[C-01] Incorrect token amount transferred during ERC20 migration

Description:

The Migrator::migrateERC20Token function transfers the wrong amount of tokenV2 to the user during an ERC20 token migration. Instead of transferring the calculated amount tokenBToRecieve, the function mistakenly transfers the input amount _amount of tokenV1. This results in users receiving fewer tokenV2 tokens than intended, violating the migration ratio defined by the price parameter. Additionally, the function updates an event with the correct tokenBToRecieve value, which demonstrates a mismatch between the transfer logic and the emitted event. This further proves the existence of the bug and may mislead developers and users who rely on the event data for verification.

Impact:

The incorrect transfer logic leads to users being under-compensated during migrations, undermining trust in the migration process and potentially resulting in financial loss for users. The mismatch between the emitted event and the actual transfer amount highlights the issue, making it harder to detect and resolve discrepancies. If unaddressed, this bug could dissuade users from participating in token migrations and damage the credibility of the smart contract.

Proof of Concept:

Here is the scenario to demonstrate this: Initial Balances: 1. User A has 6 units of tokenV1 and 0 units of tokenV2. Migrator contract holds 1000 units of tokenV2. Execution: 1. User A calls migrateERC20Token(6, tokenV1, tokenV2) with a price of 2 set by the admin. Expected Outcome: 1. User A should receive 12 units of tokenV2. The event emitted: TokenMigrationCompleted(userA, tokenv1, tokenV2, 6, 12). Actual Outcome: 1. User A receives only 6 units of tokenV2 due to the incorrect transfer logic. Event emitted misleads the user. The following code segment in the migrateERC20Token function illustrates the bug:

```
success = IERC20Upgradeable(Requirements.tokenV2).transfer(
    _msgSender(),
@>    _amount // Incorrect: Should use tokenBToRecieve instead
);
```

When a user deposits 6 units of tokenV1 for migration, the expected behavior (assuming a price of 2) is that they receive 12 units of tokenV2. However, the actual behavior is that only 6 units of tokenV2 are transferred to the user. The mapping `tokensMigrated[_token2] += tokenBToRecieve` here correctly logs it;

Test Case to prove:

```
function test_tokenB_received_is_correct_migration_amount() public {
    vm.startPrank(user);

    uint initialV1Balance = tokenV1.balanceOf(user);
    uint initialV2Balance = tokenV2.balanceOf(user);
    console.log("initialv1Balance: ", initialV1Balance);
    console.log("initialV2Balance: ", initialV2Balance);
    uint migrationAmount = 6; // Migrate 6 tokens

    // Approve migrator to spend tokens
    tokenV1.approve(address(migrator), migrationAmount);

    // Perform migration
    migrator.migrateERC20Token(
        migrationAmount,
        address(tokenV1),
        address(tokenV2)
    );

    // Check final balances
    uint finalV1Balance = tokenV1.balanceOf(user);
    uint finalV2Balance = tokenV2.balanceOf(user);
    console.log("finalv1balance: ", finalV1Balance);
    console.log("finalV2Balance: ", finalV2Balance);

    // Calculate expected amounts
    uint expectedV1Decrease = migrationAmount;
    uint expectedV2Increase = migrationAmount * 2; // Should get 2x tokens due to
price=2

    // Verify V1 token decrease is correct
    assertEq(
        initialV1Balance - finalV1Balance,
        expectedV1Decrease,
        "V1 token decrease incorrect"
    );

    // This assertion will fail because of the bug
    // User receives migrationAmount instead of migrationAmount * 2
    assertEq(
        finalV2Balance - initialV2Balance,
        expectedV2Increase,
        "V2 token increase incorrect - Bug detected!"
    );
    vm.stopPrank();
}
```

