SecondSwap Audit Report

Zach Katz

May 22, 2025

SecondSwap Audit Report

Zach Katz

May 22, 2025

Prepared by: Zach Katz

Table of Contents

Protocol Summary

SecondSwap is a secondary market for trading locked/vesting tokens. It gives owners of vesting tokens the opportunity to sell their locked tokens and gain

instant liquidity, while opportunistic buyers can purchase these locked tokens at a discount.

Disclaimer

Zach 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 him 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

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

Executive Summary

Issues found

Severity	Number of issues found
High	3
Medium	1
Low	1
Info	2
Total	7

Findings

High

[H-1] Admin can change `SecondSwap_VestingDeployer::_tokenIssuer` to claim all vestings

Description: The SecondSwap admin can at any point change the address of the `_tokenIssuer`, which grants the ability to transfer all vestings to an arbitrary address.

Impact: This significantly reduces the trustworthiness and security of the protocol, as all funds vested in SecondSwap are vulnerable to "rugpull" behavior, where the admin can hijack the token issuer and transfer out all of the funds to any address.

Proof of Concept:

```
function testAdminCanChangeTokenOwnerAndTransferVestings() public {
        vm.prank(admin);
        vestingDeployer.setTokenOwner(address(token), tokenIssuer);

        vm.startPrank(tokenIssuer);
        vm.recordLogs();
        vestingDeployer.deployVesting(
            address(token),
            startTime,
            endTime,
            numOfSteps,
            "testId"
        );
        Vm.Log[] memory entries = vm.getRecordedLogs();
        address vestingAddr = abi.decode(entries[0].data, (address));
        token.approve(vestingAddr, 1000000000000000000);
        vestingDeployer.createVesting(seller, 1000000000000000000, vestingAddr);
        vm.stopPrank();

        vm.prank(admin);
        vestingDeployer.setTokenOwner(address(token), admin);

        console.log("token balance of seller", token.balanceOf(seller));

        vm.prank(admin);
        vestingDeployer.transferVesting(
            seller,
            rugPullAddress,
            1000000000000000000,
            vestingAddr,
            "testId"
        );

        vm.warp(block.timestamp + 366 days);
        vm.prank(rugPullAddress);
        SecondSwap_StepVesting(vestingAddr).claim();

        assertEq(token.balanceOf(rugPullAddress), 10000000000000000000);
    }
```

Recommended Mitigation: Only allow the tokenOwner to be set once, or modified by the existing tokenOwner – remove admin privileges to assign this role after its initialization.

[H-2] Incorrect calculation of releaseRate in `SecondSwap_StepVesting::transferVesting` allows beneficiaries to claim more than they should be allowed.

Description: The calculation of releaseRate does not accurately reflect what the new release rate should be in situations where the beneficiary has already claimed some of their vestings.

Impact: Allows beneficiaries to claim some of their vestings earlier than they should be able to.

Proof of Concept:

```
function testIncorrectReleaseRateCalculationOnTransfer() public {
        address addr2 = makeAddr("addr2");

        vm.prank(admin);
        vestingDeployer.setTokenOwner(address(token), tokenIssuer);

        vm.startPrank(tokenIssuer);
        vm.recordLogs();
        vestingDeployer.deployVesting(
            address(token),
            startTime,
            endTime,
            numOfSteps,
            "testId"
        );
        Vm.Log[] memory entries = vm.getRecordedLogs();
        address vestingAddr = abi.decode(entries[0].data, (address));
        token.approve(vestingAddr, 1900000000000000000);
        vestingDeployer.createVesting(seller, 1000000000000000000, vestingAddr);
        vm.stopPrank();

        token.mint(vestingAddr, 1000000000000000000);

        vm.warp(startTime + 50 days);
        vm.prank(seller);
        SecondSwap_StepVesting(vestingAddr).claim();

        vm.prank(tokenIssuer);
        vestingDeployer.transferVesting(
            seller,
            addr2,
```

```
            1000000000000000000 / 10,
            vestingAddr,
            "test"
        );

        vm.warp(startTime + 91 days);
        vm.prank(seller);
        SecondSwap_StepVesting(vestingAddr).claim();

        // Claimed 5e18 already, and transfered 1e18.
        // 4e18 vestings remaining / 5 steps left = 0.8e18 per step
        // 4 steps * 0.8e18 = 3.2e18
        // 3.2e18 + 5e18(already claimed) = 8.2e18 (should be the balance of seller after se

        // Incorrect release rate calculation results in more than this being claimed.
        assert(token.balanceOf(seller) > 820000000000000000);
    }
```

Recommended Mitigation:

```
- grantorVesting.releaseRate = grantorVesting.totalAmount / numOfSteps;
+ grantorVesting.releaseRate = grantorVesting.totalAmount - grantorVesting.amountClaimed
+                             / numOfSteps - grantorVesting.stepsClaimed;
```

[H-3] Currency transfers based on token decimals, not currency decimals

Description: In `SecondSwap_Marketplace::_handleTransfers`, the baseAmount is calculated using decimals from the vested token, not from the currency with which value is being exchanged.

Impact: Transfers from the seller, to the buyer, and to the fee collector can be of excessive value – if, for example, the currency of exchange is USDT which has only 6 decimals.

Recommended Mitigation: Calculate baseAmount using currency token address, not vesting token address.

[H-3] Currency transfers may not go through and will not revert.

Description: In `SecondSwap_Marketplace::_handleTransfers`, if the currency being transfered is USDT and the value of attempted transfer is larger than what is approved or held, the transfer will not go through nor will the transaction revert.

Impact: Intended currency transfers will not go through and the transaction will still not revert

Recommended Mitigation: Check if the transfers are returning false to catch USDT transfer failure.

Medium

[M-1] Anyone can whitelist themselves

Description: `SecondSwap_Whitelist::whitelistAddress` has no access controls on it

Impact: Anyone can whitelist themselves to interact with a listing

Recommended Mitigation: Implement access controls for the function

Low

[L-1] Penalty fee initialized to 10 ether

Description: `SecondSwap_MarketingplaceSetting` sets the penalty fee to 10 ether by default.

Impact: If the value is not reset to a more realistic penalty fee, early delisting will either revert or charge an exorbatant penalty fee.

Recommended Mitigation: Choose a more reasonable default penalty fee to initailize to.

Informational

[I-1] `referralFeeCost` is never used

Description: In `SecondSwap_Marketplace::_handleTransfers` a `referralFeeCost` is calculated but never used.

[I-2] Initializable contracts can be frontrun

Description: `SecondSwap_Marketplace` and `SecondSwap_VestingManager` are upgradeable smart contracts. Be sure to initialize them at the same time as deployment or risk being front run and losing ownership of the protocol.