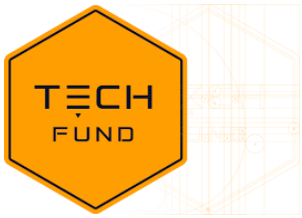


Ebitempura
Audit Report

Conducted in June 2021

1. Executive Summary

There were 3 medium issues, 1 low level issue, 5 note level findings. The source code was made available via Github https://github.com/ebitempuraswap/solid-giggle (Commit Id b008265a25164a62fec02c079fc687bd9eaf6448)

2.0 Detailed Analysis and Suggestions

These are the results of detailed analysis and suggestions derived from it. There are totally 9 findings.

Risk Level: Critical	0
Risk Level: High	0
Risk Level: Medium	3
Risk Level: Low	1
Risk Level: Note	5
Risk Level: None	0

2.1. Fix a compiler version to use

Risk Level: Note

The solidity compiler used in the contract is outdated **(pragma solidity >= 0.5.0)**, we suggest using the latest stable versions of Solidity compiler.
It is suggested to use a particular version as there might be changes in future versions of a compiler which cannot be checked for now and might lead to breaking changes in deployed smart contracts.

2.2 MasterChef massUpdatePools loop

Risk Level: Note

Avoid loops with big or unknown number of steps

```
// Update reward variables for all pools. Be careful of gas spending!
function massUpdatePools() public {
    uint256 length = poolInfo.length;
    for (uint256 pid = 0; pid < length; ++pid) {
        updatePool(pid);
    }
}
```

2.3 EbitempuraToken getChainId

Risk Level: Note

Avoid using Assembly code where possible.

```
function getChainId() internal pure returns (uint) {
    uint256 chainId;
    assembly { chainId := chainid() }
    return chainId;
}
```

2.4 Loss of Precision

Risk Level: Medium

```
ebitempura.mint(devAddress, ebitempuraReward.div(10).mul(devFee).div(100));
ebitempura.mint(treasuryAddress, ebitempuraReward.div(10).mul(100-devFee).div(100)
```

Solidity integer division might truncate, performing mul before division can cause loss of precision

div > mul > div ==> mul > div > div

2.5 Avoid direct mathematical actions

Risk Level: Note

Avoid using + , - , * , / directly as these may lead to overflow issue. Consider using safemath functions.

```
        // First check most recent balance
        if (checkpoints[account][nCheckpoints - 1].fromBlock <= blockNumber) {
            return checkpoints[account][nCheckpoints - 1].votes;
        }

        // Next check implicit zero balance
        if (checkpoints[account][0].fromBlock > blockNumber) {
            return 0;
        }

        uint32 lower = 0;
        uint32 upper = nCheckpoints - 1;
        while (upper > lower) {
            uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
            Checkpoint memory cp = checkpoints[account][center];
            if (cp.fromBlock == blockNumber) {
                return cp.votes;
            } else if (cp.fromBlock < blockNumber) {
                lower = center;
            } else {
                upper = center - 1;
            }
        }
        return checkpoints[account][lower].votes;
    }
```

2.6 Missing Checks

Risk Level: Note

Missing check or result of transfers : The result of some transfers lack verification , consider adding **require** check for them

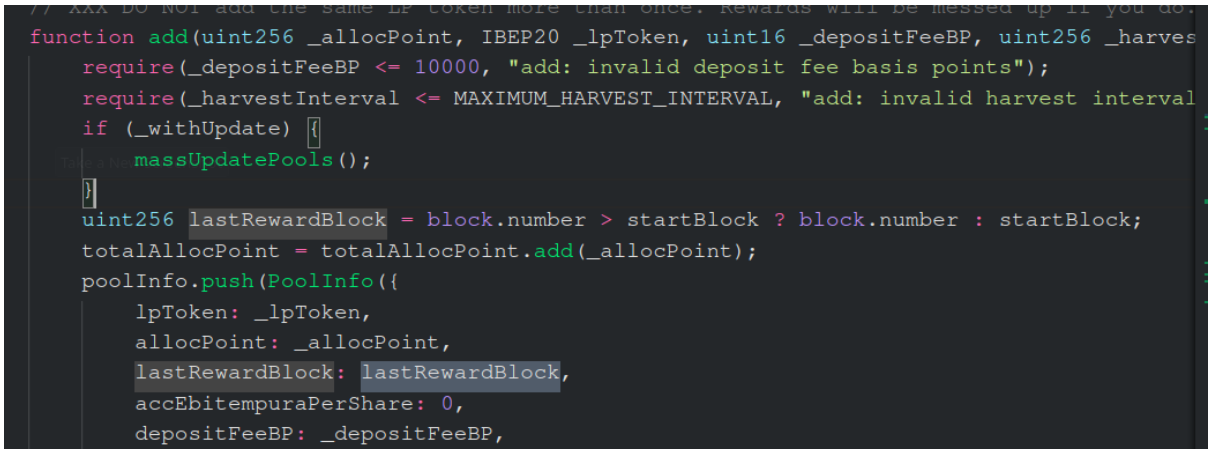
```
// Safe ebitempura transfer function, just in case if rounding error causes po
function safeEbitempuraTransfer(address _to, uint256 _amount) internal {
    uint256 ebitempuraBal = ebitempura.balanceOf(address(this));
    if (_amount > ebitempuraBal) {
        ebitempura.transfer(_to, ebitempuraBal);
    } else {
        ebitempura.transfer(_to, _amount);
    }
}
```

```
payOrLockupPendingEbitempura(_pid);
if (_amount > 0) {
    pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
    if (address(pool.lpToken) == address(ebitempura)) {
        uint256 transferTax = _amount.mul(ebitempura.transferTaxRate()).div(10000);
        _amount = _amount.sub(transferTax);
    }
```

