SECONDSWAP
Security Review

DECEMBER, 2024

www.chaindefenders.xyz
https://x.com/ChDefendersEth

Lead Auditors

PeterSR

0x539.eth

Table of Contents

Protocol Summary

SecondSwap addresses the need for a secondary market where locked tokens can be traded offering Sellers an opportunity to get liquidity, at a discount or premium, while allowing opportunistic or higher conviction Buyers to capitalize on future upside. In addition, Token Issuers have more control and are incentivised to facilitate transactions because they benefit by earning fees for every successful transaction.

Disclaimer

The ChainDefenders team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

Likelihood/Impact	High	Medium	Low
High	H	H/M	M
Medium	H/M	M	M/L
Low	M	M/L	L

Audit Details

Scope

Id	Files in scope
1	SecondSwap_Marketplace.sol
2	SecondSwap_MarketplaceSetting.sol
3	SecondSwap_StepVesting.sol
4	SecondSwap_VestingDeployer.sol
5	SecondSwap_VestingManager.sol
6	SecondSwap_Whitelist.sol
7	SecondSwap_WhitelistDeployer.sol

Roles

Id	Roles
1	2S Admin
2	Token Issuer

Executive Summary

Issues found

Severity	Count	Description
High	1	Critical vulnerabilities
Medium	4	Significant risks
Low	0	Minor issues with low impact
Informational	0	Best practices or suggestions
Gas	0	Optimization opportunities

Findings

High

[HIGH-01] Wrong Logic In `transferVesting`

Finding description and impact

There is an error in the `SecondSwap_StepVesting::transferVesting`.

```
function transferVesting(address _grantor, address _beneficiary, uint256 _amount) external {
        require(
            msg.sender == tokenIssuer || msg.sender == manager || msg.sender == vestingDeployer,
            "SS_StepVesting: unauthorized"
        );
        require(_beneficiary != address(0), "SS_StepVesting:
```

```
beneficiary is zero");
        require(_amount > 0, "SS_StepVesting: amount is zero");
        Vesting storage grantorVesting = _vestings[_grantor];
        require(
            grantorVesting.totalAmount - grantorVesting.amountClaimed >= _amount,
            "SS_StepVesting: insufficient balance"
        ); // 3.8. Claimed amount not checked in transferVesting function

        grantorVesting.totalAmount -= _amount;
        grantorVesting.releaseRate = grantorVesting.totalAmount / numOfSteps;
```

If we take at this line `grantorVesting.releaseRate = grantorVesting.totalAmount / numOfSteps;`, releaseRate is calculated using the totalAmount and number of steps. But the problem is that if the `grantor` already claimed some of the amount. In that case this line will result into a wrong computation. User will be able to claim more of the tokens earlier.

Proof of Concept

Let's have the following situation:

1. User A has `totalAmount` of 100 and `amountClaimed` of 20. The total steps are 5 and user claimed just for the first step.
2. `transferVesting` is called as grantor is passed User A and `_amount` = 20.
3. `totalAmount` is equal to 80 and `releaseRate` = 80 / 5 = 16.
4. This means that User A can claim 64 in the next 4 steps.
5. So he can withdraw 48 in the next three steps and in the last step 12, which will break the linear vesting.

Recommended mitigation steps

Refactor this line to be:

```
+ if(numOfSteps > grantorVesting.stepsClaimed) {
+    grantorVesting.releaseRate = grantorVesting.totalAmount - grantorVesting.amountClaimed / numOfSteps - grantorVesting.stepsClaimed;
+ } else {
+    grantorVesting.releaseRate = 0;
+ }
- grantorVesting.releaseRate = grantorVesting.totalAmount / numOfSteps;
```

Medium

[MEDIUM-01] One Token Can Have Multiple Owners

Finding description and impact

Currently the method `SecondSwap_VestingDeployer::setTokenOwner` could be called by the protocol admin to set a token owner. The problem with the current implementation is one token could have more than owners. This is due to the fact that the mapping is owner to token, not token owner. And protocol owner can't easily check if a token is already having an owner.

```
function setTokenOwner(address token, address _owner) external onlyAdmin {
        require(_tokenOwner[_owner] == address(0), "SS_VestingDeployer: Existing token have owner");
        _tokenOwner[_owner] = token;
    }
```

If there are two owners of one token, if the first one is malicious owner(currently there is no function to revoke an owner of a token), they could manipulate `transferVesting`:

```
function transferVesting(
        address _grantor,
        address _beneficiary,
        uint256 _amount,
        address _stepVesting,
        string memory _transactionId
    ) external {
        require(
            _tokenOwner[msg.sender] == address(SecondSwap_StepVesting(_stepVesting).token()),
            "SS_VestingDeployer: caller is not the token owner"
        ); // 3.2. Arbitrary transfer of vesting
        SecondSwap_StepVesting(_stepVesting).transferVesting(_grantor, _beneficiary, _amount);
        emit VestingTransferred(_grantor, _beneficiary, _amount, _stepVesting, _transactionId);
    }
```

They can manipulate it due to the fact that the check here checks does `msg.sender` is authorized for this token, but the malicious owner could change the balance of users and move all of the funds to his account, so he can benefit from this.

As discussed with the sponsor, there should be only one token owner.

Proof of Concept

Let's have the following scenario:

1. Admin sets for Token A owner to User A.
2. After some time due to mistake or because User A acts maliciously Admin sets User B as owner.
3. Now there are two owners of the same token. Even if the first one is not malicious he can make changes to vestings created by the other one.

Recommended mitigation steps

First the function to set token owner should be refactored to be setting ONE owner per token and the mapping should be changed.

And there should be a second function to revoke token owner.

