Alliance Block Liquidity Staker

Security Assessment

November 3rd, 2020

For :

Alliance Block

By :

Alex Papageorgiou @ CertiK

alex.papageorgiou@certik.org

Sheraz Arshad @ CertiK

sheraz.arshad@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Alliance Block Liquidity Staker
Description	Implementation of liquidity staking and rewards economics based on a specific time duration of staking.
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commits	Pre-audit: 65290342a3515df5857f9d066495c759a7a50333 Post-audit: f1b518ea678499a3be4a29e291355389ab572fb4

Audit Summary

Delivery Date	Nov. 03, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Oct. 27, 2020 - Nov. 03 2020

Vulnerability Summary

Total Issues	41
Total Critical	2
Total Major	1
Total Minor	3
Total Informational	35

Executive Summary

The codebase comprise of contracts implementing `Staking Rewards` contract and its `Factory` contract. The `Factory` contract keeps track of `stakingToken` and its correspondingly deployed `Rewards` contract. Users are able to deposit `stakingToken` and receive rewards in the form of `Reward Tokens`.
The contracts make use of OpenZeppelin contracts to implement ERC20 token, pausable, whitelisting and ownable functionalities.

All of the contracts in repository were reviewed and majority of the findings are informational for enhancing the optimization and code legibility of the contracts and two critical findings were identified related to correctness of the code.

Findings

ID	Title	Type	Severity	Resolved
RDR-01	Confusing Entity Name	Coding Style	Informational	✓
RDR-02	Unlocked Compiler Version	Language Specific	Informational	✓
SRS-01	Incorrect `import`	Inconsistency	Informational	✓
SRS-02	Redundant `struct` Property	Gas Optimization	Informational	✓
SRS-03	Unnecessary Constructor Parameter	Gas Optimization	Informational	✓
SRS-04	Confusing Variable Name	Language Specific	Informational	✓
SRS-05	Inefficient `storage` `Read`	Optimization	Informational	✓
SRS-06	Use of Literal Value	Undocumented Literal	Major	✓
SRS-07	Inefficient `storage` Read	Optimization	Informational	✓
SRS-08	Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	✓
SRS-09	Inefficient `storage` Read	Optimization	Informational	✓
SRS-10	Explicitly Returning a Local Variable	Optimization	Informational	✓
SRS-11	Incorrect Function Implementation	Code Legibility	Critical	✓
SRS-12	Comment Discrepency	Comment	Informational	✓
SRS-13	Incorrect Grammar	Grammar	Informational	✓

ID	Title	Type	Severity	Resolved
SRS-14	Inefficient Read from `storage`	Optimization	Informational	✓
SRS-15	Inefficient `storage` Read	Optimization	Informational	✓
SRS-16	Spelling Error	Comment	Informational	✓
SRS-17	Redundant require statement	Code Legibility	Minor	✓
SRS-18	Unnecessary Function Parameter	Optimization	Informational	✓
SRS-19	Spelling Error	Comment	Informational	✓
SRS-20	Returned Value of Function is not Checked	Code Legibility	Minor	✓
SRS-21	Usage of both `now` and `block.timestamp`	Code Legibility	Informational	✓
SRS-22	Incorrect Manipulation of State Variable	Code Legibility	Critical	✓
SRS-23	Unlocked Compiler Version	Language Specific	Informational	✓
SRS-24	Incorrect Order of Functions	Language Specific	Informational	✓
SRS-25	Visibility can be changed from `public` to `external`	Language Specific	Informational	✓
SRF-01	Inefficient storage Layout	Optimization	Informational	✓
SRF-02	Usage of `uint` alias instead of `uint256`	Language Specific	Informational	✓
SRF-03	Unnecessary explicit usage of modifier `Ownable`	Optimization	Informational	✓