2.7 Reward Miscalculation

Risk Level: Medium

Reward miscalculation, the issue will only happen when the _withUpdate parameter is set to false. We suggest removing the _withUpdate variable in the set and add functions and always call massUpdatePools().

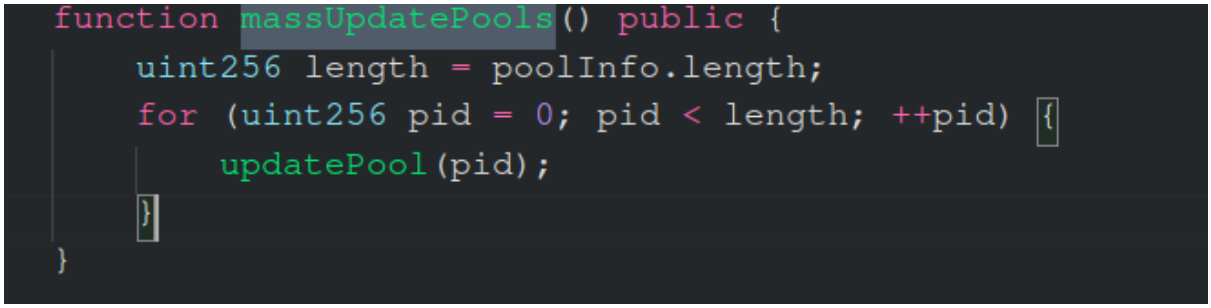


```
// XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.
function add(uint256 _allocPoint, IBEP20 _lpToken, uint16 _depositFeeBP, uint256 _harves
    require(_depositFeeBP <= 10000, "add: invalid deposit fee basis points");
    require(_harvestInterval <= MAXIMUM_HARVEST_INTERVAL, "add: invalid harvest interval
    if (_withUpdate) {
        massUpdatePools();
    }
    uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
    totalAllocPoint = totalAllocPoint.add(_allocPoint);
    poolInfo.push(PoolInfo({
        lpToken: _lpToken,
        allocPoint: _allocPoint,
        lastRewardBlock: lastRewardBlock,
        accEbitempuraPerShare: 0,
        depositFeeBP: _depositFeeBP,
```

2.8 Unusable massUpdatePools

Risk Level: Note

massUpdatePools will eventually be unusable due to excessive Gas usage (although
the likelihood is very low that poolInfo size will be raised until it is eventually unusable)
With current design pools cannot be removed and only disabled by pool allocPoint.



```
function massUpdatePools() public {
    uint256 length = poolInfo.length;
    for (uint256 pid = 0; pid < length; ++pid) {
        updatePool(pid);
    }
}
```

5.9 User will receive fewer tokens

Risk Level: Medium

Users will receive fewer tokens than claimed in payOrLockupPendingEbitempura user`s rewardLockedUp is set to zero and then safeEbitempuraTransfer is used to transfer. In case of low ebitempuraBal the complete amount is not transferred.

```
function payOrLockupPendingEbitempura(uint256 _pid) internal {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];

    if (user.nextHarvestUntil == 0) {
        user.nextHarvestUntil = block.timestamp.add(pool.harvestInterval);
    }

    uint256 pending = user.amount.mul(pool.accEbitempuraPerShare).div(1e12).sub(user.r
    if (canHarvest(_pid, msg.sender)) {
        if (pending > 0 || user.rewardLockedUp > 0) {
            uint256 totalRewards = pending.add(user.rewardLockedUp);

            // reset lockup
            totalLockedUpRewards = totalLockedUpRewards.sub(user.rewardLockedUp);
            user.rewardLockedUp = 0;
            user.nextHarvestUntil = block.timestamp.add(pool.harvestInterval);

            // send rewards
            safeEbitempuraTransfer(msg.sender, totalRewards);
            payReferralCommission(msg.sender, totalRewards);
        }
    } else if (pending > 0) {
        user.rewardLockedUp = user.rewardLockedUp.add(pending);
        totalLockedUpRewards = totalLockedUpRewards.add(pending);
        emit RewardLockedUp(msg.sender, _pid, pending);
    }
}
```

But in the above function the reward locked up is already set to zero. This will cause users to receive lower tokens than expected.

We suggested changing safeEbitempuraTransfer to return uint32 of balance and deducting that value from rewardLockedUp in payOrLockupPendingEbitempura.