RD
AUDITORS

OASIS SMART CONTRACT, CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: OASIS
Prepared on: 3rd October 2021
Platform: Binance
Language: Solidity

TABLE OF CONTENTS

Document	5
Introduction	6
Project Scope	7
Executive Summary	8
Code Quality	9
Documentation	10
Use of Dependencies	10
AS-IS Overview	11
Severity Definitions	15
Audit Findings	16
Conclusion	22
Our Methodology	23
Disclaimers	25

THIS DOCUMENT MAY CONTAIN CONFIDENTIAL INFORMATION ABOUT ITS SYSTEMS AND INTELLECTUAL PROPERTY OF THE CUSTOMER AS WELL AS INFORMATION ABOUT POTENTIAL VULNERABILITIES AND METHODS OF THEIR EXPLOITATION.

THE REPORT CONTAINING CONFIDENTIAL INFORMATION CAN BE USED INTERNALLY BY THE CUSTOMER OR IT CAN BE DISCLOSED PUBLICLY AFTER ALL VULNERABILITIES ARE FIXED - UPON THE DECISION OF THE CUSTOMER.

Document

Name	**Smart Contract Code Review and Security Analysis Report for Oasis**
Platform	BSC / Solidity
File	FixedSwap.sol
MD5 hash	B9FE3EBAD223F9FDA5B8009BD0F0CE35
SHA256 hash	F5A972615799813E55A84C8043AFAB5571D3B5054526A1765EC248F637241E10
File	MasterChef.sol
MD5 hash	6CC0AA6FCC2F9103C97075C38A54E3A4
SHA256 hash	011B8EBF6D0E843690EA858ABF026B54DC4F0BFA40DC21A847F9F6E5D87617B9
File	Oasis Token.sol
MD5 hash	7BE845C4D4AAFBC23DEE79C48CFA73CD
SHA256 hash	C3BAD1EC6FBF389287BF3782FB4F413D04B30A8EFAFDE2A95D7E544DDFF65412
File	RewardLocker.sol
MD5 hash	2356CBCCA1C30369757A28389F085F79
SHA256 hash	AAE290EC1EE1A177EABF9421079579DC51B0B052B4D88DE161A38BCC8BFF9910
Date	03/10/2021

Introduction

RD Auditors (Consultant) were contracted by Oasis (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report represents the findings of the security assessment of the customer`s smart contracts and its code review conducted between
28th September - 3rd October 2021.

This contract consists of four files.

Project Scope

The scope of the project is a smart contract.

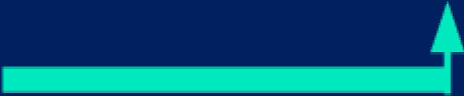
We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but is not limited to):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

According to the assessment, the customer's solidity smart contract is secure.

You are here



Automated checks are with smartDec, Mythril, Slither and remix IDE. All issues were performed by our team, which included the analysis of code functionality, manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the AS-IS section and all issues found are located in the audit overview section.

We found 0 critical, 0 high, 0 medium, 0 low and 0 very low level issues.

Code Quality

The libraries within this smart contract are part of a logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned to a specific address and its properties/methods can be reused many times by other contracts.

The Oasis team has not provided unit test scripts, which fortify functionality and security of the contract, which also helps us to determine the integrity of the code in an automated way.

Overall, the code is well commented. Commenting can provide rich documentation for functions, return variables and more. Use of Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

We were given the Oasis contract in the form of a file.

https://github.com/projectoasis-io/contracts/blob/main/contracts/launchpad/FixedSwap.sol

https://github.com/projectoasis-io/contracts/blob/main/contracts/staking/MasterChef.sol

https://github.com/projectoasis-io/contracts/blob/main/contracts/OasisToken.sol

https://github.com/projectoasis-io/contracts/blob/main/contracts/staking/RewardLocker.sol

The hash of that file is mentioned in the table. As mentioned above, It's well commented smart contract code, so anyone can quickly understand the programming flow as well as complex code logic.

Comments are very helpful in understanding the overall architecture of the protocol. It also provides a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure. Those were based on well-known industry standard open source projects and even core code blocks that are written well and systematically.