[MEDIUM-02] `buyFee` And `sellFee` Should Be Known Before Purchase

Finding description and impact

The platform allows the `buyFee` and `sellFee` parameters for a vesting plan to be modified after a listing is created. This creates a significant issue in terms of transparency and predictability for users engaging in transactions.

Impact on Users:

1. Uncertainty for Buyers and Sellers:
   Both buyers and sellers cannot reliably determine the exact fees associated with a transaction until the `spotPurchase` function is executed. This lack of transparency diminishes user confidence in the platform.
2. Financial Discrepancies for Sellers:
   Sellers may receive less revenue than anticipated if the seller fee (`sellFee`) is increased after the listing is created. This directly impacts their earnings and could lead to dissatisfaction or distrust in the platform's fee structure.

Proof of Concept

Let's have the following scenario:

1. User A creates listing, currently the fees for this vesting plan are 1000 and 1000.
2. After some period these fees are changed to 5000 for the seller fee and 1000 for buyer fee.
3. User A will receive less money, because the fee is bigger.

Recommended mitigation steps

Consider two additional parameters to be added for the listing:

```
(uint256 bfee, uint256 sfee) = _getFees(_vestingPlan);
listings[_vestingPlan][listingId] = Listing({
        seller: msg.sender,
        total: _amount,
        balance: _amount,
        pricePerUnit: _price,
        listingType: _listingType,
        discountType: _discountType,
        discountPct: _discountPct,
        listTime: block.timestamp,
        whitelist: whitelistAddress,
        currency: _currency,
        minPurchaseAmt: _minPurchaseAmt,
        status: Status.LIST,
        vestingPlan: _vestingPlan,
+       buyerFee: bfee,
+       sellerFee: sfee
    });
    emit Listed(_vestingPlan, listingId);
```

```
}
```

Also make the changes to the `Listing` struct and `spotPurchase` to use the correct fees.

[MEDIUM-03] `referralFeeCost` Could Be The Whole `buyerFee` Or Bigger Part

Finding description and impact

In the current implementation there is potential flaw how the `refferalFeeCost` is calculated.

```
referralFeeCost = 0;
if (_referral != address(0) && listing.whitelist == address(0)) {
    referralFeeCost =
        buyerFeeTotal -
        (baseAmount * bfee * IMarketplaceSetting(marketplaceSetting).referralFee()) /
        (BASE * BASE);
}
```

Due to the current implementation smaller the refferalFee, more the refferer will receive.

Proof of Concept

Let's look at the following scenario:

1. `refferalFee` is 1000 and `bfee` is 1000 and baseAmount is `1000` the protocol supports low decimals tokens, so this could be equal to 10 GUSD.
2. `buyerFeeTotal = 1000 * 1000 / 10000 = 100`
3. `baseAmount * bfee * IMarketplaceSetting(marketplaceSetting).referralFee()) /(BASE * BASE) = 1000 * 1000 * 1000 / 10000 * 10000 = 10.`
4. In that case `refferalFeeCost` is equal to 90

And another scenario:

1. `refferalFee` is 1 and `bfee` is 1000 and baseAmount is `1000` the protocol supports low decimals tokens, so this could be equal to 10 GUSD.

2. `buyerFeeTotal = 1000 * 1000 / 10000 = 100`
3. `baseAmount * bfee * IMarketplaceSetting(marketplaceSetting).referralFee()) /(BASE * BASE) = 1000 * 1000 * 1 / 10000 * 10000 = 0.`
4. In that case `refferalFeeCost` is equal to 100

Refferal fee is paid off-chain and most of the `totalBuyerFee` will be paid to the refferer.

Recommended mitigation steps

Refactor the method to calculate the `refferalFee` to be part of the `buyerFee`:

```
referralFeeCost = 0;
if (_referral != address(0) && listing.whitelist == address(0)) {
    referralFeeCost =
-         buyerFeeTotal -
        (baseAmount * bfee * IMarketplaceSetting(marketplaceSetting).referralFee()) /
                (BASE * BASE);
}
```

[MEDIUM-04] `setManager` Should Be Invoked Only If There Are No Active Listing

Finding Description and Impact

The `SecondSwap_MarketplaceSetting::setManager` function can be called even when there are active listings associated with the current `vestingManager`. This means that all vestings that are currently held by the VestingManager will be lost, if the vesting manager is changed. On top of that all information regarding the bought and sold by user is also lost, which means that user can sell more than the `maxSellPercent`.

This behavior results in the following issues:

- Data Loss: The `allocations` mapping (`mapping(address => mapping(address => Allocation)) public allocations;`) tied to the current `vestingManager` is lost when the manager is replaced.
- System Disruption: Active listings remain unfinished, and the vestings which are listed, but not transferred can't be claimed because they will be stucked in the VestingManager.

Proof of Concept

1. The `vestingManager` is set to `managerA`, and users have active allocations in the `allocations` mapping.
2. The admin invokes `setManager(newManagerB)`.
3. The `vestingManager` is updated to `newManagerB`, and all data associated with `managerA` in the `allocations` mapping becomes inaccessible.

```
function setManager(address _vestingManager) external onlyAdmin {
    require(_vestingManager != address(0), "SS_Marketplace_Settings: Cannot be zero address");
    require(
        _vestingManager != address(vestingManager),
        "SS_Marketplace_Settings: Cannot be the same vestingManager address"
    );
    vestingManager = _vestingManager; // Old manager's data is lost
}
```

Recommended Mitigation Steps

Implement functionality to force users to complete or withdraw active listings before changing the manager or announce something like a grace period in which all users should finalized their `listing` etiher unlist or transfer them.