Total issues found: 9

Severity	Number of issues found
High	6
Medium	2
Low	1

[H - 1] Anyone can steal Maav token from Airdrop contract

Severity

High

Description

The `Airdrop.airDrop()` does not have any access control mechanism, for this reason anyone can call this function and steal all Maav token from this contract.

POC

See the `test_anyoneCanStealFromAirdrop` test from TestMaav.t.sol contract.

Recommendation

It is better to use OZ's Merkletree script & Merkleproof library. If you are planning to do it straightforward then add access control to this function so that only admins can handle the airdrop.

[H - 2] Anyone can front run & create a collection with wrong data

Severity

High

Description

The `MaAvatar.createCollection()` let anyone create a collection where a important element of collection is its `maxSupply`. It is very crucial to carefully set the data of each collections. But assume this scenario: (i) A call was made for `createCollection()` with correct data like `name`, `maxSupply` etc. (ii) When that tx was in mempool an attacker front run it with a `createCollection()` call with same `name` but 1 as `maxSupply`. As a result a collection was created with correct `name` but incorrect `maxSupply`. (iii) Now that user's tx was executed & as there is already a collection with that name the tx will revert.

POC

Run the test `test_anyoneCanFrontRunAndCreateCollectionWithWrongData` in TestMaAvatar.t.sol contract.

Recommendation

Although there is no proper protection from front running in this particular case a group discussion is needed on this, if you are intended to allow a particular group of users to create collection [this is really good idea] then have a whitelist & edit the function according to it.

[H - 3] Anyone can fill up a collection by minting max nfts

Severity

High

Description

The `MaAvatar.mintToken()` allows anyone mint any amount of token (limited to maxSupply of a collection). This is risky because anyone can mint the maximum supply limit & fill up the collection. This can be repeated as many times the attacker wants.

POC

Run the test `test_anyOneCanFillACollectionByMintingAllNFTs` in TestMaAvatar.t.sol contract.

[H - 4] `_nextCollectionId` never used

Severity

High

Description

As per the MaAvatar contract logic the collection id must be increased by 1 with each creation. But there is no check for whether the to be created collection id is following the intention. As a result any one can create a collection with any random id like 100, 1000 or 100000.

POC

Run the test `test_nextCollectionIdNeverUsed` in TestMaAvatar contract.

Recommendation

Put a check in `createCollection()` that the collectionId is equal to the `_nextCollectionId`.

[H - 5] `_nextCollectionId` was initialized incorrectly

Severity

High

Description

During deployment a collection is created by default whose id is 1. But the variable `_nextCollectionId` also initialized with 1 which will create conflict.

Recommendation

Initialize the variable with 2.

[H - 6] There is no way to reach maxSupply for Maav token if initialSupply is less than maxSupply

Severity

High

Description

As per the Maav.sol contract the maximum supply of Maav token is 1_000_000_000e18, but it allows less than that amount of token as initial supply. The problem is there is no way to increase the supply later if needed. Because only way to increase the token supply is minting new tokens, but as `_mint()` is an internal function so there is no way to call it externally. So it is not possible to reach the max supply or increase supply if needed in future.

Recommendation

Add a `mint()` with public/external visibility & access control mechanism & call the `ERC20._mint()` from there. This function will work as backup.

[M - 1] `commonElements` array is not updated properly after updatin an element

Severity

Medium

Description

When a element details is updated the element name is pushed in the respective array, depends on which category it fells into. But during creating the element it is already added in the respective array. So adding the element name during update creates duplicate.

POC

Run the test `test_updateResultsDuplicateElement` in TestMaaviMetaverse.t.sol contract.

Recommendation

This whole logic in `updateElementDetails()` is redundant:

```
uint8 nftID  = getNFTType(nftType);
        if (nftID == 1){
            commonElements.push(elementName);
        } else if (nftID == 2){
            rareElements.push(elementName);
        } else if (nftID == 3){
            epicElements.push(elementName);
        } else if (nftID == 4){
            legendaryElements.push(elementName);
        }
```

because the element name is pushed in the array during creating the element, as the element name is not updated here so there is no need to add the element name in the array again.

[M - 2] Element can be created with invalid nftID

Severity

Medium

Description

There is no revert if the nftId is 0 for any element, it is added in elemenDetail mapping. As a result the mapping may consist of incorrect elements.

POC

Run the test `test_addElementWithInvalidNftId` in TestMaaviMetaverse.t.sol contract.

Recommendation

Revert if the nftId is 0.

[L - 1] Unnecessary check may lead to DoS

Severity

Low

Description

In MaaviAirdrop contract there is a check for collection existense in the constructor:

```
require(bytes(name).length > 0 && mintedSupply >= 0, "Collection ID doesn't exist");
```

However the `mintedSupply >= 0` check is unnecessary. The `name` check is enough to identify whether the collection exists or not, but there is possibility that a collection was not involved in any mint yet, in that case, as per current implementation, the tx will be reverted due to DoS.

Recommendation

Remove the check or have a particular collection only for airdrop.