AS-IS Overview

Oasis

File And Function Level Report

File:FixedSwap.sol

Contract: Fixed Swap
Inherit: Pausable, whitelist
Import: ERC20, Ownable, SafeMath, Whitelist, Pausable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	isBuyer	read	Passed	All Passed	No Issue	Passed
2	totalRaiseCost	write	Passed	All Passed	No Issue	Passed
3	availableTokens	write	Passed	All Passed	No Issue	Passed
4	tokensleft	read	Passed	All Passed	No Issue	Passed
5	hasMinimumRaise	read	Passed	All Passed	No Issue	Passed
6	minimumRaiseNotAchieved	write	Passed	All Passed	No Issue	Passed
7	minimumRaiseAchieved	write	Passed	All Passed	No Issue	Passed
8	hasFinalized	write	Passed	All Passed	No Issue	Passed
9	hasStarted	write	Passed	All Passed	No Issue	Passed
10	isPreStart	write	Passed	All Passed	No Issue	Passed
11	isOpen	write	Passed	All Passed	No Issue	Passed
12	hasMinimumAmount	write	Passed	All Passed	No Issue	Passed
13	cost	write	Passed	All Passed	No Issue	Passed
14	getPurchase	write	Passed	All Passed	No Issue	Passed
15	getPurchaseIdS	write	Passed	All Passed	No Issue	Passed
16	getBuyers	write	Passed	All Passed	No Issue	Passed

17	getMyPurchases	write	Passed	All Passed	No Issue	Passed
18	fund	write	Passed	All Passed	No Issue	Passed
19	setUnlockDates	write	Passed	All Passed	No Issue	Passed
20	getLocked	write	Passed	All Passed	No Issue	Passed
21	Swap	write	Passed	All Passed	No Issue	Passed
22	redeemTokens	write	Passed	All Passed	No Issue	Passed
23	redeemGivenMinimumGoalNot Achived	write	Passed	All Passed	No Issue	Passed
24	withdrawFunds	write	Passed	All Passed	No Issue	Passed
25	withdrawunsoldTokens	write	Passed	All Passed	No Issue	Passed
26	removeOtherERC20Tokens	write	Passed	All Passed	No Issue	Passed
27	safePull	write	Passed	All Passed	No Issue	Passed

File: MasterChef.sol

Contract: MasterChef
Inherit: Ownable, ReentrancyGuard
Import: Ownable, SafeMath, SafeERC20, ReentrancyGuard
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	PoolLength	read	Passed	All Passed	No Issue	Passed
2	add	write	Passed	All Passed	No Issue	Passed
3	set	write	Passed	All Passed	No Issue	Passed
4	getMultiplier	read	Passed	All Passed	No Issue	Passed
5	PendingOasis	read	Passed	All Passed	No Issue	Passed
6	massUpdatePools	write	Passed	All Passed	No Issue	Passed
7	UpdatePool	write	Passed	All Passed	No Issue	Passed
8	deposit	write	Passed	All Passed	No Issue	Passed
9	withdraw	write	Passed	All Passed	No Issue	Passed
10	emergencyWithdraw	write	Passed	All Passed	No Issue	Passed
11	harvest	write	Passed	All Passed	No Issue	Passed
12	harvestMultiple	write	Passed	All Passed	No Issue	Passed
13	harvestAll	write	Passed	All Passed	No Issue	Passed
14	SetDevAddress	write	Passed	All Passed	No Issue	Passed
15	updateEmissionRate	write	Passed	All Passed	No Issue	Passed

16	setOasisTransferOwner	write	Passed	All Passed	No Issue	Passed
17	acceptOasisOwnership	write	Passed	All Passed	No Issue	Passed
18	SetPendingOasisOwnership	write	Passed	All Passed	No Issue	Passed

File:Oasis Token.sol

Contract: Oasis Token
Inherit: Ownable, ERC20Burnable
Import: Ownable, ERC20Burnable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

File:Reward Locker.sol

Contract: RewardLocker
Import: Ownable, safeCast, SafeMath, SafeERC20, ReentrancyGuard, EnumerableSet, IRewardLocker
Inherit: IRewardLocker, Ownable, ReentrancyGuard
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	addRewardsContract	write	Passed	All Passed	No Issue	Passed
2	removeRewardsContract	write	Passed	All Passed	No Issue	Passed
3	setVestingDuration	write	Passed	All Passed	No Issue	Passed
4	lock	write	Passed	All Passed	No Issue	Passed
5	VestCompletedSchedulesMultipleTokens	write	Passed	All Passed	No Issue	Passed
6	VestScheduleForMultipleTokensAtIndices	write	Passed	All Passed	No Issue	Passed
7	lockWithStartBlock	write	Passed	All Passed	No Issue	Passed

