AUDITONE

Audit Report

ShimmerSea

Created on: 11.10.2022

Audit Report

by AuditOne

Smart Contract Security Analysis Report

Note: This report may contain sensitive information on potential vulnerabilities and exploitation methods. This must be referred internally and should be only made available to the public after issues are resolved (to be confirmed prior by the client and AuditOne).

Table of Contents

Introduction

Csanuragjain, Hrishibhat, Minhquanym and Raja, who are auditors at AuditOne, successfully audited the smart contracts of ShimmerSea. The audit has been performed using manual analysis. This report presents all the findings regarding the audit performed on the customer's smart contracts. The report outlines how potential security risks are evaluated. Recommendations on quality assurance and security standards are provided in the report.

Project Description

ShimmerSea is a leading decentralized exchange (DEX) on Shimmer focused on offering a premier trading experience.

The ultimate Decentralized Exchange (DEX) to launch a fully decentralized automated market maker service without transaction costs and low fees on the Shimmer Network**.** ShimmerSea is developed and powered by the TangleSea Team. It will function as a testing environment to validate the TangleSea MVP. In addition, it will serve as a pilot platform to explore future advanced features before we launch them with TangleSea on the Assembly/IOTA Network. That being said, ShimmerSea will become more than just a testing ground for us. ShimmerSea is meant to be an ecosystem accelerator.

Project and Audit Information

Term	Description
Auditor	Csanuragjain, Hrishibhat, Minhquanym and Raja
Reviewed by	Tomo
Type	Advanced Uniswap V2, ERC20
Language	Solidity
Ecosystem	ShimmerEVM / BSC testnet for this audit
Methods	Manual Review
Repository	https://github.com/ShimmerSea/shimmersea-contracts
Commit hash (at audit start)	cd030531e3a6b338e4dddc281570b3f1f0bd2c37
Commit hash (after resolution)	ed879be8b1c80c6d999626dd034388cd7a9a5786
Documentation	[Added once the whitepaper is published by the project]
Unit Testing	No
Website	https://shimmersea.finance/
Submission Time	2022-09-19
Finishing Time	2022-10-11

Contracts in scope

- *contracts/LumToken.sol*
- *contracts/Migrations.sol*
- *contracts/PearlToken.sol*
- *contracts/TangleSeaMasterChef.sol*
- *contracts/zaps/FarmUniV2Zap.sol*
- *contracts/pools/RestrictedLumPool.sol*
- *contracts/pools/RewardPool.sol*
- *contracts/misc/FeeVault.sol*
- *contracts/misc/WETH9.sol*
- *main/contracts/interfaces/IMasterChef.sol*
- *contracts/interfaces/IRewardToken.sol*
- *contracts/interfaces/IRewarder.sol*
- *contracts/dex/TangleseaERC20.sol*
- *contracts/dex/TangleseaFactory.sol*
- *contracts/dex/TangleseaPair.sol*
- *contracts/dex/TangleseaRouter.sol*
- *contracts/dex/libraries/Babylonian.sol*
- *contracts/dex/libraries/Math.sol*
- *contracts/dex/libraries/SafeMath.sol*
- *contracts/dex/libraries/TangleseaLibrary.sol*
- *contracts/dex/libraries/TransferHelper.sol*
- *contracts/dex/interfaces/IERC20.sol*
- *contracts/dex/interfaces/ITangleseaCallee.sol*
- *contracts/dex/interfaces/ITangleseaERC20.sol*
- *contracts/dex/interfaces/ITangleseaFactory.sol*
- *contracts/dex/interfaces/ITangleseaPair.sol*
- *contracts/dex/interfaces/ITangleseaRouter01.sol*
- *contracts/dex/interfaces/ITangleseaRouter02.sol*
- *contracts/dex/interfaces/IWETH.sol*

Executive Summary

ShimmerSea's smart contracts were audited between 2022-09-19 and 2022-10-11 by Csanuragjain, Hrishibhat, Minhquanym and Raja. Manual analysis was carried out on the code base provided by the client. The following findings were reported to the client. For more details, refer to the findings section of the report.