The output of the test is seen below:

```
$ forge test --match-test test_tokenB_received_is_correct_migration_amount -vv
[:] Compiling...
[⠃] Compiling 3 files with Solc 0.8.27
[⠊] Solc 0.8.27 finished in 4.77s
Compiler run successful!

Ran 1 test for test/PoCs/MigratorPoc.sol:TokenBTransferAmountTest
[FAIL: V2 token increase incorrect - Bug detected!: 6 != 12]
```

```
test_tokenB_received_is_correct_migration_amount() (gas: 177947)
Logs:
  initialv1Balance:  6
  initialV2Balance:  0
  finalv1balance:  0
  finalV2Balance:  12

Suite result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 747.23ms (94.35ms
CPU time)

Ran 1 test suite in 1.13s (747.23ms CPU time): 0 tests passed, 1 failed, 0 skipped
(1 total tests)

Failing tests:
Encountered 1 failing test in test/PoCs/MigratorPoc.sol:TokenBTransferAmountTest
[FAIL: V2 token increase incorrect - Bug detected!: 4 != 8]
test_tokenB_received_is_correct_migration_amount() (gas: 177947)
```

it fails because the _amount the user inputted is being sent instead of `tokenBtoReceive` The test confirms: The actual transfer is incorrect (6 units of tokenV2 instead of 12). The emitted event misleads by stating the correct amount (12).

Mitigation:

Replace the incorrect `_amount` variable with `tokenBToRecieve` in the transfer call:

~~~
success = IERC20Upgradeable(Requirements.tokenV2).transfer(
    _msgSender(),
-     _amount
);
```diff
success = IERC20Upgradeable(Requirements.tokenV2).transfer(
    _msgSender(),
+     tokenBToRecieve // Correct: Transfer the calculated tokenBToRecieve amount
);
~~~

Tools Used:

Foundry, Manual Review

[M-01] Misconfigured authorization in `setFreeParticipant` function prevents controllers from managing free participants

Description:

The `setFreeParticipant` function in all contracts scoped in `land-nfts` and `land-nfts-v2` iis ntended to allow both the owner and designated `freeParticipantControllers` to modify the `freeParticipant` mapping. However, the function is restricted by the onlyOwner modifier, allowing only the contract owner to execute it. This implementation prevents `freeParticipantControllers` from fulfilling their intended role, resulting in an ineffective delegation of authority.

Impact:

The current implementation creates the following issues: 1. Delegation Failure: The `freeParticipantControllers` mapping is non-functional because controllers cannot set free participants. 2. Unnecessary Gas Costs: Maintaining an ineffective `freeParticipantControllers` mapping introduces redundant storage costs.

Proof of Concept:

Current implementation:

```
function setFreeParticipant(address participant, bool free) public onlyOwner {
    freeParticipant[participant] = free;
}
```

Issue:

- The `onlyOwner` modifier restricts access to the owner only, making the `freeParticipantControllers` mapping unusable.

Expected Behavior:

- Both the owner and addresses in `freeParticipantControllers` should be able to execute the function.

Recommended Mitigation:

Update the `setFreeParticipant` function to allow both owner and free participant controllers to set free participants

```
- function setFreeParticipant(address participant, bool free) public onlyOwner {

+ function setFreeParticipant(address participant, bool free) external {
+        require(freeParticipantControllers[msg.sender] || msg.sender == owner(),
"Not authorized");
        freeParticipants[participant] = free;
}
```

Tools Used

Manual Review, Foundry

[L-01] Poor Documentation/Natspec in Mutiple Contracts

Description:

The follwing contracts lacks sufficient Documentation and NatSpec comments to explain the purpose and functionality of its components.

1. `src/COA-Contracts/land-nfts/PLOT.sol`
2. `src/COA-Contracts/land-nfts/ACRE.sol`
3. `src/COA-Contracts/land-nfts/YARD.sol`
4. `src/COA-Contracts/land-nfts-v2/ACRE.sol`
5. `src/COA-Contracts/land-nfts-v2/PLOT.sol`
6. `src/COA-Contracts/land-nfts-v2/YARD.sol`
7. `src/Migrator.sol`

Impact:

1. Increases the risk of misinterpretation during further development or auditing processes.
2. Makes integration with external systems and use by third parties less efficient due to lack of detailed documentation
3. Reduces code readability and maintainability.

[L-02] - Gas Optimization: Unnecessary storage reads from `_currentBatch` struct in all contracts scoped in `land-nfts` and `land-nfts-v2`

Description:

The `_currentBatch` struct is accessed directly from storage multiple times in the `mint` function in all contracts scoped in `land-nfts` and `land-nfts-v2`. This results in repeated and unnecessary storage reads, which are more expensive in terms of gas compared to memory access.

Impact:

Repeated storage reads increase the gas cost of the mint function and other functions that frequently access `_currentBatch`. This inefficiency can lead to higher transaction costs for users, particularly during minting events with high demand.

Proof of Concept:

Recommended Mitigation:

Use a memory copy of the _currentBatch struct to reduce storage reads.