ID	Title	Type	Severity	Resolved
SRF-04	Visibility can be changed from `public` to `external`	Optimization	Informational	✓
SRF-05	Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	✓
SRF-06	Ineffectual Predicate in `require` Statement	Code Legibility	Informational	✓
SRF-07	Spelling Error	Comment	Informational	✓
SRF-08	Reward Tokens and Amounts can be moved to StakingRewards Contract	Code Legibility	Informational	✓
SRF-09	Explicit Passing of msg.sender can be Omitted	Optimization	Informational	✓
SRF-10	Incorrect Order of Functions	Language Specific	Informational	✓
SRF-11	Unlocked Compiler Version	Language Specific	Informational	✓
SRF-12	Returned Value of Function is not Checked	Code Legibility	Minor	✓
SRF-13	Visibility can be changed from public to external	Optimization	Informational	✓
TEC-01	Usage of `uint` alias instead of `uint256`	Language Specific	Informational	✓

RDR-01: Confusing Entity Name

Type	Severity	Location
Coding Style	Informational	RewardsDistributionRecipient.sol L5 RewardsDistributionRecipient.sol L9

Description:

The state variable `rewardsDistribution` on `L7` refers to the factory contract that deploys and distribute rewards to instances of `StakingRewards` contracts. The name of the variable is confusing as it seems to refer to an operation rather than an entity. Similarly, the same variable name is utilized in the modifer `onlyRewardsDistribution` on `L9`. The confusing names for state variable and modifier decreases the quality of code.

Recommendation:

We advise that the state variable's and modifier's names on the aforementioned lines be rectified to correctly represent its content of address of `StakingRewardsFactory`. The names can be changed to f.e. `rewardsDistributor` and `onlyRewardsDistributor` or `factory` and `onlyFactory`.

```
address public rewardsDistributor;
```

```
modifier onlyRewardsDistributor() {
        require(msg.sender == rewardsDistribution, "Caller is not RewardsDistribution contract");
    _;
 }
```

Alleviation:

Alleviations were applied as advised.

RDR-02: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	RewardsDistributionRecipient.sol L1-L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.5.16` the contract should contain the following line:

```
pragma solidity 0.5.16;
```

Alleviation:

Alleviations were applied as advised.

SRS-01: Incorrect `import`

Type	Severity	Location
Coding Style	Informational	StakingRewards.sol L6

Description:

The import of `ERC20Detailed.sol` on the aforementioned line provides access to `IERC20` interface inside the context of contract. A more appropriate way would have to import the `IERC20.sol` directly.

```
import "openzeppelin-solidity-2.3.0/contracts/token/ERC20/ERC20Detailed.sol";
```

Recommendation:

We advise that the unused import on the aforementioned line be removed to increase the legibility of the code and instead replaced with the following import.

```
import "openzeppelin-solidity-2.3.0/contracts/token/ERC20/IERC20.sol";
```

Alleviation:

Alleviations were partly applied. An interface for `IERC20` by the name of `IERC20Detailed` was introduced and used in the contract. We further recommend that `IERC20Detailed` be named to `IERC20` so that only `IERC20` is used instead of both `IERC20Detailed` and `IERC20`.

SRS-02: Redundant `struct` Property

Type	Severity	Location
Optimization	Informational	StakingRewards.sol L31 StakingRewards.sol L42

Description:

The property `tokenInstance` of struct `RewardInfo` can be removed because the same property is avaiable as `key` of the mapping `rewardsTokensMap` in form of type `address`. As the mentioned mapping is the only data structure utilizing `RewardInfo` struct, so wherever the property `tokenInstance` is needed, it can be replaced by the `key` value casted to `IERC20`. This will save gas cost associated with additional `storage` slot utilization for writing and reading.

Recommendation:

We recommend to remove `tokenInstance` property of the struct `RewardInfo` and instead the `key` of mapping `rewardsTokensMap` be used by casting it to `IERC20` to save gas cost associated with `storage` operations.

Alleviation:

Alleviations were applied as advised.

SRS-03: Unnecessary Constructor Parameter

Type	Severity	Location
Optimization	Informational	StakingRewards.sol L83 StakingRewards.sol L94

Description:

The `StakingRewards` contract is always deployed by the contract `StakingRewardsFactory`. The `constructor` of `StakingRewards` has parameter `_rewardsDistribution` on `L83` which is always equal to the address of `StakingRewardsFactory`. The parameter on `L83` can be removed and the state variable `rewardsDistribution` can initialized with `msg.sender`. This will result in savings in gas cost associated with additional `constructor` parameter.