S.no.	Issue Category	Issues found	Resolved	Acknowledged
1.	High	1	1	0
2.	Medium	7	6	1
3.	Low	8	6	2
4.	Quality Assurance	11	10	1

Severity definitions

Risk factor matrix	Low	Medium	High
Occasional	L	M	H
Probable	L	M	H
Frequent	M	H	H

High: Funds or control of the contracts might be compromised directly. Data could be manipulated. We recommend fixing high issues with priority as they can lead to severe losses.

Medium: The impact of medium issues is less critical than high, but still probable with considerable damage. The protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions.

Low: Low issues impose a small risk on the project. Although the impact is not estimated to be significant, we recommend fixing them on a long-term horizon. Assets are not at risk: state handling, function incorrect as to spec, issues with comments.

Quality Assurance: Informational and Optimization - Depending on the chain, performance issues can lead to slower execution or higher gas fees. For example, code style, clarity, syntax, versioning, off-chain monitoring (events etc.)

Audit Overview

Code quality: 95.0

Security score: 97.6

Documentation quality: 98.0

Architecture quality: Not in scope

Audit Findings

Finding: #1

Issue: Possibility of token lock with owner calling function more than once

Severity: High

Where:

https://github.com/ShimmerSea/shimmersea-fairlaunch/blob/main/contracts/FairLaunchERC20.sol#L223

Impact: All LP token will be locked in `FairLauchERC20` forever if owner calls `buildLP()` more than once.

Description: The "In `FairLauchERC20`, after sale ended, owner should call `buildLP()` function to build liquidity pool for LUM-SRM. In this function, `totalLPAmountToClaim` is set to returned value from `ITangleseaPair.mint()` function.

But it did not check if lp is already built or not. So in case owner call this function second time, `ITangleseaPair.mint()` will return 0 which mean `totalLPAmountToClaim = 0` and users cannot claim any lp token back."

Recommendations: Consider adding check if `isLpBuilt = false`

```
require(hasEnded() && !isLpBuilt, ""buildLP: sale has not ended"");
```

Status: Resolved

Finding: #2

Issue: Dev Reward fee is minted separately

Severity: Medium

Where:

https://github.com/ShimmerSea/shimmersea-contracts/blob/main/contracts/TangleSeaMasterChef.sol#L551

Impact: Dev rewards are minted rewards separately than main reward

Description: In _updatePool function, it was observed that devRewardFee was not inclusive of total reward

Recommendations: Dev reward should be part of main reward. It could be revised like below:

```
uint256 rewards =nbSeconds.mul(rewardsPerSec).mul(pool.allocPoint).div(totalAllocPoint);
uint256 devReward=rewards.mul(devRewardFee).div(10000);
uint256 rewardToBeDistributed=rewards-devReward;
rewardToken.mint(opsInfraAddr,devReward);
rewardToken.mint(address(this), rewardToBeDistributed);
uint256 adjustedAccRewardPerShare = pool.accRewardPerShare.add(rewardToBeDistributed.mul(PRECISION_FACTOR).div(lpSupply));
```

Status: Resolved

Finding: #3

Issue: Miscalculation of rewards

Severity: Medium

Where:

https://github.com/ShimmerSea/shimmersea-contracts/blob/main/contracts/TangleSeaMasterChef.sol#L153

Impact: Pool rewards could be affected in case `_withUpdate` is false.

Description: totalAllocPoint` is used to determine the portion of reward each pool is entitled to. So, when `totalAllocPoint` is modified it is important to update the pool rewards until that point. If not, the rewards will be calculated incorrectly.

For example, when `_withUpdate` is false, in the `add()` or `set()` functions, the totalAllocPoint is changed without updating the rewards `massUpdatePools()`.

Recommendations: Consider removing the parameter `_withUpdate`

Status: Acknowledged

Finding: #4

Issue: Owner has privileged access that can be used to make changes without any delay.

Severity: Medium

Where:

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/TangleSeaMasterChef.sol#L253

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/misc/FeeVault.sol#L57

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/pools/RestrictedLumPool.sol#L281

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/pools/RewardPool.sol#L248
https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/pools/RewardPool.sol#L372
https://github.com/ShimmerSea/shimmersea-fairlaunch/blob/main/contracts/FairLaunchERC20.sol#L211

Impact: Owner has privileged access to make changes without any delay or reaction time for users.

