SyncSwap Vortex Audit Report

2024-03-01

Scope

https://github.com/syncswap/vortex-contracts/tree/8015ac5a126f56fe570ea6e4612695e7c1696996

VortexToken.sol

VortexDividends.sol

VortexBoostAdjuster.sol

VortexBoost.sol

SyncSwapVoter.sol

SyncSwapGauge.sol

SyncSwapBribe.sol

Conclusion

During this audit, 2 high risk findings, 7 medium risk findings and 9 low risk findings were identified, all of which have been addressed accordingly.

Findings

High Risk

H-1 In function `_withdraw`, `_updateAdjustedStakedBalances` should update `from` instead of `to`

In function `_withdraw`, the line below should update `from`'s adjusted balance instead of `to`'s. Otherwise a malicious user may be able to manipulate other's reward.

```
_updateAdjustedStakedBalances(to, _newUserStaked, _newTotalStaked, _adjustedUserStaked, _adjustedTotalStaked);
```

Reaction: Confirmed

H-2 `updateRewardToken` should not be used

```
function updateRewardToken(address rewardToken, address account, uint8 mode,
bytes memory claimData) external   override nonReentrant {
    if (account == address(0)) {
        _updateRewardToken(rewardToken, adjustedTotalStaked, account, 0, 0, "");
    } else {
        uint _adjustedUserStaked = adjustedUserStaked[account];
        uint _adjustedTotalStaked = adjustedTotalStaked;
        _updateRewardToken(rewardToken, _adjustedTotalStaked, account,
_adjustedUserStaked, mode, claimData);
        _updateAdjustedStakedBalances(account, userStaked[account], totalStaked,
_adjustedUserStaked, _adjustedTotalStaked);
    }
}
```

In function `updateRewardToken`, the `adjustedUserStaked` is updated. However, only one of the reward tokens is updated, accounting for the remaining tokens will be incorrect. (similar to M-2)

In a potential exploit scenario, Alice, who has a whale friend named Bob, only receives a 1.2x boost by holding veSync. She asks Bob to deallocate a large amount of tokens for a short period. Then, Alice calls `updateRewardToken`, enabling her to receive a 2.0x boost (due to the decreased `totalAllocation`). As a result, Alice can steal rewards from those not updated tokens and the reward will be insolvent.

Reaction: Confirmed

Medium Risk

M-1 `redeemAndClaim` will always fail

In function `redeemAndClaim`, `_burn(msg.sender, amount);` will always revert due to the check in `_beforeTokenTransfer`:

```
function _beforeTokenTransfer(address from, address to, uint amount) internal
view override {
    require(
        from == address(0) || from == address(this) || to == address(this) ||
        IVortexTokenWhitelister(whitelister).isTransferWhitelisted(msg.sender,
from, to, amount)/*,
        "Not allowed"*/
    );
}
```

Reaction: Confirmed, the workaround is using whitelister but better whitelist address(0) as 'to' by default.

M-2 `increaseAllocation` or `decreaseAllocation` without updating will lead to incorrect accounting

Suppose the user's `debtRewardPerShare` is last updated at time *t0*, the global `_rewardPerShare` is last updated at time *t1*. If `increaseAllocation` is called at time *t2* with `shouldUpdate = false`, the reward accounting from *t0* to *t1* will be incorrect, leading to insolvency.

Reaction: Confirmed, fixed by removing 'shouldUpdate', and update is always required.

M-3 When `_userAllocation < amount < _totalAllocation`, calculation in `decreaseAllocation` is incorrect

```
userAllocations[account] = amount >= _userAllocation ? 0 : (_userAllocation -
amount);
totalAllocation = amount >= _totalAllocation ? 0 : (_totalAllocation - amount);
```

In function `decreaseAllocation`, if `_userAllocation < amount < _totalAllocation`, `userAllocations` will be deducted by `_userAllocation` while `totalAllocation` will be deducted by `amount`. The invariant `totalAllocation = sum(userAllocations)` will be broken.

Reaction: Confirmed, fixed by using '-=' on 'userAllocation' and 'totalAllocation' directly.

M-4 Allocated amount is not taken into consideration at `getAvailableBalance`

```
function getAvailableBalance(address account) external view override returns
(uint) {
    uint _userUsedVotes = IVoter(voter).userUsedVotes(account);
    uint _balance = balanceOf[account];
    return _balance > _userUsedVotes ? (_balance - _userUsedVotes) : 0;
}
```

Allocated amount also counts for user's voting power. However, it is not considered in function `getAvailableBalance`.

Reaction: Confirmed, it seems getAvailableBalance function is not used anywhere in our smart contracts/frontend at the moment, we are considering removing it or reevaluating it before final deployment.

M-5 Vortex Dividends should respond when deallocation fee is burnt

When user deallocates, a deallocation fee could be charged. The VortexDividends should decrease their allocation by `amountToBurn` instead of doing nothing. Otherwise, the user may receive more rewards than deserved.

```
/// @dev called on user deallocating veSYNC from the base module.
function onDeallocate(
    address sender,
    address account,
    address submodule,
    uint amount,
    uint deallocateAmount,
    uint fee
```

```
) external override onlyVortex {
    /// @dev dividends is the base module, and not a submodule,
    /// all veSYNC holders are automatically (de)allocated.
    /// Doing explicit deallocating is not allowed. We do nothing here and pass
to hook.

    address _hook = hook;
    if (_hook != address(0)) {
        IVortexBaseModule(_hook).onDeallocate(sender, account, submodule, amount,
deallocateAmount, fee);
    }
}
```

Reaction: Confirmed, it seems the logic in VortexDividends has been removed by accident in the latest commit.