Recommendation:

We recommend to remove the `constructor` parameter on `L83` and initialize `rewardsDistribution` with `msg.sender`.

```
 constructor(
        address _rewardsDistribution,
        address[] memory _rewardsTokens,
        address _stakingToken,
  uint256 _rewardsDuration
) public {...}
```

```
rewardsDistribution = msg.sender;
```

Alleviation:

Alleviations were applied as advised.

SRS-04: Confusing Variable Name

Type	Severity	Location
Language Specific	Informational	StakingRewards.sol L26

Description:

The name of state variable `_totalSupply` can be confusing and misleading as it is generally associated with total supply of tokens of a kind but here it represents the total amount of `stakeToken` deposited by users in the contract.

Recommendation:

We recommend that the name of state variable `_totalSupply` be changed to something more close to what it has inside it f.e. totalStakesAmount.

Alleviation:

Alleviations were applied as advised.

SRS-05: Inefficient `storage` Read

Type	Severity	Location
Optimization	Informational	StakingRewards.sol L118 - L133

Description:

The function `rewardPerToken` access `storage` pointer to struct `RewardInfo` several times in its body using mapping `rewardsTokensMap`. It is inefficient to compute struct `storage` pointer multiple times.

Recommendation:

We recommend to store the struct pointer in a local `storage` variable instead of performing mapping lookup multiple times to reduce gas costs.

```
function rewardPerToken(address rewardToken) public view returns (uint256) {
    RewardInfo storage ri = rewardsTokensMap[rewardToken];
    if (_totalSupply == 0) {
        return ri.latestRewardPerTokenSaved;
    }

    uint256 timeSinceLastSave = lastTimeRewardApplicable(rewardToken).sub(
        ri.lastUpdateTime
    );

    uint256 rewardPerTokenSinceLastSave = timeSinceLastSave
        .mul(ri.rewardRate)
        .mul(1e18)
        .div(_totalSupply);

    return ri.latestRewardPerTokenSaved.add(rewardPerTokenSinceLastSave);
}
```

Alleviation:

Alleviations were applied as advised.

SRS-06: Use of Literal Value

Type	Severity	Location
Undocumented Literal	Major	StakingRewards.sol L129 StakingRewards.sol L146

Description:

The aforementioned lines make of use of literal `1e18` to represent `stakingToken` decimals multiplier. There will be incorrect calculations if any `stakingToken` contract has different decimals number than `18`.

Recommendation:

We recommend that `decimals` function is called on the `stakingContract` to get decimals and compute decimals multiplier using it.

```
uint256 rewardPerTokenSinceLastSave = timeSinceLastSave
    .mul(rewardsTokensMap[rewardToken].rewardRate)
    .mul(10 ** IERC20(stakingToken).decimals())
    .div(_totalSupply);
```

```
uint256 newReward = _balances[account]
    .mul(userRewardPerTokenSinceRecorded)
    .div(10 ** IERC20(stakingToken).decimals());
```

If it is certain that all of the staking tokens will have exactly `18` decimals then a constant can be introduced in the contract and be used instead of the literal value.

```
uint256 constant private DECIMALS_MULTIPLIER = 1e18;
```

```
uint256 rewardPerTokenSinceLastSave = timeSinceLastSave
    .mul(rewardsTokensMap[rewardToken].rewardRate)
    .mul(DECIMALS_MULTIPLIER)
    .div(_totalSupply);
```

```
uint256 newReward = _balances[account]
    .mul(userRewardPerTokenSinceRecorded)
    .div(DECIMALS_MULTIPLIER);
```

Alleviation:

Alleviations were applied as advised.

SRS-07: Inefficient `storage` Read

Type	Severity	Location
Optimization	Informational	StakingRewards.sol L139 - L149

Description:

The function `earned` access `storage` pointer to struct `RewardInfo` twice in its body using mapping `rewardsTokensMap`. It is inefficient to compute struct `storage` pointer multiple times.

