Protocol Audit Report

Version 1.0

Vesko.io

December 12, 2024

2024-12-SECONDSWAP

Vesko.io

Prepared by: Vesko Lead Auditors: - Veselin Vachkov

Issues Found

Severity	Number of Issues Found
Low	9
Informational	1
Total	10

Findings

Low

[L-1] Allocation Tracking The function `unlistVesting` in the `SecondSwap_VestingManager` contract includes the following line:

```
allocations[seller][plan].sold -= amount;
```

```
/*
* @notice Tracks bought and sold amounts for each user's vesting
    allocation
* @param bought Amount of tokens bought by the user
* @param sold Amount of tokens sold by the user
```

```
*/
    struct Allocation {
        uint256 bought;
        uint256 sold;
    }
```

This line decreases the sold value in the `Allocation` struct for the specified seller and vesting plan. The sold field is intended to track the cumulative total of all tokens listed/sold by a user. Reducing this value implies that the user has sold fewer tokens than they actually have, which misrepresents their transaction history.

Recommended mitigation

The contract should avoid modifying `sold` when tokens are unlisted. Instead, it should track active listings separately. Introduce Active Listings Tracking Add a new mapping to track the number of tokens actively listed in the marketplace:

```
mapping(address => mapping(address => uint256)) public activeListings;
```

[L-2] Underflow Prevention In the function `unlistVesting` in the `SecondSwap_VestingManager` contract the following hapends:

```
function unlistVesting(address seller, address plan, uint256 amount)
    external onlyMarketplace {
        allocations[seller][plan].sold -= amount;
        SecondSwap_Vesting(plan).transferVesting(address(this), seller,
            amount);
    }
```

If `amount` is greater than `allocations[seller][plan].sold`, an underflow will occur. While Solidity 0.8.0 and above introduces automatic underflow and overflow checks, which cause a revert in such situations, it is still a good practice to explicitly ensure this condition doesn't arise using a require statement.

Add a require statement to verify that allocations[seller][plan].sold is greater than or equal to amount before performing the subtraction:

```
require(allocations[seller][plan].sold >= amount, "Underflow:
    insufficient allocation sold");
```

[L-3] Redundant Checks Unnecessary Check in `setMaxWhitelist` Function in the `SecondSwap_Whitelist` contract

```
require(
    _maxWhitelist > totalWhitelist,
    "SS_Whitelist: amount cannot be lesser that the current whitelist
       amount"
);
```

This check is redundant because the preceding condition already ensures that `_maxWhitelist` is greater than `maxWhitelist`, which inherently satisfies the condition of being greater than `totalWhitelist` since `totalWhitelist` cannot exceed `maxWhitelist` under the contract's logic.

Implications

Having redundant checks increases the gas cost of the transaction and makes the codebase harder to maintain without providing additional safety or functionality.

Recommended Mitigation

Remove the redundant condition to streamline the function

By removing the unnecessary check, the function remains secure while improving efficiency and readability.

[L-4] Excessive Default Penalty Fee The contract sets the `penaltyFee` in `SecondSwap_MarketplaceSettin` to `10 ether` during initialization:

```
penaltyFee = 10 ether; // 10 ether == 10000000000000000000 wei
```

Implications

A default penalty fee of 10 ether is unusually high and may discourage user engagement with the marketplace. Also for smaller transactions, this fee could exceed the value of the listed assets, leading to an unfair and impractical penalty.

Recommended Mitigation

Set a reasonable default penalty fee that aligns with typical transaction values on the platform. For instance:

```
penaltyFee = 0.1 ether; // 0.1 ether == 100000000000000000 wei
```

[L-5] Lack of Penalty Fee Cap The `setPenaltyFee` function in `SecondSwap_MarketplaceSetting` allows the admin to set the `penaltyFee` to any arbitrary value:

```
function setPenaltyFee(uint256 _amount) external onlyAdmin {
    require(_amount > 0, "SS_Marketplace_Settings: Penalty fee cannot
       be less than 0");
    penaltyFee = _amount;
    emit DefaultPenaltyFeeUpdated(_amount);
}
```

Implications

Without an upper limit, the penalty fee could be set to an excessively high amount, enabling harmful activities such as indirectly locking user funds or deterring asset listings. This creates an unfair environment for users.