Description:

There are contracts that contain functions that change important parameters of the system, e.g. None of these functions emit events, nor they are timelocked. Usually, it is a good practice to give time for users to react and adjust to changes.

Reference:
https://code4rena.com/reports/2022-05-alchemix/#m-05-no-storage-gap-for-upgradeable-contract-might-lead-to-storage-slot-collision

Recommendations: Use a timelock to avoid instant changes of the parameters. And use appropriate events for these important changes. Emit events for important changes.

Status: Resolved

Finding: #5

Issue: Users might be unable to burn LP token in case pool is extremely imbalanced

Severity: Medium

Where:
https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/dex/TangleseaPair.sol#L178

Impact: Users might be unable to burn LP token in case pool is extremely imbalanced.

Description: In `TangleseaPair.burn()` function, it requires that amount out of both tokens must bigger than 0. So in case the pool is extremely imbalanced, then the rouding down calculation could make one of the amounts equal to 0 and `burn()` revert.

For example, in case of extreme condition, like UST collapse, there will be a pool have almost a few Wei of token compared to millions of UST token on the other side (E.g. `1 billion UST ~ 10 USDC`). Now if users try to burn an amount of LP token equivalent to `amount0 = 100 UST`, the rounding down in calculation could make `amount1 = 0 DAI` and revert."

Recommendations: Consider change the check condition from `AND` to `OR`

```
require(amount0 > 0 || amount1 > 0, ""Tanglesea: INSUFFICIENT_LIQUIDITY_BURNED"");
```

Status: Resolved

Finding: #6

Issue: Bypass Deposit restriction in RestrictedLumPool

Severity: Medium

Where:

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/pools/RestrictedLumPool.sol#L137

Impact: Unauthorized users will be able to make deposits.

Description:

1. Any Deposit in RestrictedLumPool.sol requires the user to have ownership of LumiNFT
2. This particular setup can be tricked simply by using transferFrom function
3. Assume User A who is in possession of Lumi NFT makes a deposit using ""deposit"" function
4. After making the deposit, he transfers the Lumi NFT to User B
5. Since User B now holds ownership of Lumi NFT so he would also be able to use the ""deposit"" function
6. User B can repeat the same with third user and so on
7. Thus the restriction placed is simply bypassed and all these Users will be able to enjoy staking on their LUM token, using a single LumiNFT

Recommendations: If a LUMI NFT has been used already then only allow it to be reused post X timestamp. Although this does not remove the attack vector completely but could reduce it. Remember this solution also brings one problem where a genuine new buyer of LUMI NFT will be prohibited from deposit for certain timestamp.

Status: Resolved

Finding: #7

Issue: Incorrect Rewards are distributed

Severity: Medium

Where:

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/pools/RewardPool.sol#L286

Impact: The User will get more reward than expected.

Description:

1. The updateRewardPerSecond function is used to update the reward distribution per second for an existing pool

```
function updateRewardPerSecond(uint256 _rewardsPerSec) external onlyOwner {
        require(_rewardsPerSec <= MAX_EMISSION_RATE_PER_SEC, ""update: emission rate too high"");
        rewardsPerSec = _rewardsPerSec;
        emit UpdateEmissionRate(msg.sender, _rewardsPerSec);
    }
```

2. As we can see above, ther is no call to _updatePool function which means this reward change will also impact the past deposits even though it should only be applicable for future deposits
3. An example is User deposits amount 50 and keeps this for 36000 seconds. For this period rewardsPerSec was 10
4. Now if this user has withdrawn he would have got 50*36000*10 but what if before user withdraw, Admin makes call to updateRewardPerSecond and sets rewardsPerSec to 20
5. In this case the reward distributed will be 50*36000*20 which is double. This is incorrect ideally the new rewardsPerSec should have been in effect from current time and not for past.