Recommendation:

We recommend to store the struct pointer in a local `storage` variable instead of performing mapping lookup multiple times to reduce gas costs.

```
function earned(address account, address rewardToken) public view returns (uint256) {
    RewardInfo storage ri = rewardsTokensMap[rewardToken];
    uint256 userRewardPerTokenSinceRecorded = rewardPerToken(rewardToken).sub(
         ri.userRewardPerTokenRecorded[account]
    );

    uint256 newReward = _balances[account]
        .mul(userRewardPerTokenSinceRecorded)
        .div(1e18);

    return  ri.rewards[account].add(newReward);
}
```

Alleviation:

Alleviations were applied as advised.

SRS-08: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Optimization	Informational	StakingRewards.sol L166 StakingRewards.sol L167 StakingRewards.sol L181 StakingRewards.sol L217 StakingRewards.sol L232 StakingRewards.sol L248

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

Alleviations were applied as advised.

SRS-09: Inefficient `storage` Read

Type	Severity	Location
Optimization	Informational	StakingRewards.sol L167 StakingRewards.sol L169

Description:

The function `earned` access `storage` pointer to struct `RewardInfo` twice in its body using mapping `rewardsTokensMap`. It is inefficient to compute struct `storage` pointer multiple times.

Recommendation:

We recommend to store the struct pointer in a local `storage` variable instead of performing mapping lookup multiple times to reduce gas costs.

```
function getPeriodsToExtend(address rewardToken, uint256 rewardAmount)
    public
    view
    returns (uint256 periodsToExtend)
{
    require(rewardAmount > 0, "Rewards should be greater than zero");
    RewardInfo storage ri = rewardsTokensMap[rewardToken];
    require(ri.rewardRate > 0, "Staking is not yet started");

    uint256 periodToExtend = rewardAmount.div(ri.rewardRate);
    return periodToExtend;
}
```

Alleviation:

Alleviations were applied as advised.

SRS-10: Explicitly Returning a Local Variable

Type	Severity	Location
Optimization	Informational	StakingRewards.sol L164 StakingRewards.sol L169 StakingRewards.sol L170

Description:

The function `getPeriodsToExtend` declares and explicitly returns a `uint256` periodToExtend local variable, which increases the overall cost of gas:

```
uint256 periodToExtend = rewardAmount.div(rewardsTokensMap[rewardToken].rewardRate);
return periodToExtend;
```

Additionally, a local variable `periodsToExtend` declared in the function signature as part of return value is never used within the function.

Recommendation:

We advise to consider refactoring to remove the local variable declaration and explicit return statement in order to reduce the overall cost of gas and replace unused named return local variable `periodsToExtend` with `periodToExtend` that was being explicitly returned.

```
function getPeriodsToExtend(address rewardToken, uint256 rewardAmount)
    public
    view
    returns (uint256 periodToExtend)
{...}
```

```
periodToExtend = rewardAmount.div(rewardsTokensMap[rewardToken].rewardRate);
```

Alleviation:

Alleviations were applied as advised.

SRS-11: Incorrect Function Implementation

Type	Severity	Location
Code Legibility	Critical	StakingRewards.sol L192 - L205

Description:

The function `hasPeriodFinished` is supposed to return `true` if all of the reward tokens have their `periodFinish` reached and return `false` if a single reward token does not have its `periodFinish` reached. The function has incorrect implementation such that on `L199`, the predicate inside the `if` statement evaluates to `true` if a single reward token has its `periodFinish` reached and the function returns `false` which is wrong.

```
if (block.timestamp >= rewardsTokensMap[rewardsTokensArr[i]].periodFinish) {
    return false;
}
```

Recommendation:

We recommend to change the predicate of `if` statement such that it returns `false` when any of the reward tokens has not reached its `periodFinish`.

```
if (block.timestamp < rewardsTokensMap[rewardsTokensArr[i]].periodFinish) {
    return false;
}
```

Alleviation:

Alleviations were applied as advised.