```diff
function mint(uint256 quantity) external {
+    Batch memory batch = _currentBatch;
-    require(_currentBatch.quantity > 0, "No more tokens left to mint");
-    require(_currentBatch.active, "Current Batch is not active");
+    require(batch.quantity > 0, "No tokens left in batch");
+    require(batch.active, "Batch is not active");
+    require(quantity > 0, "Quantity must be greater than zero");
+    require(quantity <= maxBuyAmount || msg.sender == owner(), "Exceeds max buy limit");
```

```
    if (!freeParticipants[msg.sender]) {
        require(_pay(msg.sender, quantity), "Must Pay Minting fee");
    }

-   _currentBatch.quantity = (_currentBatch.quantity - quantity);
+    batch.quantity -= quantity;

+    // Write the updated memory struct back to storage
+    _currentBatch = batch;

    // Mint tokens
    _safeMint(msg.sender, quantity);
}
```

[L-05] Inconsistenrt free mint handling

Description:

Impact:

Proof of Concept:

Recommended Mitigation:

Tools used:

[L-03] Unused `_tax` Functionality in all contracts scoped in `land-nfts` and `land-nfts-v2`

Description:

all `land-nfts` and `land-nfts-v2` contracts includes redundant functionality in the form of the `_tax` function which is defined but never used in the contract. It performs a token transfer for a fixed transaction fee `_txFeeAmount` but does not serve any purpose in the current implementation.

Impact

Redundant functions increase the contract size unnecessarily, leading to higher gas costs for deployment and potential confusion for maintainers or auditors. They also create additional surface area for misuse or inconsistencies, especially during upgrades or audits, which could lead to security risks.

Proof of Concept:

```
function _tax(address payee) internal virtual returns (bool) {
    IERC20 token = IERC20(_paymentToken);
    token.transferFrom(payee, _feeCollector, _txFeeAmount);
```

```
        return true;
    }
```

This function is never called anywhere in the contract, making it dead code. It adds to the contract size without serving a purpose.

Recommended Mitigation:

1. Remove the _tax function entirely unless there is a specific use case for it. the contract will be cleaner, more efficient, and easier to maintain.

```diff
-    function _tax(address payee) internal virtual returns (bool) {
-        IERC20 token = IERC20(_paymentToken);
-        token.transferFrom(payee, _feeCollector, _txFeeAmount);
-        return true;
-    }
```

[L-04] - Informational: Non-Standard Code Layout in `src/COA-Contracts/land-nfts/PLOT.sol`

Description:

The contract does not follow Solidity's recommended code layout and organization practices, as outlined in the Solidity Style Guide. This deviation from best practices makes the code harder to read, understand, and maintain, especially as the contract evolves or expands.

Recommended Mitigation:

Contract should be layed out in the following order according to the official Solidity Style Guide

1. pragma
2. Import statements
3. Events
4. Errors
5. Interfaces
6. Libraries
7. Contracts

Inside each contract, library or interface, use the following order:

1. Type declarations
2. State Variables
3. Events
4. Errors
5. Modifiers
6. External and Public view functions
7. Internal and private view functions