Other Instance:

updateDevRewardFee function (updatePool not called):

https://github.com/ShimmerSea/shimmersea-contracts/blob/main/contracts/TangleSeaMasterChef.sol#L233

Recommendations: Kindly change this as below:

```
function updateRewardPerSecond(uint256 _rewardsPerSec) external onlyOwner {
_updatePool();
        require(_rewardsPerSec <= MAX_EMISSION_RATE_PER_SEC, ""update: emission rate too high"");
        rewardsPerSec = _rewardsPerSec;
        emit UpdateEmissionRate(msg.sender, _rewardsPerSec);
    }
```

Status: Resolved

Finding: #8

Issue: Calculation in `_mintFee()` is not too precise

Severity: Medium

Where:

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/dex/TangleseaPair.sol#L128-L134

Impact: Malicious calls can try to gain ownership of contract by calling unprotected `initialize` functions or by upgrading implementation's address.

Description: "In `_mintFee()` function, calculation of `d` value is a division without precision, it can lead to precision loss.

```
if (rootK > rootKLast) {
    uint256 d = (FEE_DENOMINATOR / ITangleseaFactory(factory).ownerFeeShare()).sub(1);
    uint256 numerator = totalSupply.mul(rootK.sub(rootKLast));
    uint256 denominator = rootK.mul(d).add(rootKLast);
    uint256 liquidity = numerator / denominator;
    if (liquidity > 0) _mint(feeTo, liquidity);
}
```

For example, if `ownerFeeShare()` is `50000` or `35000` doesn't matter since `d` will be `1` in both case since `FEE_DENOMINATOR / ownerFeeShare()` rounded down to 2 in both case."

Recommendations: Use precision factor for division

Status: Resolved

Finding: #9

Issue: Owner can withdraw contract funds in an emergency scenario

Severity: Low

Where:

https://github.com/ShimmerSea/shimmersea-fairlaunch/tree/main/contracts/FairLaunchERC20.sol#L261

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/misc/FeeVault.sol#L48

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/pools/RewardPool.sol#L248

Impact: Funds of the FairLaunch would be withdrawn.

Description:

1. It is mentioned that emergencyOperator is set of trusted partners and any execution by emergencyOperator is behind multisig which prevents fraud using emergencyWithdrawFunds function
2. But still there is always a chance of collaboration of 51% of these trusted partners to execute emergencyWithdrawFunds function in order to drain funds
3. This function will discourage investors because of possible rugpull at any stage of fair launch
4. Similarly in FeeVault, Admin can rugPull all fees collected for Fee Manager
5. The same is also possible in RewardPool where Owner can steal all yields from the contract

Recommendations: Verify all scenarios for which this function has been built and decide this function existence as per risk of not having one.

Status: Acknowledged

Finding: #10

Issue: Missing Zero address Validation

Severity: Low

Where:

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/TangleSeaMasterChef.sol#L133

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/pools/RestrictedLumPool.sol#L74

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/pools/RewardPool.sol#L95

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/dex/TangleseaFactory.sol#L67 (Also contructor)

https://github.com/ShimmerSea/shimmersea-fairlaunch/tree/main/contracts/FairLaunchERC20.sol#L51

Impact: Zero address could cause loss of funds by transferring to incorrect address.

Description: Zero address validation is missing in for multiple variable in listed contract files.

Recommendations: Add a require check to see that variables are not address(0).

Status: Resolved

Finding: #11

Issue: Input Validation missing

Severity: Low

Where:

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/zaps/FarmUniV2Zap.sol#L200

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/zaps/FarmUniV2Zap.sol#L185

https://github.com/ShimmerSea/shimmersea-fairlaunch/tree/main/contracts/FairLaunchERC20.sol#L211

Impact: Input are not validated properly which could produce unexpected results in contract.

Description:

1. In _getMasterChefPair function, there is no check to validate that passed pid is valid.
2. Most functions at FarmUniV2Zap.sol are missing a check to validate if WETH passed is valid.
3. The setUsersAllocation is missing emit event emission.

Recommendations:

1. In _getMasterChefPair function, Add a check masterChef.poolLength()>pid.
2. Update the constructor of FarmUniV2Zap.sol to include checkWETH function.
3. Add emit event in setUsersAllocation function (FairLaunchERC20.sol).

Status: Resolved

Finding: #12

Issue: Approval event is not emitted.

Severity: Low

Where:

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/dex/TangleseaERC20.sol#L87

Impact: In `TangleseaERC20.sol`contract `transferFrom()` function does not emit Approval event.