SRS-12: Comment Discrepency

Type	Severity	Location
Comment	Informational	StakingRewards.sol L190

Description:

The comment on the aforementioned says that the function `hasPeriodFinished` returns `true` when at least one of the reward tokens does not have its `periodFinish` reached. This is incorrect as it would/should return `false` when one of the reward tokens does not have its `periodFinished` reached.

```
* Returns true if atleast one reward token has not yet finished
```

Recommendation:

We recommend to rectify the comment on the aforementioned line such that the function following returns `false` when one of the reward token does not have its `periodFinish` reached.

```
* Returns false if atleast one reward token has not yet finished
```

Alleviation:

Alleviations were applied as advised.

SRS-13: Incorrect Grammar

Type	Severity	Location
Grammar	Informational	StakingRewards.sol L198

Description:

The comment on the aforementioned line has incorrect grammar.

```
// on first token which period has not been expired return false
```

Recommendation:

We advise to rectify the grammar of the comment.

```
// on first token for which the period has not expired, returns false.
```

Alleviation:

Alleviations were applied as advised.

SRS-14: Inefficient Read from `storage`

Type	Severity	Location
Optimization	Informational	StakingRewards.sol L246

Description:

The `for` loop on the aforementioned line reads a `storage` slot `rewardsTokensArr` in its predicate. As this comparison part of the `for` loop is executed repeatedly and hence the same `storage` slot is read multiple times rendering the implementation inefficient.

```
for (uint i = 0; i < rewardsTokensArr.length; i++) {...}
```

Recommendation:

We recommend to store length of `rewardsTokenArr` read `storage` to be stored in a local variable and that be used in the predicate of `for` loop, so the gas cost from repeated `storage` read could be saved.

```
uint256 tokenArrLength = rewardsTokensArr.length;
for (uint i = 0; i < tokenArrLength; i++) {...}
```

Alleviation:

Alleviations were applied as advised.

SRS-15: Inefficient `storage` Read

Type	Severity	Location
Optimization	Informational	StakingRewards.sol L

Description:

The function `getReward` access `storage` pointer to struct `RewardInfo` several times in its body using mapping `rewardsTokensMap`. It is inefficient to compute struct `storage` pointer multiple times. Additionaly, it accesses `storage` twice to read address of reward token at `rewardsTokensArr[i]`,

Recommendation:

We recommend to store the struct pointer in a local `storage` variable instead of performing mapping lookup multiple times to reduce gas costs. And also the address read for reward token from `rewardsTokensArr[i]` can be stored in a local variable of type `address`.

```
function getReward()
    public
    nonReentrant
    updateReward(msg.sender)
{
    for (uint i = 0; i < rewardsTokensArr.length; i++) {
        address token = rewardsTokensArr[i];
        RewardInfo storage ri = rewardsTokensMap[token];
        uint256 reward = ri.rewards[msg.sender];
        if (reward > 0) {
            ri.rewards[msg.sender] = 0;
            ri.tokenInstance.safeTransfer(msg.sender, reward);
            emit RewardPaid(msg.sender, token, reward);
        }
    }
}
```

Alleviation:

Alleviations were applied as advised.

SRS-16: Spelling Error

Type	Severity	Location
Comment	Informational	StakingRewards.sol L265

Description:

The comment on the aformentioned line has incorrect spelling for the word `staking`.

```
/** @dev Makes the needed calculations and starts the starking/rewarding.
```

Recommendation:

We recommend to recitify the incorrect spelling.

```
/** @dev Makes the needed calculations and starts the staking/rewarding.
```

Alleviation:

Alleviations were applied as advised.

SRS-17: Redundant `require` statement

Type	Severity	Location
Code Legibility	Minor	StakingRewards.sol L274 - L277

Description:

The function `start` is called from `StakingRewardsFactory` contract's function `startStaking`. The `require` statement on the aforementioned line is redundant as the same check is performed in `startStaking` function of `StakingRewardsFactory` on `L141`.

Recommendation:

We recommend to remove the `require` check on the aforementioned line as it is redundant.

Alleviation:

Alleviations were applied as advised.

