Draft Findings

Title	Severity	Status
Wrong implementation in excessive USD calculation	CRITICAL	Open
Unexpected revert when finalizing IDO	CRITICAL	Open
Incorrect implementation in `withdrawSpareIDO`	HIGH	Open
`claimableTime` is not initialized	HIGH	Open
Incorrect rounding up in `_getPostionValue`	MEDIUM	Open
Incorrect requirement in `finalize` function	MEDIUM	Open
Unused variable and modifier in `IDOPoolAbstract`	LOW	Open
Not use `__Ownable2Step_init` instead of `__Ownable_init` in `IDOPoolAbstract`	LOW	Open
Arbitrary input decimal of ido token	LOW	Open

CRITICAL - Wrong implementation in excessive USD calculation

Positions:

- `IDOPoolAbstract.sol`#L155

Description:

The `excessiveInUSD` calculation is incorrect. In the contract, The `excessiveInUSD` is calculated as the difference between the `posInUSD` and the `exceedAlloc` multiplied by the `idoExp` divided by the `idoPrice`. Unfortunately, this calculation is incorrect.

```
function _getPostionValue(
    Position memory pos
) internal view returns (uint256 allocated, uint256 excessive) {
    //...
    if (((idoSize * idoPrice) / idoExp) >= fundedUSDValue) {
        return (buyAlloc, 0);
    } else {
        uint256 excessiveInUSD = posInUSD - ((exceedAlloc * idoExp) / idoPrice); // Incorrect
calculation
        return (
            exceedAlloc,
            (excessiveInUSD * snapshotPriceDecimals) / snapshotTokenPrice
```

```
            );
        }
    }
```

Recommendation:

The `excessiveInUSD` should be calculated as the difference between the `posInUSD` and the `exceedAlloc` multiplied by the `idoPrice` divided by the `idoExp`.

CRITICAL - Unexpected revert when finalizing IDO

Positions:

- `IDOPoolAbstract.sol` #L125

Description:

The `idoSize` state is `zero` before the `finalize()` function is called. Therefore, the function reverts with the `FudingGoalNotReached` error because the `idoSize` is always less than the `minimumFundingGoal`.

```
  function finalize() external onlyOwner notFinalized {
    if (block.timestamp < idoEndTime) revert IDONotEnded();
    else if (idoSize < minimumFundingGoal) revert FudingGoalNotReached();
    //...
  }
```

Recommendation:

The `idoSize` should be updated before minimum funding goal check.

HIGH - Incorrect implementation in `withdrawSpareIDO`

Position:

- `IDOPoolAbstract.sol` #withdrawSpareIDO()

Description:

The `withdrawSpareIDO` function is implemented to withdraw the remaining IDO tokens after the IDO is finalized. However, the ido balance is dynamic and subject to alter based on user claims. As a result, the function may not work as expected because ido balance is not enough to withdraw. Additionally, the function cannot be called if finalized is not invoked.

```
    function withdrawSpareIDO() external finalized onlyOwner {
        uint256 totalIDOGoal = (idoSize * idoPrice) / (10 ** idoDecimals);
        if (totalIDOGoal <= fundedUSDValue) revert();

        uint256 totalBought = fundedUSDValue / idoPrice * (10 ** idoDecimals);
        uint256 idoBal = IERC20Mintable(idoToken).balanceOf(address(this));
        uint256 spare = idoBal - totalBought;
        TokenTransfer._transferToken(idoToken, msg.sender, spare);
    }
```

Recommendation:

Carefully review and change the implementation of the `withdrawSpareIDO` function.

HIGH - `claimableTime` is not initialized

Position:

- `IDOPoolAbstract.sol` #L17

Description:

The `claimableTime` state is not initialized in the initialization function. Therefore, the `claimableTime` is always zero.

```
    uint256 public claimableTime;

    modifier claimable() {
        if (!isFinalized) revert NotFinalized();
        if (block.timestamp < claimableTime) revert NotClaimable();
        _;
    }
```

Recommendation:

The `claimableTime` should be initialized in the initialization function.

MEDIUM - Incorrect rounding up in `_getPostionValue`

Position:

- `IDOPoolAbstract.sol` #_getPostionValue()

Description:

In the `_getPostionValue` function, the `excessiveInUSD` is rounded up when calculated. As a result, the last user's token getting stuck.

```
   function _getPostionValue(
```

```
    Position memory pos
  ) internal view returns (uint256 allocated, uint256 excessive) {
    //...
    if (((idoSize * idoPrice) / idoExp) >= fundedUSDValue) {
      return (buyAlloc, 0);
    } else {
      uint256 excessiveInUSD = posInUSD - ((exceedAlloc * idoExp) / idoPrice); // Incorrect
rounding up
      return (
        exceedAlloc,
        (excessiveInUSD * snapshotPriceDecimals) / snapshotTokenPrice
      );
    }
  }
```

Recommendation:

Always round down the amount transferred to the user.

MEDIUM - Incorrect requirement in `finalize` function

Position:

- `IDOPoolAbstract.sol`#finalize()

Description:

The `idoSize` should be hard-coded when initializing the contract. However, the contract get the `idoSize` from contract balance when finalization. This is incorrect because the `idoSize` is dynamic. The minimum funding goal should be checked by the `fundedUSDValue` or sum of amount `buytoken` and `fytoken` instead of `idoSize`.

```
  function finalize() external onlyOwner notFinalized {
    if (block.timestamp < idoEndTime) revert IDONotEnded();
    else if (idoSize < minimumFundingGoal) revert FudingGoalNotReached(); // Incorrect
requirement
    //...
  }
```

Recommendation:

Carefully review and change the implementation of the `finalize` function.

LOW - Unused variable and modifier in `IDOPoolAbstract`

Position:

- `IDOPoolAbstract.sol`#L18

- `IDOPoolAbstract.sol`#L45

Description:

The `isClaimable` state and `notStart` modifier are not used in the contract. Therefore, these should be removed from the contract.

```
    bool public isClaimable;
    modifier notStart() {
        if (block.timestamp >= idoStartTime) revert AlreadyStarted();
        _;
    }
```

LOW - Not use `__Ownable2Step_init` instead of `__Ownable_init` in `IDOPoolAbstract`

Position:

- `IDOPoolAbstract.sol`#L77

Description:

When the contract use `Ownable2StepUpgradeable` abstract, the `__Ownable_init` should be replaced with `__Ownable2Step_init`.

```
  function __IDOPoolAbstract_init(
    //...
  ) internal onlyInitializing {
    //...
    __Ownable_init(); // Wrong initialization
  }
```

LOW - Arbitrary input decimal of ido token

Position:

- `IDOPoolAbstract.sol`#L94
- `IDOPoolAbstract.sol`#L108

Description:

The owner can input wrong decimal of `idoToken`. Therefore, the decimal should be got from contract by `decimals()` function.

```
    function setIDOToken(
        address _token,
        uint256 _idoDecimals
```

```
    ) external onlyOwner {
        //...
        idoDecimals = _idoDecimals; // Incorrect
    }

    function __IDOPoolAbstract_init_unchained(
        //...
    ) internal onlyInitializing {
        //...
        idoDecimals = idoDecimals_; // Incorrect
        //...
    }
```