Description: The ERC-20 standard specifies that an approval event should beemitted when the allowance of a user changes. In the `transferFrom()` function in `TangleseaERC20.sol` even though the allowance is changed, `Allowance` event is not emitted.

Recommendations: Consider adding emit Approval() in transferFrom when allowance is modified.

Status: Resolved

Finding: #13

Issue: Critical changes not tracked

Severity: Low

Where:

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/pools/RestrictedLumPool.sol#L285, #L288

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/pools/RewardPool.sol#L413, #L416

Impact: No track of off chain changes

Description: has no event, so it is difficult to track off-chain changes.

Recommendations: Emit an event for critical parameter changes.

Status: Resolved

Finding: #14

Issue: Contract vulnerable to Replay attack

Severity: Low

Where:

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/dex/TangleseaERC20.sol#L29

Impact: Replay attack will allow transaction to be replayed

Description:

1. In an event of forking, the chainId will not get updated, since chainId is initialized at constructor only.

2. This means the forked chain will have incorrect chain id and hence incorrect DOMAIN_SEPARATOR

Recommendations: Create a new function which computes the chainId everytime so that DOMAIN_SEPARATOR is updated correctly.

```
function getChainId() external returns (uint256 chainId){
        assembly {
            chainId := chainid()
        }
    }

function getDOMAINSEPARATOR() external returns (uint256 chainId){
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256(""EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)""),
                keccak256(bytes(name)),
                keccak256(bytes(""1"")),
                getChainId(),
                address(this)
            )
        );
    }
```

Status: Resolved

Finding: #15

Issue: LUM rewards are not reused

Severity: Low

Where:

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/pools/RestrictedLumPool.sol#L253

Impact: Rewards will get wasted and locked in contract.

Description:

1. User can make use of emergencyWithdraw in order to recover their principal without caring about rewards

2. But in this case these rewards will simply lye in the contract without any usage

Recommendations:

1. Create a new variable in emergencyWithdraw which keeps track of lost reward amount and later add this reward amount in the pendingRewards from MasterChef in _updatePool function.
2. Or simply deposit these rewards in MasterChef.

Status: Resolved

Finding: #16

Issue: Lp contract can withdraw treasury balance.

Severity: Low

Where:

https://github.com/ShimmerSea/shimmersea-fairlaunch/tree/main/contracts/FairLaunchERC20.sol#L232

Impact: Treasury amount would be withdrawn

Description:

1. It seems Reentrancy attack is possible in case lpToClaim address is malicious
2. While buildLP is called, mint function from lpToClaim address is called
3. Now if lpToClaim address is malicious and it reenters the buildLP function, it will transfer the lpSMRAmount again to the lpToClaim address
4. Calling this loop recursively will cause full SMR balance to be drained to this malicious contract so that nothing is left for treasury.

Recommendations: Add nonReentrant modifier from openzepplin which prevents Reentrancy attacks. Also add a check require(!isLpBuilt).

Status: Acknowledged

Finding: #17

Issue: Rounding vulnerabilities for token with large supply

Severity: Quality Assurance

Where:

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/TangleSeaMasterChef.sol#L449

Impact: Tokens with a large supply can cause them to receive zero emissions.

Description: In the `pendingRewards()` and `_updatePools()` accRewardPerShare is calculated. If `lpSupply` exceeds to a large value, it will cause a precision error due to rounding. This can happen when pools decide to include tokens with large supplies and no decimals.

Recommendations: In case such tokens are a possibility consider increasing the PRECISION_FACTOR to `1e18`.

Status: Resolved

Finding: #18

Issue: msg.sender is unnecessarily cast to address(msg.sender)

Severity: Quality Assurance

Where:

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/TansgleSeaMasterChef.sol

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/pools/RestrictedLUMPool.sol

https://github.com/ShimmerSea/shimmersea-fairlaunch/tree/main/contracts/FairLaunchERC20.sol

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/pools/RewardPool.sol

Impact: NA

Description: The msg.sender is cast to address(msg.sender) throughout the contracts.

Recommendations: Consider replacing all occurrences of address(msg.sender) with msg.sender.