SRS-18: Unnecessary Function Parameter

Type	Severity	Location
Optimization	Informational	StakingRewards.sol L269

Description:

The function `start` on the aforementioned line has parameter `_rewardsTokens` which can be omitted and instead `rewardsTokensArr` can be read from `storage` of the contract which will have exactly the same data. There might not be any significant benefit with respect to gas cost but it should make the code legible.

Recommendation:

We recommend to remove the said parameter from the function signature and instead it read from the from `storage` using `rewardsTokensArr`.

Alleviation:

Alleviations were applied as advised.

SRS-19: Spelling Error

Type	Severity	Location
Comment	Informational	StakingRewards.sol L302

Description:

The comment on the aforementioned line has incorrect spelling for the word `distributed`.

Recommendation:

We advise to rectify the spellings.

Alleviation:

Alleviations were applied as advised.

SRS-20: Returned Value of Function is not Checked

Type	Severity	Location
Code Legibility	Informational	StakingRewards.sol L311

Description:

The `transferFrom` call on the aforementioned does not have its returned value checked which can lead to inaccuracies if the call returns `false` and it will be treated as successful in the context of current execution.

```
IERC20(rewardToken).transferFrom(msg.sender, address(this), rewardAmount);
```

Recommendation:

We recommend to `safeTransferFrom` function from the library `SafeERC20` which is already imported in the contract.

```
IERC20(rewardToken).safeTransferFrom(msg.sender, address(this), rewardAmount);
```

Alleviation:

Alleviations were applied as advised.

SRS-21: Usage of both `now` and `block.timestamp`

Type	Severity	Location
Code Legibility	Informational	StakingRewards.sol L315

Description:

The aforementioned line uses global variable `now` to read epoch time. All the other such instances in the contract use `block.timestamp` to read epoch time.

```
emit RewardExtended(rewardToken, rewardAmount, now, periodToExtend);
```

Recommendation:

We recommend to use `block.timestamp` instead of `now` on the aforementioned line make the code legible and consistent.

```
emit RewardExtended(rewardToken, rewardAmount, block.timestamp, periodToExtend);
```

Alleviation:

Alleviations were applied as advised.

SRS-22: Incorrect Manipulation of State Variable

Type	Severity	Location
Code Legibility	Critical	StakingRewards.sol L155 StakingRewards.sol L313

Description:

The function `addRewards` rewards extends reward duration for a specific reward token by increasing it `periodFinish` property. However, on `L313` the state variable `rewardsDuration` is also increased by a certain amount which results in the function `getRewardForDuration` returning correct amount on `L155` only for the reward token for which `addRewards` was executed and for the rest it will return incorrect amounts.

```
return rewardsTokensMap[rewardToken].rewardRate.mul(rewardsDuration);
```

Recommendation:

We recommend to add `rewardsDuration` as property of the struct `RewardInfo` and it be increased for only the reward token for which the `addRewards` is executed.

```
struct RewardInfo {
    uint256 rewardDuration;
}
```

```
rewardsTokensMap[rewardToken].rewardDuration =
rewardsTokensMap[rewardToken].rewardDuration.add(periodToExtend);
```

The function `getRewardForDuration` will be rectified to read `rewardDuration` specific to the reward token from its struct.

```
function getRewardForDuration(address rewardToken) external view returns (uint256) {
    RewardInfo storage ri = rewardsTokensMap[rewardToken];
    return ri.rewardRate.mul(ri.rewardsDuration);
}
```

Alleviation:

Alleviations were applied as advised.

SRS-23: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	StakingRewards.sol L2

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.5.16` the contract should contain the following line:

```
pragma solidity 0.5.16;
```

Alleviation:

Alleviations were applied as advised.

SRS-24: Incorrect Order of Functions

Type	Severity	Location
Language Specific	Informational	StakingRewards.sol

Description:

The structure of the codebase does not conform to the official Solidity style guide of `v0.5.16`

Recommendation:

Functions should be grouped according to their visibility and ordered:

```
constructor
fallback function (if exists)
external
public
internal
private
```

Within a grouping, place the `view` and `pure` functions last.

Alleviation:

Alleviations were partly applied.

