Shutter Staking Audit

Côme du Crest

2024-09-01

Table of contents

Shutter Staking Audit

This document presents the finding of a smart contract audit conducted by Côme du Crest for Shutter.

Scope

The scope includes all contracts within blockful-io/shutter-staking/src as of commit `a5210f4`.

Context

The core of the contracts comprise two staking contracts for regular and delegated staking of SHU tokens as well as a reward distributor contract that distribute SHU tokens to stakers accrued every seconds.

Status

The report has been sent to the core developers.

Issues

[Low] Keyper can have delegated stake below min value

Summary

The contract `Staking` enforces a min value for the first stake of a keyper. However, anyone can use `DelegateStaking` to delegate any value to a keyper holding no stake at all. In an abstract way keypers can be responsible for a stake below min value.

Vulnerability Detail

`Staking.stake()` enforces a min value for the first stake of a keyper:

```
    function stake(
        uint256 amount
    ) external updateRewards returns (uint256 stakeId) {
        require(keypers[msg.sender], OnlyKeyper());

        require(amount > 0, ZeroAmount());

        // Get the keyper stakes
        EnumerableSetLib.Uint256Set storage stakesIds = userStakes[msg.sender];

        // If the keyper has no stakes, the first stake must be at least the
            minimum stake
        if (stakesIds.length() == 0) {
            require(amount >= minStake, FirstStakeLessThanMinStake());
        }

        stakeId = nextStakeId++;

        // Add the stake id to the user stakes
        userStakes[msg.sender].add(stakeId);

        // Add the stake to the stakes mapping
        stakes[stakeId].amount = amount;
        stakes[stakeId].timestamp = block.timestamp;
        stakes[stakeId].lockPeriod = lockPeriod;

        _deposit(amount);

        emit Staked(msg.sender, amount, lockPeriod);
    }
```

Anyone can delegate via `DelegateStaking.stake()` to a keyper with no restriction on the stake amount:

```
    function stake(
        address keyper,
        uint256 amount
    ) external updateRewards returns (uint256 stakeId) {
        require(amount > 0, ZeroAmount());

        require(staking.keypers(keyper), AddressIsNotAKeyper());

        stakeId = nextStakeId++;

        // Add the stake id to the user stakes
        userStakes[msg.sender].add(stakeId);

        // Add the stake to the stakes mapping
        stakes[stakeId].keyper = keyper;
        stakes[stakeId].amount = amount;
        stakes[stakeId].timestamp = block.timestamp;
        stakes[stakeId].lockPeriod = lockPeriod;

        // Increase the keyper total delegated amount
        unchecked {
            totalDelegated[keyper] += amount;
        }

        _deposit(amount);

        emit Staked(msg.sender, keyper, amount, lockPeriod);
    }
```

Impact

If we take into account the self-locked stake and delegated stake, a keyper could have a stake below `Staking.minStake`.

Code Snippets

https://github.com/blockful-io/shutter-staking/blob/a5210f40d61fc6f002b0ed48e46a327aa56975f4/src/Staking.sol#L145-L173

https://github.com/blockful-io/shutter-staking/blob/a5210f40d61fc6f002b0ed48e46a327aa56975f4/src/DelegateStaking.sol#L144-L171

Recommendation

In `DelegateStaking.stake()` enforce that the delegatee has at least `minStake` locked.

[Info] RewardsDistributor may fail to distribute rewards

Summary

When rewards are not distributed for a long period of time, the total reward to distribute may accumulate above the balance of the `RewardsDistributor` which will fail to distribute any rewards.

Vulnerability Detail

The function to distribute rewards `collectRewards()` will return `0` when rewards exceed the balance of the contract:

```
    function collectRewards() external override returns (uint256 rewards) {
        RewardConfiguration storage rewardConfiguration = rewardConfigurations[
            msg.sender
        ];

        // difference in time since last update
        uint256 timeDelta = block.timestamp - rewardConfiguration.lastUpdate;

        rewards = rewardConfiguration.emissionRate * timeDelta;

        // the contract must have enough funds to distribute
        // we don't want to revert in case its zero to not block the staking contract
        if (rewards == 0 || rewardToken.balanceOf(address(this)) < rewards) {
            return 0;
        }  // @audit if rewards are not collected for a while, they will accumulate past the balance of the contract and not be distributed

        // update the last update timestamp
        rewardConfiguration.lastUpdate = block.timestamp;

        // transfer the reward
        rewardToken.transfer(msg.sender, rewards);

        emit RewardCollected(msg.sender, rewards);
    }
```

Impact

I do not see a strong incentive to collect rewards regularly. As such, we could reach a point where the `RewardsDistributor` owes more rewards than its balance and is unable to disburse them.

The only solution to distribute rewards correctly if that happens is via the admin function `withdrawFunds()` to withdraw funds to the `Staking` contract.