M-6 After an account is removed from exemptions, its balance will be frozen

When an account is in exemptions, its allocation will be zero. If it is removed from exemptions again, its allocation will still be zero, so it can no longer redeem or transfer the balance since `userAllocations[account]` will underflow.

```
function _deallocate(
    address account,
    uint amount
) private {
    uint _userAllocation = userAllocations[account];
    uint _totalAllocation = totalAllocation;

    _update(account, _userAllocation, _totalAllocation, account, 0, "");

    if (_exemptions.contains(account)) {
        if (_userAllocation != 0) {
            userAllocations[account] = 0;

            if (_userAllocation > _totalAllocation) {
                totalAllocation = 0;
            } else {
                unchecked {
                    totalAllocation -= _userAllocation;
                }
            }
        }
    } else {
        // revert due to underflow
        userAllocations[account] -= amount;
        totalAllocation -= amount;
    }
}
```

Consider allocating `getUserTotalVortexAmount` manually in `setExemption` when removing an account from exemptions.

Reaction: Acknowledged, no changes will be made at the moment. The exemptions will be preset for address like liquidity pools. The workaround is using setAllocation for the address when removing it from the exemptions.

M-7 Calculation when `allowPartialVotes` is incorrect

```
if (allowPartialVotes) {
    uint _lastVotingPowers = userLastVotingPowers[account];
    if (_lastVotingPowers > totalWeight) { // partially voted
        uint unusedVotingPowers;
        unchecked { /// @dev already checked
            unusedVotingPowers = (_lastVotingPowers - totalWeight);
        }
        totalWeight += (totalWeight * unusedVotingPowers / _lastVotingPowers);
    }
}
```

The calculation should keep the same proportion of vote used before and after the recast. To achieve this goal, we should directly let `totalWeight = _lastVotingPowers` here.

Reaction: Confirmed

Low Risk

L-1 Non validated external call in `_allocateGuage` and `_deallocateGuage`

```
IGauge(gauge).update(account, 0, "");
```

Consider checking `isGauge` before making the call.

Reaction: Confirmed

L-2 An account in exemptions can still receive rewards as long as it does not redeem or transfer

Similar to M-6, consider clearing allocation in `setExemption` manually when adding an account to exemptions.

Reaction: Confirmed

L-3 `claimable` is still decreased to zero when balance is insufficient

When sending rewards, if balance is insufficient, `claimable` is still set to zero. Consider updating the `claimable` to the amount of the shortfall instead.

Reaction: Confirmed

L-4 `stake` will always fail

```
function stake(uint amount, address to) external { // router compatible interface
    put(amount, to, "");
}
```

The empty string here will always lead to revert.

```
(address gauge) = abi.decode(data, (address));
......
IGauge(gauge).update(account, 0, "");
```

Reaction: VortexBoost is not a default submodule, so the stake() function will not call it. This should be fine.

L-5 `userPoolVotes` can be tricked to be lower than actual amount by including duplicate pools in the array

```
for (uint i; i < n; ) {
    _pool = _pools[i];
    _gauge = gauges[_pool];
    require(_gauge != address(0), "Gauge not exists");

    _poolWeight = _weights[i];
    _userPoolVotes = _poolWeight * _votingPowers / _totalWeight;

    if (_userPoolVotes != 0) {
        if (!isPoolPaused[_pool]) {
            userPoolVotes[_pool][_account] = _userPoolVotes;
```

When the input array includes duplicate pools, `userPoolVotes` will be overwritten, `userPoolVotes` can be tricked to be lower than actual amount. Consider changing the line to `userPoolVotes[_pool][_account] += _userPoolVotes;`

Reaction: Confirmed

L-6 No check that `minDistributeAmount` is larger than `rewardDuration`,

```
if (_claimable != 0) {
            if (_distribute(pool, _claimable, _minDistributeAmount)) { // whether
successful
                data.claimable = 0; // resets claimable if succeed.
            } else {
                data.claimable = _claimable; // updates claimable if not succeed.
            }
```

```
(bool success,) = IGauge(gauge).supplyRewards(rewardToken, claimable);`
```

If `minDistributeAmount` is smaller than `rewardDuration`, `_distribute` will return false and `claimable` will remain unchanged after failed `supplyRewards`. A malicious user can update the pool repeatedly to transfer the reward token to the gauge.

Consider always checking that `minDistributeAmount` is larger than `rewardDuration`.

Reacion: Confirmed. Fix: Gauge.supplyRewards will revert if rewardRate is 0, and voter to use try-catch to check if the function is successful.

L-7 No pop for `pools`

The length of `pools` can only increase. Consider adding a method to pop.

Reaction: Acknowledged. No changes to be made at the moment given the limited impact.

L-8 Deallocation fee can be set too high

```
function setDeallocationFee(address _submodule, uint24 fee) external onlyOwner {
    require(fee <= 1e5, "Invalid fee");
    deallocationFees[_submodule] = fee;
    emit SetDeallocationFee(_submodule, fee);
}
```

Deallocation fee can be set up to 100%! Consider setting a lower limit.

Reaction: Confirmed. Add a limit is reasonable.

L-9 veSync is not considered in `rescueERC20` in VortexToken

All the veSync token can be transferred out! Consider adding these lines in `rescueERC20`.

```
if (token == address(this)) {
    balance -= totalAllocatedAmount + totalRedeemingAmount;
}
```

Reaction: Confirmed