Steadefi

Security Testing and Assessment

May 1, 2023

Prepared for Steadefi

Narya.ai

Table of Content

Summary

Overview

From April 10, 2023, to April 17, 2023, Steadefi engaged Narya.ai to test the security of its TokenManager and Farms contracts in the GitHub repository: https://github.com/steadefi/steadefi-contracts (commit 82835ba2240beef856c56e8e4a5e241357b02db0).

TokenManager and Farms contracts are the masterChef and farming contracts for Steadefi's native token STEADY. They have the following features:

- Converting STEADY to esSTEADY (the non-transferable version of STEADY).
- Staking STEADY and claiming rewards in STEADY and esSTEADY.
- Vesting for esSTEADY tokens.

Project Scope

We reviewed and tested the smart contracts including STEADY, esSTEADY, TokenManager, and Farms. There are other security-critical components of Steadefi, such as off-chain services, the web front end, and all the other smart contracts. They are not included in the scope of this security assessment. We recommend a further review of those components.

Summary of Findings

Severity	# of Findings
High	1
Medium	1
Low	2
Informational	0
Total	4

Throughout the testing, we identified 4 issues of high, medium, and low severity:

- 1 issue of users being unable to withdraw their staked tokens
- 1 issue of wrong behavior of the owner function withdrawRewardTokens()
- 1 issue of index collision in the redeem array data structure
- 1 issue of missing events in the functions that change the important states of Farms

Disclaimer

This testing report should not be used as investment advice.

Narya.ai uses an automatic testing technique to test smart contracts' security properties and business logic rapidly and continuously. However, we do not provide any guarantees on eliminating all possible security issues. The technique has its limitations: for example, it may not generate a random edge case that violates an invariant during the allotted time. Its use is also limited by time and resource constraints.

Unlike time-boxed security assessment, Narya.ai advises continuing to create and update security tests throughout the project’s lifetime. In addition, Narya.ai recommends proceeding with several other independent audits and a public bug bounty program to ensure smart contract security.

Project Overview

We manually review and test the security of four smart contracts:

- STEADY: An ERC20 contract that implements Steadefi's native token.
- esSTEADY: An ERC20 contract that represents a non-transferable "escrowed" version of STEADY. The esSTEADY tokens can only be transferred to the addresses included in the whitelist by the owner.
- TokenManager: A contract used to convert STEADY tokens to esSTEADY, allocate them into plugin contracts, and redeem esSTEADY.
- Farms: A contract used by the owner to create farms for a user to stake STEADY tokens and claim rewards. It has the ability to boost the positions of a user.

Tests

Tested Invariants and Properties

We relied on the Narya engine that used a smart fuzzing approach to test the following [TODO] invariants and properties of the smart contracts.

Table 1 Tested Invariants

ID	Invariant/Property Description	Found Bug(s)
01	Owner should be able to update the transfer whitelist for esSTEADY Contract.	Passed
02	Owner Should be able to update the TokenManager address for esSTEADY Contract.	Passed
03	STEADY value should be returned correctly by vesting duration.	Passed
04	User should be able to convert STEADY to esSTEADY.	Passed
05	Redemption should not be finalized before the duration is up.	Passed
06	User should be able to redeem esSTEADY to STEADY for maximum duration.	Passed
07	User should be able to cancel redeeming.	Passed
08	User should be able to redeem the correct amount of tokens.	Passed
09	User should be able to finalize multiple positions in any order.	2. Index collision in redeem array

10	User should be able to set approvals for usage plugins.	Passed
11	Owner should correctly set deallocation fees for usage plugins.	Passed
12	User should be to allocate esSTEADY to a plugin.	Passed
13	User should be to deallocate esSTEADY from a plugin.	Passed
14	Owner should be able to create a new farm.	Passed
15	Owner should be able to deposit reward tokens to Farms	Passed
16	Owner should be able to withdraw reward tokens from Farms.	1. Wrong behavior of withdrawRewardsTokens()
17	User should be able to stake tokens to the active Farms	Passed
18	Users should have earned reward from staking.	Passed
19	Users should be able to claim rewards from Farms in esSTEADY and STEADY tokens.	Passed
20	Users should not be able to transfer esSTEADY.	Passed
21	Users should be able to claim their staked tokens and rewards.	4. Users unable to withdraw their stakes if farm rewards less than claimable

Key Findings and Recommendations

1. Wrong behavior of withdrawRewardsTokens()

Severity: **Medium**

Description

The `withdrawRewardsTokens` function of the `Farms` contract does have an expected behavior. It duplicates the behavior of the `depositRewardsTokens` function.

