Incorrect logic in A26ZDividendDistributor::getUnclaimedAmounts() may result in divide by 0 error

Description

This line says if the token in TVS and token in Allocation is same then return 0. As per the code walkthrough the actual logic was if those tokens are not same then return 0.

Impact

This may lead to divide by 0 error if both tokens are same for any TVSs while setting up dividend. As those tokens are same this line will return 0. For that case if _totalUnclaimedAmounts set to 0 then the 'divide by 0' error will occur by this line.

Call to setUpDividends() fails if someone split his TVS and then merge before setUpDividends() getting called

Summary

The A26ZDividendDistributor::setUpTheDividends() call fails due to flaw in getTotalUnclaimedAmounts(). If someone splits his TVS & then merge again, and after that if owner calls the setUpDividends() then the call will be reverted by evm.

Root Cause

The for loop expects the nftId starts with 0.

```
uint256 len = nft.getTotalMinted();
        for (uint i; i < len;) {    // i = 0 for the 1st iteration
            (, bool isOwned) = safeOwnerOf(i);
            if (isOwned) _totalUnclaimedAmounts += getUnclaimedAmounts(i);
            unchecked {
                ++i;
            }
        }
```

But when there is more than 1 nft was minted but few of them were burned for any reason then this logic reverts.

Impact

Dividend can't be set up.

Outdated state while merging TVS result is loss of dividends

Summary

In AlignerzVesting::mergeTVS() after merging TVSs one important state is not updated, the allocationOf mapping. As dividend calculation is depended on the value of this mapping, to be specific the amounts[], as allocationOf mapping returns an Allocation struct for a NFT id, there will be a huge loss of dividend for a user.

Root Cause

As mentioned in summary, the root cause of the issue is the allocationOf mapping is not updated for the given NFT/TVS. For ex- A user did split his TVS into 5, for 5 different percentage, let say 2000 percentage each, amounts[] was 1000e18 so it will have 200e18 each. After splitting main TVS's Allocation.amounts[] will have 200e18. After split the allocationOf mapping for main nft/TVS is set to current value, i.e amounts[] will have 200e18, let say the main TVS id is 1. So allocationOf(1) returns a Allocation which has amounts[200e18]. You can see the exact line where allocationOf mapping stores the data here. Now user decided to merge those TVSs again, after merging the final Allocation will have updated amounts[] i.e amounts[200e18,200e18,200e18,200e18,200e18] for the main TVS id i.e 1. But note that the merge function i.e mergeTVS() does not update the allocationOf mapping for nft id 1, it is still the old one. When dividend is calculated the allocationOf mapping is read, see here: 1, 2, 3. As the allocationOf mapping is still outdated i.e the amounts[] of Allocation of that nft id only holds one element which is 200e18 instead of five 200e18, so only 200e18 will be counted instead of 1000e18. Means the user will loss a lot of dividend.

Impact

Loss of dividend.

TVS is not splitting with correct amount due to logic flaw in AlignerzVesting::splitTVS()

Summary

When user calls the splitTVS() to split a TVS into few percentages, then all TVSs, after splitting, should get equal amount. For ex- if I want to split my TVS for 5 percentages, i.e 2000,2000,2000,2000 & 2000, & if the amount allocated for the main TVS is 1000e18 then all 5 TVSs will get 200e18 amount at the end. But the logic flaw in this function does not let this happen.

Root Cause

Look at these lines:

```
        uint256 nftId = i == 0 ? splitNftId : nftContract.mint(msg.sender);
        if (i != 0) newNftIds[i - 1] = nftId;
        Allocation memory alloc = _computeSplitArrays(allocation,
percentage, nbOfFlows);
        NFTBelongsToBiddingProject[nftId] = isBiddingProject ? true :
false;
        Allocation storage newAlloc = isBiddingProject ?
biddingProjects[projectId].allocations[nftId] :
rewardProjects[projectId].allocations[nftId];
        _assignAllocation(newAlloc, alloc);
```

Now assume we want to split a RewardProject NFT into 5 percentages, 2000,2000,2000,2000 & 2000, just as I said in summary. Here the _computeSplitArrays() returning the splitted version of Allocation for the percentage 2000.