Recommended Mitigation

Introduce an upper limit to the penalty fee, ensuring it remains within a reasonable range:

```
uint256 public constant MAX_PENALTY_FEE = 5 ether; // Example cap

function setPenaltyFee(uint256 _amount) external onlyAdmin {
    require(_amount > 0, "SS_Marketplace_Settings: Penalty fee cannot
       be less than 0");
    require(_amount <= MAX_PENALTY_FEE, "SS_Marketplace_Settings:
       Penalty fee exceeds maximum limit");
    penaltyFee = _amount;
    emit DefaultPenaltyFeeUpdated(_amount);
}
```

[L-6] Excessive Maximum Buyer and Seller Fees The `SecondSwap_MarketplaceSetting` contract allows the admin to set buyer and seller fees up to 50%:

```
require(_amount <= 5000, "SS_Marketplace_Settings: Buyer fee cannot be
   more than 50%");
require(_amount <= 5000, "SS_Marketplace_Settings: Seller fee cannot be
    more than 50%");
```

Implications

A maximum fee of 50% is exceptionally high and could discourage participation in the marketplace. This level of fee is unusual for legitimate marketplaces and may be perceived as exploitative by users.

Recommended Mitigation

Set more reasonable maximum fee limits, such as 5-10%:

```
uint256 public constant MAX_FEE = 1000; // 10% in basis points

require(_amount <= MAX_FEE, "SS_Marketplace_Settings: Fee cannot be
    more than 10%");
```

[L-7] Referral Fee of 100% is excessive The `SecondSwap_MarketplaceSetting` contract permits referral fees up to 100%, which is excessive and unrealistic:

```
require(_percentage <= 10000, "SS_Marketplace_Settings: Percentage
    cannot be more than 100%");
```

Implications

Allowing referral fees of up to 100% effectively redirects the entire fee revenue to the referral program, leaving the platform without income from fees. This could lead to operational and financial challenges.

Recommended Mitigation

Set a more practical upper limit for referral fees, such as 10-20%:

```
uint256 public constant MAX_REFERRAL_FEE = 2000; // 20% in basis points

require(_percentage <= MAX_REFERRAL_FEE, "SS_Marketplace_Settings:
    Percentage cannot exceed 20%");
```

[L-8] Lack of Restrictions on Marketplace Freeze The contract allows the admin to freeze the marketplace by calling `setMarketplaceStatus` in `SecondSwap_MarketplaceSetting`:

```
function setMarketplaceStatus(bool _status) external onlyAdmin {
    isMarketplaceFreeze = _status;
```

```
        emit MarketplaceStatusUpdated(_status);
    }
```

Implications

There are no safeguards to prevent misuse of the marketplace freeze functionality. A malicious or compromised admin could permanently freeze the marketplace, disrupting all activity.

Recommended Mitigation

Introduce limitations on the freeze functionality, such as requiring multi-signature approvals or limiting the freeze duration time.

[L-9] Incompatibility with Functions Requiring Arguments The abi.encodeWithSelector(selector) encodes the selector but no arguments for the function. This will work only if the function being checked has no arguments. (`SecondSwap_Marketplace`)

```
function doesFunctionExist(address target, string memory
    functionSignature) public view returns (bool) {
        bytes4 selector = bytes4(keccak256(bytes(functionSignature)));
        (bool success, ) = target.staticcall(abi.encodeWithSelector(
            selector));
        return success;
}
```

If the function signature involves arguments (e.g., transfer(address,uint256)), this will not work correctly because the function is expected to have arguments, and the staticcall will fail (even if the function exists, but with different arguments).

Informational

[I-1] Issue: Wrong error message The error message in both `setBuyerFee` and `setSellerFee` functions, is misleading as it suggests that the fee cannot be less than 0 but it actualy can be -1. This could cause confusion for the users of the contract.

```
require(_fee >= -1 && _fee <= 5000, "SS_VestingManager: Buyer Fee
    cannot be less than 0");
```

```
require(_fee >= -1 && _fee <= 5000, "SS_VestingManager: Seller fee
    cannot be less than 0");
```