SRS-25: Visibility can be changed from `public` to `external`

Type	Severity	Location
Optimization	Informational	StakingRewards.sol L45 StakingRewards.sol L53 StakingRewards.sol L61 StakingRewards.sol L192

Description:

The function on the aforementioned lines can have their visibilities changed from `public` to `external` as they are never called from within the contract.

Recommendation:

We recommend to change the visibilities of functions on the aforementioned lines from `public` to `external`.

Alleviation:

Alleviations were applied except for `L192` which is on `L311` in the post-audit commit.

SRF-01: Inefficient `storage` Layout

Type	Severity	Location
Optimization	Informational	StakingRewardsFactory.sol L17 - L25

Description:

The mappings on the aforementioned lines have same `key` type of `stakingToken` and can be condensed with their value types declared in a `struct`. This will greatly reduce the mappings lookup gas cost in the function where the read operations are performed on two or more mappings.

Recommendation:

We recommend to introduce a `struct` type which will have all the values types of the mappings as its keys.

```
struct StakingInfo {
  address stakingReward;
  address[] rewardsTokens;
  uint256[] rewardsAmount;
}
```

```
mapping(address => StakingInfo) public stakingInfoMap;
```

Alleviation:

The concerned code was removed as an alleviation for another exhibit, so this exhibit is longer applicable.

SRF-02: Usage of `uint` alias instead of `uint256`

Type	Severity	Location
Language Specific	Informational	StakingRewardsFactory.sol

Description:

The contract is using `uint` to declare 256-bit unsigned integers. Although, `uint` is an alias for `uint256` and both represent the same underlying integer allocation. It is advisable that for clean coding practices the complete form `uint256` should be used instead of the alias `uint`.

Recommendation:

We advise to use `uint256` instead of alias `uint` in all of the occurrences in the contract.

Alleviation:

Alleviations were applied as advised.

SRF-03: Unnecessary explicit usage of modifier `Ownable`

Type	Severity	Location
Optimization	Informational	StakingRewardsFactory.sol L42

Description:

The explicit usage of `Ownable()` to call constructor of the `Ownable` contract is unnecessary as the constructor does not expect arguments and is called implicitly nevertheless.

```
constructor(
    uint _stakingRewardsGenesis
) Ownable() public {...}
```

Recommendation:

We recommend to remove the explit usage of modifier `Ownable` as it is implicitly called.

```
constructor(
    uint _stakingRewardsGenesis
) public {...}
```

Alleviation:

Alleviations were applied as advised.

SRF-04: Visibility can be changed from `public` to `external`

Type	Severity	Location
Optimization	Informational	StakingRewardsFactory.sol L56

Description:

The function `deploy` on the aforementioned line can be declared `external` as it is never called internally from within the contract. This function when declared `external` would be cheaper to execute as its arguemnts of `_rewardsTokens` and `_rewardsAmounts` would reside in `calldata` which is cheaper to use than `memory`

Recommendation:

We recommmed to change visibility of the aforementioned function to `external` and accordingly change the data location of its `memory` parameters to `calldata`.

```
function deploy(
    address           _stakingToken,
    address[] calldata _rewardsTokens,
    uint[]    calldata _rewardsAmounts,
    uint              _rewardsDuration
) public onlyOwner {...}
```

Alleviation:

Alleviations were applied as advised.

SRF-05: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Optimization	Informational	StakingRewardsFactory.sol L63 StakingRewardsFactory.sol L64 StakingRewardsFactory.sol L69 StakingRewardsFactory.sol L92 StakingRewardsFactory.sol L101 StakingRewardsFactory.sol L112

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

Alleviations were applied as advised.

SRF-06: Ineffectual Predicate in `require` Statement

Type	Severity	Location
Code Legibility	Informational	StakingRewardsFactory.sol L64 - L65

Description:

Any of the comparison in `require` statement on `L64` can be removed because the `L65` does the equality comparison between both which makes one of the comparisons redundant in the `require` statement on `L64`.

```
require(_rewardsTokens.length > 0 && _rewardsAmounts.length > 0);
```

Recommendation:

We recommend that any one of the comparison be removed to save gas associated with execution of redundant code.

```
require(_rewardsTokens.length > 0);
```

Alleviation:

Alleviations were applied as advised.