Code 1 contracts/staking/Farms.sol#L434

```
function withdrawRewardsTokens(uint256 _id, uint256 _amount) external nonReentrant
onlyOwner {
        Farm storage farm = farms[_id];
        require(_amount > 0, "Cannot withdraw 0 amount");
        require(
            farm.totalRewards > 0, "Cannot withdraw when farm has no reward tokens
deposited"
        );
        require(
            _amount <= farm.totalRewards,
            "Cannot withdraw more reward tokens than deposited in farm"
        );

        IERC20(steady).safeTransfer(msg.sender, _amount);
        farm.totalRewards += _amount;
    }
```

Impact

The owner of the farm cannot withdraw reward tokens from the farm and thus lose them.

Failed Invariant

```
function testOwnerWithdrawRewardTokensFromFarm(uint256 _amountOfRewardTokens,
uint256 _amountOfWithdrawTokens) public {
        vm.assume(_amountOfRewardTokens < steady.balanceOf(owner));
        vm.assume(_amountOfRewardTokens > 0);
        vm.assume(_amountOfWithdrawTokens < _amountOfRewardTokens);
        vm.assume(_amountOfWithdrawTokens > 0);
        _createNewFarm();
        vm.startPrank(owner);
```

```
        farms.depositRewardsTokens(0, _amountOfRewardTokens);
        farms.withdrawRewardsTokens(0, _amountOfWithdrawTokens);

        Farms.Farm memory farm0 = _getFarm(0);

        require(farm0.totalRewards == _amountOfRewardTokens -
_amountOfWithdrawTokens, "Total number of rewards is wrong");
    }
```

Recommendation

- Change addresses for token transfer.
- Subtract the amount from total rewards.

Remediation

This issue has been acknowledged by Steadefi and several logic changes were made:

- Variable `endTime` was added to Farm struct to prevent acquiring of rewards after block.timestamp has passed it.
- Logic of functions `currentRewardPerStakedToken` and `_updateFarm` also was changed.

Issue was fixed at commit 1ba974db10693838aa7378ca2a5c9e14cb7cce56.

2. Index collision in redeem array

Severity: **Low**

Description

Users can have several redeem positions in the TokenManager contract. These positions will have their own redeem indexes that users use when they want to finalize or cancel redemption. If users want to redeem or cancel any position other than the last added, the indexes of other positions will be changed, and users should take this into account when redeeming or cancelling the next position.

Code 2 contracts/tokens/TokenManager.sol#L253-L264

```
function finalizeRedeem(uint256 _redeemIndex) external nonReentrant
validateRedeem(msg.sender, _redeemIndex) {
    EsSTEADYBalance storage balance = esSTEADYBalances[msg.sender];
    RedeemInfo storage _redeem = userRedeems[msg.sender][_redeemIndex];
    require(_currentBlockTimestamp() >= _redeem.endTime, "finalizeRedeem: vesting
duration has not ended yet");

    // remove from SBT total
    balance.redeemingAmount = balance.redeemingAmount - (_redeem.esSTEADYAmount);
    _finalizeRedeem(msg.sender, _redeem.esSTEADYAmount, _redeem.STEADYAmount);

    // remove redeem entry
    _deleteRedeemEntry(_redeemIndex);
  }
function _deleteRedeemEntry(uint256 _index) internal {
    userRedeems[msg.sender][_index] =
userRedeems[msg.sender][userRedeems[msg.sender].length - 1];
    userRedeems[msg.sender].pop();
  }
```

This happens because the `userRedeems` struct that represents the user's positions stores them as an array. When the user calls `finalizeRedeem` it swaps the last element with the redeemed one and the indexes of elements are changed.

Impact

If a user swaps the elements other than the last one and then redeems the last one, the transaction will be reverted. This is because the index of the last element will be changed to the the previous deleted entry.