Status: Resolved

Finding: #19

Issue: Gas consumption not optimized

Severity: Quality Assurance

Where:

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/TangleSeaMasterChef.sol#L496-L501

Impact: Extra gas consumption

Description: NA

Recommendations: Should use cached `amount` instead of reading storage to save gas

Status: Resolved

Finding: #20

Issue: Gas consumption not optimized

Severity: Quality Assurance

Where:
https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/TangleSeaMasterChef.sol#L206-L211

Impact: Extra gas consumption

Description: NA

Recommendations: Should only write new `allocPoint` value only when it is changed too.

Status: Resolved

Finding: #21

Issue: Gas consumption not optimized

Severity: Quality Assurance

Where:
https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/dex/TangleseaPair.sol#L89-L96

Impact: Extra gas consumption

Description: NA

Recommendations: Should use `newFeeAmount` instead of `feeAmount` when emitting event to save gas.

Status: Resolved

Finding: #22

Issue: Gas consumption not optimized

Severity: Quality Assurance

Where:
https://github.com/ShimmerSea/shimmersea-fairlaunch/tree/main/contracts/FairLaunchERC20.sol#L161-L171

Impact: Extra gas consumption

Description: In `buy()` function, data of user is repeatedly used in calculation (`user.maxAllocation` is read 2 times, `user.allocation` is read 2 times, written once).

Recommendations: Should load whole userInfo to stack since it is used repeatedly in `FairLauchERC20.buy()`

Status: Resolved

Finding: #23

Issue: Gas consumption not optimized

Severity: Quality Assurance

Where:

https://github.com/ShimmerSea/shimmersea-fairlaunch/tree/main/contracts/FairLaunchERC20.sol#L287

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/pools/RestrictedLumPool.sol#L50

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/TangleSeaMasterChef.sol#L139

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/pools/RestrictedLumPool.sol#L90

Impact: Redundant zero initialization

Description: Solidity does not recognize null as a value, so bool variables are initialized to false. Setting a bool variable to false is redundant and can waste gas.

Recommendations: Avoid redundant zero initializations

Status: Resolved

Finding: #24

Issue: Stale Comments

Severity: Quality Assurance

Where:

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/dex/libraries/TangleseaLibrary.sol#L32

Impact: Unclean code

Description: NA

Recommendations: Remove unnecessary comments

Status: Resolved

Finding: #25

Issue: Typo in ITangleseaCallee

Severity: Quality Assurance

Where:

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/dex/interfaces/ITangleseaCallee.sol#L4-L12

Impact: related functionality affected

Description: Using `pancakeCall()` in `ITangleseaCallee()` interface

Recommendations: Fix the typo

Status: Resolved

Finding: #26

Issue: No track of old rewards

Severity: Quality Assurance

Where:

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/TangleSeaMasterChef.sol#L214

Impact: Old rewards could be lost

Description: If the _rewarder is changed in set function,there is no tracking to old reward collected at old rewarder contract.

Recommendations: Add a facility through UI which enables user to access funds from old Rewarder contract as well if not already present.

Status: Acknowledged

Finding: #27

Issue: Constructor inputs lack a zero-address input verification

Severity: Quality Assurance

Where:

https://github.com/ShimmerSea/shimmersea-contracts/tree/main/contracts/dex/TangleseaRouter.sol#L24-26

Impact: NA

Description: Inadvertently inputting zero address in the constructor will render the contract unusable and will require redeployment.

Recommendations: Consider adding

```
require (_factory != address(0), ""Cannot be zero address"" );
require (_WETH != address(0), ""Cannot be zero address"" );
```

Status: Resolved

Disclaimer

The smart contracts provided to AuditOne have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions). The ethical nature of the project is not guaranteed by a technical audit of the smart contract. Any owner-controlled functions should be carried out by the responsible owner. Before participating in the project, all investors/users are recommended to conduct due research.

The focus of our assessment was limited to the code parts associated with the items defined in the scope. We draw attention to the fact that due to inherent limitations in any software development process and product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which cannot be free from any errors or failures. These preconditions can impact the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure, which adds further inherent risks as we rely on correctly executing the included third-party technology stack itself. Report readers should also consider that over the life cycle of any software product, changes to the product itself or the environment in which it is operated can have an impact leading to operational behaviors other than initially determined in the business specification.