SRF-07: Spelling Error

Type	Severity	Location
Comment	Informational	StakingRewardsFactory.sol L130

Description:

The comment on the aforementioned line incorrectly refers to function `startStakings()` as `startsStakings()`.

Recommendation:

We advise correct the reference to function by provifing its correct name.

Alleviation:

Alleviations were applied as advised.

SRF-08: Reward Tokens and Amounts can be moved to `StakingRewards` Contract

Type	Severity	Location
Code Legibility	Informational	StakingRewardsFactory.sol L21 - L25

Description:

The `StakingRewardsFactory` contract in addition to acting as factory for `StakingRewards` contract also stores reward tokens and amounts corresponding to staking tokens. To observe separation of concern and increase legibility and quality of the codebase, the reward amounts can be moved `StakingRewards` contract as reward tokens are already stored in the contract.

Recommendation:

We recommend to move the reward amounts to the corresponding `StakingRewards` contract by passing the values through constructor and any utlization of reward amount can be accessed through quering the relevent `StakingRewards` contract.

Alleviation:

Alleviations were applied as advised.

SRF-09: Explicit Passing of `msg.sender` can be Omitted

Type	Severity	Location
Optimization	Informational	StakingRewardsFactory.sol L72

Description:

The aforementioned line explicitly pass `msg.sender` as argument to the constructor of `StakingRewards` contract. Taking into the exhibit `SRS-03`, the parameter can be removed.

Recommendation:

We recommend to remove the unnecessary constructor argument of `msg.sender` on the aformentioned line as part of exhibit `SRS-03`.

Alleviation:

Alleviations were applied as advised.

SRF-10: Incorrect Order of Functions

Type	Severity	Location
Language Specific	Informational	StakingRewardsFactory.sol

Description:

The structure of the codebase does not conform to the official Solidity style guide of `v0.5.16`

Recommendation:

Functions should be grouped according to their visibility and ordered:

```
constructor
fallback function (if exists)
external
public
internal
private
```

Within a grouping, place the `view` and `pure` functions last.

Alleviation:

Alleviations were partly applied.

SRF-11: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	StakingRewardsFactory.sol L2

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.5.16` the contract should contain the following line:

```
pragma solidity 0.5.16;
```

Alleviation:

Alleviations were applied as advised.

SRF-12: Returned Value of Function is not Checked

Type	Severity	Location
Code Legibility	Informational	StakingRewardsFactory.sol L103

Description:

The `approve` call on the aforementioned does not have its returned value checked which can lead to inaccuracies if the call returns `false` and it will be treated as successful in the context of current execution.

```
tkn.approve(sr, extendRewardAmount);
```

Recommendation:

We recommend to use `safeApprove` function from the library `SafeERC20`.

```
tkn.safeApprove(sr, extendRewardAmount);
```

Alleviation:

Alleviations were applied as advised.

SRF-13: Visibility can be changed from `public` to `external`

Type	Severity	Location
Optimization	Informational	StakingRewardsFactory.sol L27 StakingRewardsFactory.sol L84 StakingRewardsFactory.sol L111

Description:

The function on the aforementioned lines can have their visibilities changed from `public` to `external` as they are never called from within the contract.

Recommendation:

We recommend to change the visibilities of functions on the aforementioned lines from `public` to `external`.

Alleviation:

Alleviations were applied as advised.

TEC-01: Usage of `uint` alias instead of `uint256`

Type	Severity	Location
Language Specific	Informational	TestERC20.sol L8

Description:

The contract is using `uint` to declare 256-bit unsigned integers. Although, `uint` is an alias for `uint256` and both represent the same underlying integer allocation. It is advisable that for clean coding practices the complete form `uint256` should be used instead of the alias `uint`.

Recommendation:

We advise to use `uint256` instead of alias `uint` in all of the occurrences in the contract.

Alleviation:

Alleviations were applied as advised.