Failed Invariant

```
function testShouldFinalizeMultiplePositions() public {
        _unpauseTokenManager();

        vm.startPrank(user1);
        tokenManager.convert(100e18);
        uint256 STEADYBalanceBefore = steady.balanceOf(user1);
        tokenManager.redeem(10e18, MIN_REDEEM_DURATION);
        tokenManager.redeem(10e18, MID_REDEEM_DURATION);
        tokenManager.redeem(10e18, MAX_REDEEM_DURATION);

        skip(MIN_REDEEM_DURATION);
        tokenManager.finalizeRedeem(0);
        skip(MID_REDEEM_DURATION - MIN_REDEEM_DURATION);
        tokenManager.finalizeRedeem(1);
        skip(MAX_REDEEM_DURATION - MID_REDEEM_DURATION);
        tokenManager.finalizeRedeem(2);
        vm.stopPrank();

        require(steady.balanceOf(user1) - STEADYBalanceBefore == ( ( 10e18
* _durationToPercentage(MIN_REDEEM_DURATION) / 1e18 )
                                                                  + ( 10e18
* _durationToPercentage(MID_REDEEM_DURATION) / 1e18 )
                                                                  + ( 10e18
* _durationToPercentage(MAX_REDEEM_DURATION) / 1e18 )), "Balance value is
wrong");
    }
```

Recommendation

- Reorganize the `userRedeems` data structure from array to mapping
- Taking into account changing of indexes on the off-chain side and restricting user behavior which can cause problem explained above

Remediation

This issue has been acknowledged by Steadefi. Steadefi decided to design the off-chain service to read `userRedeems` mapping to determine the correct index before calling `finalizeRedeem`. Therefore, Steadefi will not deploy any fix for this issue.

3. Missing events in the functions that change important states

Severity: **Low**

Description

In the `depositRewardsTokens and withdrawRewardsTokens` functions that affect the status of sensitive variables should be able to emit events as notifications to users.

Code 3 contracts/staking/Farms.sol#L420-L447

```
/**
  * Deposit more reward tokens to a farm
  * @param _id  Unique id of farm
  * @param _amount  Amount of reward tokens to deposit; in reward token's
decimals
  */
  function depositRewardsTokens(uint256 _id, uint256 _amount) external
nonReentrant onlyOwner {
    require(_amount > 0, "Cannot deposit 0 amount");

    Farm storage farm = farms[_id];

    IERC20(steady).safeTransferFrom(msg.sender, address(this), _amount);
    farm.totalRewards += _amount;
  }

  /**
  * Deposit more reward tokens to a farm
  * @param _id  Unique id of farm
  * @param _amount  Amount of reward tokens to deposit; in reward token's
decimals
  */
  function withdrawRewardsTokens(uint256 _id, uint256 _amount) external
nonReentrant onlyOwner {
    Farm storage farm = farms[_id];
    require(_amount > 0, "Cannot withdraw 0 amount");
    require(
      farm.totalRewards > 0, "Cannot withdraw when farm has no reward
tokens deposited"
    );
    require(
      _amount <= farm.totalRewards,
```

```
      "Cannot withdraw more reward tokens than deposited in farm"
    );

    IERC20(steady).safeTransfer(msg.sender, _amount);
    farm.totalRewards += _amount;
  }
```

Recommendation

Add events for these sensitive actions and emit them in the functions.

Remediation

This issue has been acknowledged by Steadefi and fixed at commit bd072d07956c47d1edee87aa7362f2f693d09b8e.

4. Users are unable to withdraw their stakes if the farm rewards are less than the claimable ones

Severity: **High**

Description

When a user calls the `unstake` function, the `claim` function invoked checks if the user has earned rewards. When the earned rewards are bigger than total rewards in Farms, the transaction will be reverted.