Code Snippets

https://github.com/blockful-io/shutter-staking/blob/a5210f40d61fc6f002b0ed48e46a327aa56975f4/src/RewardsDistributor.sol#L89-L91

Recommendation

Acknowledge that the `RewardsDistributor` contract will always hold enough funds to collect rewards or adapt the function so that it distribute as much rewards as possible and update the timestamp proportionally to the amount of distributed reward. This will have the drawback that rewards may be distributed unfairly to one collector above other collectors.

[Info] Cannot unstake delegatee stakes without timing restriction when delegatee is not keeper

Summary

For regular staking, unstaking is allowed with no timing and min stake restriction when staker is no longer a keyper. This is not the case for delegated staking which remains locked for the whole duration even when delegatee is no longer a keyper.

Vulnerability Detail

In `Staking.unstake()` if the `keyper` argument is no longer registered as a keeper, the stake can be withdrawn before the end of the `lockPeriod`:

```
    function unstake(
        address keyper,
        uint256 stakeId,
        uint256 _amount
    ) external updateRewards returns (uint256 amount) {
        ...

        // Checks below only apply if keyper is still a keyper
        // if keyper is not a keyper anymore, anyone can unstake for them, lock
             period is
        // ignored and minStake is not enforced
        if (keypers[keyper]) {
            // Only the keyper can unstake
            require(msg.sender == keyper, OnlyKeyper());

            ...
            uint256 lock = keyperStake.lockPeriod > lockPeriod
                ? lockPeriod
                : keyperStake.lockPeriod;

            unchecked {
                require(
                    block.timestamp > keyperStake.timestamp + lock,
                    StakeIsStillLocked()
                );
            }

            ...
            uint256 maxWithdrawAvailable = convertToAssets(balanceOf(keyper)) -
                minStake;

            require(amount <= maxWithdrawAvailable, WithdrawAmountTooHigh());
        }

        ...
        uint256 shares = _withdraw(keyper, amount);

        emit Unstaked(keyper, amount, shares);
    }
```

On the contrary `DelegateStaking.unstake()` enforces the `lockPeriod` no matter the status of the keyper:

```
    function unstake(
        uint256 stakeId,
        uint256 _amount
    ) external updateRewards returns (uint256 amount) {
        require(
            userStakes[msg.sender].contains(stakeId),
            StakeDoesNotBelongToUser()
        );
        Stake memory userStake = stakes[stakeId];

        ...
        uint256 lock = userStake.lockPeriod > lockPeriod
            ? lockPeriod
            : userStake.lockPeriod;

        unchecked {
            require(
                block.timestamp > userStake.timestamp + lock,
                StakeIsStillLocked()
            );

            // Decrease the amount from the stake
            stakes[stakeId].amount -= amount;

            // Decrease the total delegated amount
            totalDelegated[userStake.keyper] -= amount;
        }
        ...
    }
```

Impact

Discrepancy in between direct and delegated staking. If possible, users aer better off staking using a vault contract that redistribute stake rewards and is set as a keyper than to use the delegated staking system implemented by Shutter.

Code Snippets

https://github.com/blockful-io/shutter-staking/blob/a5210f40d61fc6f002b0ed48e46a327aa56975f4/src/Staking.sol#L209-L237 https://github.com/blockful-io/shutter-staking/blob/a5210f40d61fc6f002b0ed48e46a327aa56975f4/src/DelegateStaking.sol#L188-L235

Recommendation

Allow to unstake immediately in `DelegateStaking` when delegatee is no longer a keyper.

[Info] No gap in upgradeable contract storage

Summary

The `BaseStaking` contract provision for the upgradable proxy pattern but does not declare a `__gap` value for storage as common for these type of contracts in case storage values need to be added to the implementation.

Vulnerability Detail

The `BaseStaking` contract inherits from upgradeable versions of Openzeppelin contracts and indicate wanting to implement an upgradable proxy pattern but does not decalre a `__gap` storage value to provision for future storage use:

```
import {ERC20VotesUpgradeable as ERC20Votes} from "@openzeppelin-upgradeable/
    contracts/token/ERC20/extensions/ERC20VotesUpgradeable.sol";

abstract contract BaseStaking is OwnableUpgradeable, ERC20Votes {
    ...
    constructor() {
        _disableInitializers();
    }
    ...
}
```

Both `Staking` and `DelegateStaking` inherit from `BaseStaking` and use storage:

```
contract Staking is BaseStaking {
    ...
}

contract DelegateStaking is BaseStaking {
    ...
}
```

Impact

If storage values are added in `BaseStaking` in future version of the contracts, there will be a conflict of storage layout with unpredictable impact.

Code Snippets

https://github.com/blockful-io/shutter-staking/blob/a5210f40d61fc6f002b0ed48e46a327aa56975f4/src/BaseStaking.sol#L11-L25

Recommendation

Follow Openzeppelin's recommendation of declaring a `uint256[47] __gap` value where `47` is `50` minus the three storage slots already used by `BaseStaking`. See documentation.