Audit Report for Staking.sol

Accomplished by Dmitry Boiko

Target: **Staking.sol**

Notice:
1) #n - is a line number in Staking.sol file
2) Severity model taken from ConsenSys: minor, medium, major, critical
3) For static analysis used Slither

Issues found:

1. Not recommended using of floating pragma version (SWC-103):

```
#2: pragma solidity ^0.8.12;
```

severity: minor
recommendation: use fixed version of solc

2. Dangerous comparisons using timestamp (SCW-116)

```
#230: require(bool,string)(stakeInfo.activeUntil < block.timestamp,still
active)
#273: if (block.timestamp > stakeInfo.activeUntil) {
#293: if (block.timestamp > stakeInfo.activeUntil) {
```

severity: minor
recommendation: Avoid relying on block.timestamp

3. Division before multiply - precision can be lost

```
#278: periodsPassed = (time - lastRewardClaims[account]) / _DAY;
#282: (stakeInfo.amount * rewardRate * periodsPassed * 1e8)
```

severity: minor
recommendation: multiplyer can be used to avoid rounding

4. Unchecked return value on transfer (SCW-104)

```
#328: stakingToken.transfer(account, _amount);
```

severity: medium
recommendation: return value of transfer should be checked; SafeERC20 can be used

5. Reentrancy №1

after transferFrom call in _stake():

```
#310: stakingToken.transferFrom(msg.sender, address(this), amount),
```

state variables changing called:

```
#321: _setStakeInfo(account, newAmount, periods, block.timestamp, until);
#322: totalStaked = totalStaked + amount;
```

severity: medium
recommendation: use the Checks-Effects-Interactions pattern

6. Reentrancy №2

in prolong() function (#227) used:

```
#231: _collectRewards(msg.sender, true);
```

which uses transfer() at:

```
#346: rewardToken.transfer(account, rewardAmountInRewardToken),
```

and then state changing (see Reentrancy 1 issue for details):

```
#232: _stake(msg.sender, period, 0);
```

severity: medium
recommendation: use the Checks-Effects-Interactions pattern

7. Reentrancy №3

function stake() uses _collectRewards() and _stake() at:

```
#196: _collectRewards(msg.sender, true);
#198: _stake(msg.sender, period, amount);
```

which can cause Reentrancy 1 and 2

severity: medium
recommendation: use the Checks-Effects-Interactions pattern

8. Reentrancy №4

function withdraw() uses _collectRewards() (reentrancy 2)
also has it's own reentrancy at:

```
#207: _withdraw(msg.sender, stakes[msg.sender].amount);
```

by calling changing state at:

```
#327: _setStakeInfo(account, 0, 0, 0, 0);
```

before transfer() at:

```
#328: stakingToken.transfer(account, _amount);
```

severity: medium
recommendation: use the Checks-Effects-Interactions pattern

9. Reentrancy №5

in updateRewardsPool() call of transferFrom():

```
#249: stakingToken.transferFrom(msg.sender, address(this), amount),
```

before:

```
#252: rewardsBalance += amount;
```

severity: medium
recommendation: use the Checks-Effects-Interactions pattern

10. Using of rounded time values can cause not precise calculation of date

```
#76: uint256 private constant _DAYS_IN_MONT = 30;
#77: uint256 private constant _YEAR_IN_DAYS = 360;
```

severity: medium
recommendation: solidity predefined units should be used - seconds, minutes, hours, days and weeks

11. Unused variable

```
#94: uint256 public earlyWithdrawFee;
```

severity: minor
recommendation: remove unused variable

12. Not implemented logic or wrong discription

According to `emergencyWithdraw()` and `prolong()` descriptions - fee should be used in this functions, but it doesn't

severity: medium
recommendation: update description or implement fee logic

13. Unused mapping (SWC-109)

```
#80: mapping(uint256 => uint256[]) public levelPeriods;
```

severity: minor
recommendation: remove unused mapping

14. Unused mapping (SWC-109)

```
#82: mapping(address => bool) private _operators;
```

severity: minor
recommendation: remove unused mapping

15. Unused variable

```
#90: uint256 public tokenDecimals;
```

severity: minor
recommendation: remove unused variable

16. Public visibility level can be restricted

this variables/structs marked as public and some of them can be restricted:

```
stakes
levelPeriods
lastRewardClaims

availablePeriods
stakingToken
rewardToken
uniswapV2Router

tokenDecimals
totalStaked
rewardsBalance
rewardRate
```

```
earlyWithdrawFee
```

severity: minor
recommendation: review these variables according to business logic and restrict visibility if needed

17. Incorrect memory allocation

```
#161: StakeInfo storage stakeInfo = stakes[account];
#228: StakeInfo storage stakeInfo = stakes[account];
#292: StakeInfo storage stakeInfo = stakes[account];
```

severity: minor
recommendation: allocation of stakeInfo should be changed to "memory"