Code 4 contracts/staking/Farms.sol#L245-L306

```
function unstake(uint256 _id, uint256 _amount)
    public
    nonReentrant
    whenNotPaused
{
    require(_amount > 0, "Cannot unstake 0");

    Position storage position = positions[_id][msg.sender];

    require(position.stakedAmount >= _amount, "Cannot unstake more than
staked");

    claim(_id);

    _updateFarm(_id);
    position.rewardsDebt = 0;
    position.stakedAmount = position.stakedAmount - _amount;
    _updateUserRewardsDebt(_id, msg.sender, position.stakedAmount);

    Farm storage farm = farms[_id];
    farm.totalStaked = farm.totalStaked - _amount;
    IERC20(farm.stakedToken).safeTransfer(msg.sender, _amount);

    emit Unstake(_id, msg.sender, farm.stakedToken, _amount);
}

function claim(uint256 _id) public whenNotPaused {
    uint256 rewards = rewardsEarned(_id, msg.sender);

    if (rewards > 0) {
        Farm storage farm = farms[_id];
```

```
        require(
            farm.totalRewards >= rewards,
            "Rewards deposited in farm less than rewards claimable"
        );

        Position memory position = positions[_id][msg.sender];

        position.rewardsRedeemed = position.rewardsRedeemed + rewards;
        position.rewardsDebt = position.stakedAmount *
currentRewardPerStakedToken(_id)
            * getRewardMultiplier(_id, msg.sender)
            / SAFE_MULTIPLIER;
        positions[_id][msg.sender] = position;
        farm.totalRewards -= rewards;

        if (farm.esSteadySplit > 0) {
            uint256 esSteadyAmount = rewards * farm.esSteadySplit /
SAFE_MULTIPLIER;
            uint256 steadyAmount = rewards - esSteadyAmount;

            IERC20(steady).safeTransfer(msg.sender, steadyAmount);
            ITokenManager(tokenManager).convertTo(esSteadyAmount,
msg.sender);
        } else {
            IERC20(steady).safeTransfer(msg.sender, rewards);
        }

        emit Claim(_id, msg.sender, steady, rewards);
    }
}
```

Impact

User funds used for staking may get stuck in a staking contract, which requires the owner to deposit additional rewards to the farm. Another way to allow the user to unstake the staked tokens is to change `d` to 0. But this leads to losing all earned rewards for users.

Failed Invariant

```
function testUserShouldBeAbleToUnstakeAndClaimEverything(uint256 _time)
public {
        vm.assume(_time < 100 days);
```

```
        vm.assume(_time > 0);
        _createNewFarm();
        _ownerDepositsReward();
        _userStakesTokensToFarm();

        skip(20);
        vm.startPrank(user1);
        farms.claim(0);
        skip(_time);
        farms.unstake(0, 10e18);

        Farms.Farm memory farm0 = _getFarm(0);
        Farms.Position memory positionUser1 = _getPosition(0, user1);
        require(farms.rewardsEarned(0, user1) == 0, "Rewards Earned value
is wrong");
    }
```

Recommendation

- Revising the staking logic.
- Supporting a user to unstake the rewards in case his earned rewards are more than the rewards in the farm.

Remediation

This issue has been acknowledged by Steadefi and fixed at commit 24868f48cee9468d40031e3470b80bb3d990e4c5.

Fix Log

Table 6 Fix Log

ID	Title	Severity	Status
01	Wrong behavior of withdrawRewardsTokens()	**Medium**	**Fixed at commit 1ba974db10693838aa7378ca2a5c9e14cb7cce56**
02	Index collision in redeem array	**Medium**	**Acknowledged***
03	Missing events in the functions that change important states	**Low**	**Fixed at commit bd072d07956c47d1edee87aa7362f2f693d09b8e**
04	Users are unable to withdraw their stakes if the farm rewards are less than the claimable ones	**High**	**Fixed at commit 24868f48cee9468d40031e3470b80bb3d990e4c5**

* The issues have been acknowledged by Steadefi but won't be fully fixed or mitigated at the time of this report. Check the former description of each issue for details.

Appendix

Severity Categories

Severity	Description
Gas	Gas optimization
Low	A low-severity issue does not put assets at risk such as function being inconsistent with specification or issues with comments.
Medium	A medium-severity issue puts assets at risk not directly but with a hypothetical attack path, stated assumptions, or external requirements. Or the issue impacts the functionalities or availability of the protocol.
High	A high-severity issue directly results in assets being stolen, lost or compromised.