Now the very first iteration of the for loop i ==0, so as i == 0 the nftId will be splitNftId. After that the _computeSplitArrays() returned an Allocation which has 2000 in amounts[0]. After that this line executes:

```
Allocation storage newAlloc = isBiddingProject ?
biddingProjects[projectId].allocations[nftId] :
rewardProjects[projectId].allocations[nftId];
```

Note that here the nftId is the main TVS's id. Here we fetched the main TVS's allocation as newAlloc. Now just after that the code is calling _assignAllocation(), where the result Allocation from _computeSplitArrays() will be written on newAlloc. i.e the Allocation of the main TVS will be overwritten by the resulted Allocation of _computeSplitArrays(). Means after the _assignAllocation() call the amounts[0] of the main TVSs will be 200e18. Fine, not any issue till now. But in next all iterations this main TVSs Allocation is passed to _computeSplitArrays(), it means the splitting is happening now on 200e18, not the main amount of 1000e18. It means all further 4 TVSs will receive 40e18 as amount instead of 200e18.

Uninitialized array results in DoS

Summary

The _computeSplitArrays() returns an Allocation struct. This struct holds some arrays inside it, 5 arrays. In this function values are assigned into those arrays, but that value assign operation reverts because those arrays were never initialized.

Root Cause

The _computeSplitArrays() returns an Allocation struct.

```
function _computeSplitArrays(
        Allocation storage allocation,
        uint256 percentage,
```

```
        uint256 nbOfFlows
    )
        internal
        view
        returns (
            Allocation memory alloc
        )
    {
```

This Allocation struct has few arrays inside it:

```
struct Allocation {
        uint256[] amounts; // Amount of tokens committed for this
allocation for all flows
        uint256[] vestingPeriods; // Chosen vesting duration in seconds for
all flows
        uint256[] vestingStartTimes; // start time of the vesting for all
flows
        uint256[] claimedSeconds; // Number of seconds already claimed for
all flows
        bool[] claimedFlows; // Whether flow is claimed
        bool isClaimed; // Whether TVS is fully claimed
        IERC20 token; // The TVS token
        uint256 assignedPoolId; // Relevant for bidding projects: Id of the
Pool (poolId=0; pool#1 / poolId=1; pool#2 /...) - for reward projects it
will be 0 as default
    }
```

In the _computeSplitArrays() values are assigned to these arrays. But notice one thing that these arrays were never initialized. It means these arrays does not have any place inside it to hold a value yet. Without initializing these arrays it was tried to assign value into those:

```
            alloc.amounts[j] = (baseAmounts[j] * percentage) / BASIS_POINT;
            alloc.vestingPeriods[j] = baseVestings[j];
            alloc.vestingStartTimes[j] = baseVestingStartTimes[j];
            alloc.claimedSeconds[j] = baseClaimed[j];
            alloc.claimedFlows[j] = baseClaimedFlows[j];
```

which results in out of bound array error.

Infinite loop will result in out of gas error i.e DoS

Summary

The for loop in FeesManager::calculateFeeAndNewAmountForOneTVS() using a for loop which iterates to the infinity results in out of gas error.

Root Cause

```
for (uint256 i; i < length;) {
            feeAmount += calculateFeeAmount(feeRate, amounts[i]);
            newAmounts[i] = amounts[i] - feeAmount;
}
```

Here we can see that the i never increments, as the value of i is 0 at the time of declaration and never increments so it satisfies i < length always. Means it results in an infinite loop.

Flawed logic in FeesManager::calculateFeeAndNewAmountForOneTVS() will result DoS

Summary

The calculateFeeAndNewAmountForOneTVS() has a logic flaw in returning an array called newAmounts. When user will call the AlignerzVesting::splitTVS() the call flow goes to FeesManager::calculateFeeAndNewAmountForOneTVS(). In this function the tx reverts with FAIL: panic: array out-of-bounds access (0x32)] error.

Root Cause

The root cause of the error is how the (to be returned) array, named newAmounts , was initialized.

```
function calculateFeeAndNewAmountForOneTVS(uint256 feeRate, uint256[]
memory amounts, uint256 length) public pure returns (uint256 feeAmount,
uint256[] memory newAmounts) {
        for (uint256 i; i < length;) {
            feeAmount += calculateFeeAmount(feeRate, amounts[i]);
 @>         newAmounts[i] = amounts[i] - feeAmount;
        }
    }
```

In above function we can see that amounts[i] - feeAmount was directly stored in newAmounts[i], but note that this newAmounts[] was not initialized with any length. Means the value of amounts[i] - feeAmount trying to access such place which is not exist yet in newAmounts[].