18. Unreachable logic in _collectRewards()

if during stake(), prolong() or withdraw() functions execution reached call of _collectRewards() (e.g. line #196):

```
_collectRewards(msg.sender, true);
```

then inside _collectRewards() on line #335 if "periods==0" can be "true" and "true" that will cause empty return call on line #336 or if "periods != 0" require fail will be reached on line #339:

```
require(periods > 0, "too early");
```

also periods can not be less then zero, because of uint256 type using

so logic on lines `#340 - #351` can't be ever reached

severity: major
recommendation: review if-statements logic for _collectRewards()

19. Missing non-zero check of amount

```
#247: function updateRewardsPool(uint256 amount){ ... }
```

severity: minor
recommendation: add non-zero check

20. Late validation of input parameter

call of `function stake(uint256 period, uint256 amount)` with invalid "period" parameter will fail only inside _stake() call at:

```
#198: _stake(msg.sender, period, amount);
```

but all logic from line #192 to #197 will be executed

severity: minor
recommendation: check of "period" validity should be performed earlier

21. Transfer of zero amount

check of zero amount at:

```
#315: if (amount == 0) {
```

should be performed before transfer at:

```
#310: stakingToken.transferFrom(msg.sender, address(this), amount),
```

severity: minor
recommendation: transfer call should be executed only if amount > 0

22. Useless statement

this statement will be always true because amount checks exists in all places before call of _collectRewards()

```
#333: if (stakes[account].amount > 0) {
```

severity: minor
recommendation: remove

23. Zero withdraw

in emergencyWithdraw() function set 0 before withdraw that always causes withdrawing of zero value

```
#218: stakes[msg.sender].amount = 0;
#220: _withdraw(msg.sender, stakes[msg.sender].amount);
```

severity: major
recommendation: zero should be set after withdrawal

24. Reward calculation not precise

```
#278: periodsPassed = (time - lastRewardClaims[account]) / _DAY;


#281: reward =
#282: (stakeInfo.amount * rewardRate * periodsPassed * 1e8) /
#283: 100 /
#284: _YEAR_IN_DAYS /
#285: 1e8;
```

example:

```
periodsPassed: (1660223544 - 1628687540) / 86400 = 365.0000462962963
reward: (1000 * 5 * 365 * 100,000,000) / 100 / 360 / 100,000,000 =
50.69444444444444
```

severity: minor
recommendation: rounding should be avoided in another way

25. No return value check

return value from toRewardTokens() that used in transfer(#346) is not checked:

```
#344: uint256 rewardAmountInRewardToken = toRewardTokens(reward);
```

it can be failed on call to uniswapV2Router at:

```
#375: uint256[] memory outAmountsOut =
uniswapV2Router.getAmountsOut(inputAmount,path);
```

severity: medium
recommendation: return value should be checked before transfer

26. uint8 packed in uint256:

decimals() function in IERC20DetailedBurnable returns uin8, that stored to tokenDecimals which is uint256:

```
#5: function decimals() external view returns (uint8);
#90: uint256 public tokenDecimals;
#104: tokenDecimals = stakingToken.decimals();
```

severity: minor
recommendation: type of tokenDecimals can be lowered to uint8

27. Withdraw any time

```
#204: function withdraw() external {
```

user can withdraw staked takens at any time, no checks that stake period has passed or not

severity: critical
recommendation: the until period should be checked before withdraw

28. Total staked amount is not updated in withdraw

no update for ‘totalStaked’ variable inside withdraw() function

severity: minor
recommendation: totalStaked should be updated

29. Until date for a stake set to zero

```
#318: until = stakes[account].activeUntil;
```

if this condition reached inside _stake() function for user's first stake, 'until' will be set to zero (as default value for uint)
this causes always true in the next statements:

```
#230: require(stakeInfo.activeUntil < block.timestamp, "still active");
#273: if (block.timestamp > stakeInfo.activeUntil) {
#293: if (block.timestamp > stakeInfo.activeUntil) {
```

also time will be set to 0 in:

```
#274: time = stakeInfo.activeUntil;
```

and this can cause underflow here:

```
#278: periodsPassed = (time - lastRewardClaims[account]) / _DAY;
```

severity: major
recommendation: activeUntil should not be set to zero at first stake

30. No check that reward more then zero:

```
#334: (uint256 periods, uint256 reward) = _rewardAmount(account);
```

severity: minor
recommendation: reward should be checked for prevent future code execution with zero value

31. No checks for external call

if this external call will fail:

```
#375: uint256[] memory outAmountsOut = uniswapV2Router.getAmountsOut(
```

this can cause errors inside _collectRewards() function:
rewardAmountInRewardToken can not be set and this will wail transfer at:

```
#346: rewardToken.transfer(account, rewardAmountInRewardToken),
```

severity: medium
recommendation: external call result should be checked