The Standard Security Review

Version 2.0

11.05.2024

Conducted by:

MaslarovK, Independent Security Researcher

Table of Contents

1 About MaslarovK

MaslarovK is an independent security researcher from Bulgaria. He has secured various protocols through private audits and public contests - Secured ~$5M in TVL.

2 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

3 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

3.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

3.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

Overview

Project Name	The Standard
Repository	https://github.com/the-standard/staking-v2
Commit hash	d14e528e00a272a05c54c9ad7c9540336ed1e91f
Resolution	90e031f4286d1574507b93774107ac7f610c06ad
Documentation	N/A
Methods	Manual review & testing

Scope

contracts/Staking.sol
contracts/RewardGateway.sol

Issues Found

Critical risk	1
High risk	0
Medium risk	1
Low risk	1
Informational	3

5 Findings

5.1 Critical risk

5.1.1 Reentrancy can be used to drain most of the contract's funds

Severity: *Critical risk*

Context: Staking.sol#L169

Description: In the `Staking::decreaseStake`

```
function decreaseStake(uint256 _tst, uint256 _euros) external {
        IRewardGateway(rewardGateway).dropFees();
        Position memory _position = positions[msg.sender];
        runClaim(_position, false);

        if (_tst > _position.TST || _euros > _position.EUROs) revert InvalidRequest
            ();
        _position.TST -= _tst;
        _position.EUROs -= _euros;

        savePosition(_position);

        if (_tst > 0) IERC20(TST).safeTransfer(msg.sender, _tst);
        if (_euros > 0) IERC20(EUROs).safeTransfer(msg.sender, _euros);
    }
```

`runClaim` is called before updating the state and contains a call to `claimRewards`. Here comes the problem:

```
function claimRewards(address _holder, Reward[] memory _rewards) private {
        for (uint256 i = 0; i < _rewards.length; i++) {
            Reward memory _reward = _rewards[i];
            if (_reward.token == address(0)) {
                (bool sent,) = _holder.call{value: _reward.amount}("");
                require(sent);
            } else {
                IERC20(_reward.token).safeTransfer(_holder, _reward.amount);
            }
        }
    }
```

As you can see, there is a low-level call to the _holder when transfering native token. Since there is no Reentrancy guard and CEI pattern is not followed, this allows the malicious user to drain most of the reward tokens balances.

Recommendation: Add reentrancy guard to all external functions.

Resolution: Fixed

5.2 Medium risk

5.2.1 Approve to zero first due to possibility of using USDT

Severity: *Medium risk*

Context: RewardGateway.sol#L49 RewardGateway.sol#L59

Description: There are several instances where `approve` is called, but due to the possibility of using USDT and its implementation - it will revert if not approved to 0 first.

Recommendation: Approve to 0 first before approving the real amount.

Resolution: Fixed

5.3 Low risk

5.3.1 `totalDays` will return wrong result if `start = 0`

Severity: *Low risk*

Context: Staking.sol#L39

Description: In the `Staking::totalDays`, the intended behavior is to return 0 days if no stake has been made yet

```
    function totalDays() private view returns (uint256) {
        return (block.timestamp - start) / 1 days;
    }
```

But the current implementation will return the current `(block.timestamp - 0)/ 1 days`, which will result in the days passed from 1970 to the moment of execution.

Recommendation: Check if `start = 0` and return 0 directly if that's the case.

Resolution: Fixed

5.4 Informational

5.4.1 Consider refactoring the `_deleteIndexFromStarts` function to save gas.

Severity: *Informational risk*

Context: Staking.sol#L39

Description: In the `Staking::_deleteIndexFromStarts`, the whole array is itterated which is not really gas efficient.

```
    function _deleteIndexFromStarts(uint256 _index) private {
        for (uint256 i = _index; i < starts.length - 1; i++) {
            starts[i] = starts[i+1];
        }
        starts.pop();
    }
```

Recommendation: Due to how starts are positioned in the array does not matter, the function could be changed as follows:

```
    function _deleteIndexFromStarts(uint256 _index) private {
        starts[_index] = starts[starts.length - 1];
        starts.pop();
    }
```

Resolution: Fixed

5.4.2 Consider adding underscore to all the private functions for better readability

Severity: *Informational risk*

Resolution: Fixed

5.4.3 Consider adding events

Severity: *Informational risk*

Resolution: Fixed