**Consider using the external attribute for public functions that are never called within the contract.**