8	VestCompletedSchedules	write	Passed	All Passed	No Issue	Passed
9	VestScheduleAtIndices	write	Passed	All Passed	No Issue	Passed
10	SchedulesInRange	write	Passed	All Passed	No Issue	Passed
11	numVestingSchedules	read	Passed	All Passed	No Issue	Passed
12	getVestingScheduleAtIndex	read	Passed	All Passed	No Issue	Passed
13	getVestingSchedules	read	Passed	All Passed	No Issue	Passed
14	getRewardContractPerToken	read	Passed	All Passed	No Issue	Passed
15	UpdateMaxContractSize	write	Passed	All Passed	No Issue	Passed

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to lost tokens etc.
High	High level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial functions.
Medium	Medium level vulnerabilities are important to fix; however, they cannot lead to lost tokens.
Low	Low level vulnerabilities are most related to outdated, unused etc. These code snippets cannot have a significant impact on execution.
Lowest **Code Style/** **Best Practice**	Lowest level vulnerabilities, code style violations and information statements cannot affect smart contract execution and can be ignored.

Audit Findings

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

No medium severity vulnerabilities were found.

Low

No low severity vulnerabilities were found.

Very Low

No very low severity vulnerabilities were found.

Discussion:

File: FixedSwap.sol

1. Please be aware that line 268 shows that the loop is limited.

```
248     function swap(uint256 _amount) payable external whenNotPaused isFunded isSaleOpen onlyWhitelisted {
249
250         /* Confirm Amount is positive */
251         require(_amount > 0, "Amount has to be positive");
252
253         /* Confirm Amount is less than tokens available */
254         require(_amount <= tokensLeft(), "Amount is less than tokens available");
255
256         /* Confirm the user has funds for the transfer, confirm the value is equal */
257         require(msg.value == cost(_amount), "User has to cover the cost of the swap in ETH, use the cost function to determine");
258
259         /* Confirm Amount is bigger than minimum Amount */
260         require(_amount >= individualMinimumAmount, "Amount is bigger than minimum amount");
261
262         /* Confirm Amount is smaller than maximum Amount */
263         require(_amount <= individualMaximumAmount, "Amount is smaller than maximum amount");
264
265         /* Verify all user purchases, loop thru them */
266         uint256[] memory _purchases = getMyPurchases(msg.sender);
267         uint256 purchaserTotalAmountPurchased = 0;
268         for (uint i = 0; i < _purchases.length; i++) {
269             Purchase memory _purchase = purchases[_purchases[i]];
270             purchaserTotalAmountPurchased = purchaserTotalAmountPurchased.add(_purchase.amount);
271         }
```

2. There are some important functions which have no events:

- redeemTokens
- redeemGivenMinimumGoalNotAchieved

```
293     function redeemTokens(uint256 purchase_id) external isNotAtomicSwap isSaleFinalized whenNotPaused {
294         /* Confirm it exists and was not finalized */
295         require((purchases[purchase_id].amount != 0) && !purchases[purchase_id].wasFinalized, "Purchase is either 0 or finalized");
296         require(isBuyer(purchase_id), "Address is not buyer");
297
298       uint256 unlockedAmount = purchases[purchase_id].amount.sub(getLocked(purchase_id));
299       uint256 claimed = purchases[purchase_id].amount.sub(purchases[purchase_id].remainingAmount);
300       uint256 claimable = unlockedAmount - claimed;
301
302        require(claimable > 0, "To claim must be more than 0");
303
304         purchases[purchase_id].remainingAmount = purchases[purchase_id].remainingAmount - claimable;
305         if ( purchases[purchase_id].remainingAmount == 0){
306             purchases[purchase_id].wasFinalized = true;
307         }
308         require(erc20.transfer(msg.sender, claimable), "ERC20 transfer failed");
309     }
```

```
312     function redeemGivenMinimumGoalNotAchieved(uint256 purchase_id) external isSaleFinalized isNotAtomicSwap {
313         require(hasMinimumRaise(), "Minimum raise has to exist");
314         require(minimumRaiseNotAchieved(), "Minimum raise has to be reached");
315         /* Confirm it exists and was not finalized */
316         require((purchases[purchase_id].amount != 0) && !purchases[purchase_id].wasFinalized, "Purchase is either 0 or finalized");
317         require(isBuyer(purchase_id), "Address is not buyer");
318         purchases[purchase_id].wasFinalized = true;
319         purchases[purchase_id].reverted = true;
320         msg.sender.transfer(purchases[purchase_id].ethAmount);
321     }
```

File: Masterchef.sol

1. There are some places in the smart contract where some important functions - call event logs have not been added.

Functions are:

- MassUpdatePool
- UpdatePool

```
165     // Update reward variables for all pools. Be careful of gas spending!
166     function massUpdatePools() public {
167         uint256 length = poolInfo.length;
168         for (uint256 pid = 0; pid < length; ++pid) {
169             updatePool(pid);
170         }
171     }
```

```
173     // Update reward variables of the given pool to be up-to-date.
174     function updatePool(uint256 _pid) public {
175         PoolInfo storage pool = poolInfo[_pid];
176         if (block.number <= pool.lastRewardBlock) {
177             return;
178         }
179         if (pool.totalDeposited == 0 || pool.allocPoint == 0) {
180             pool.lastRewardBlock = block.number;
181             return;
182         }
183         uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
184         uint256 oasisReward = multiplier.mul(oasisPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
185       //  oasis.mint(devAddress, oasisReward.div(50)); // 2%
186       //  oasis.mint(address(this), oasisReward);
187         pool.accOasisPerShare = pool.accOasisPerShare.add(oasisReward.mul(1e18).div(pool.totalDeposited));
188         pool.lastRewardBlock = block.number;
189     }
190
```

2. 'massUpdatePools' and 'harvestMultiple' may fail after a certain length, this is fine if the Oasis team is aware of this function.

```
165    // Update reward variables for all pools. Be careful of gas spending!
166    function massUpdatePools() public {
167        uint256 length = poolInfo.length;
168        for (uint256 pid = 0; pid < length; ++pid) {
169            updatePool(pid);
170        }
171    }
```

```
263    function harvestMultiple(uint256[] calldata _pids) external nonReentrant {
264        uint256 pending = 0;
265        for (uint256 i = 0; i < _pids.length; i++) {
266            updatePool(_pids[i]);
267            PoolInfo storage pool = poolInfo[_pids[i]];
268            UserInfo storage user = userInfo[_pids[i]][msg.sender];
269            if (user.amount == 0) {
270                user.lastOasisPerShare = pool.accOasisPerShare;
271            }
272            pending = pending.add(user.amount.mul(pool.accOasisPerShare.sub(user.lastOasisPerShare)).div(1e18).add(user.unclaimed));
273            user.unclaimed = 0;
274            user.lastOasisPerShare = pool.accOasisPerShare;
275        }
276        if (pending > 0) {
277            _lockReward(msg.sender, pending);
278        }
279        emit HarvestMultiple(msg.sender, _pids, pending);
280    }
```

File: RewardLocker.sol

1. We assume that the scheduled and token length should be limited to avoid an issue with the call function.

```
92   /**
93    * @dev vest all completed schedules for multiple tokens
94    */
95   function vestCompletedSchedulesForMultipleTokens(IERC20[] calldata tokens)
96     external
97     override
98     returns (uint256[] memory vestedAmounts)
99   {
100    vestedAmounts = new uint256[](tokens.length);
101    for (uint256 i = 0; i < tokens.length; i++) {
102      vestedAmounts[i] = vestCompletedSchedules(tokens[i]);
103    }
104  }
105
```

```
110  function vestScheduleForMultipleTokensAtIndices(
111    IERC20[] calldata tokens,
112    uint256[][] calldata indices
113  ) external override returns (uint256[] memory vestedAmounts) {
114    require(tokens.length == indices.length, 'tokens.length != indices.length');
115    vestedAmounts = new uint256[](tokens.length);
116    for (uint256 i = 0; i < tokens.length; i++) {
117      vestedAmounts[i] = vestScheduleAtIndices(tokens[i], indices[i]);
118    }
119  }
```

```
182  function vestCompletedSchedules(IERC20 token) public override returns (uint256) {
183    VestingSchedules storage schedules = accountVestingSchedules[msg.sender][token];
184    uint256 schedulesLength = schedules.length;
185
186    uint256 totalVesting = 0;
187    for (uint256 i = 0; i < schedulesLength; i++) {
188      VestingSchedule memory schedule = schedules.data[i];
189      if (_blockNumber() < schedule.endBlock) {
190        continue;
191      }
192      uint256 vestQuantity = uint256(schedule.quantity).sub(schedule.vestedQuantity);
193      if (vestQuantity == 0) {
194        continue;
195      }
196      schedules.data[i].vestedQuantity = schedule.quantity;
197      totalVesting = totalVesting.add(vestQuantity);
198
199      emit Vested(token, msg.sender, vestQuantity, i);
200    }
201    _completeVesting(token, totalVesting);
202
203    return totalVesting;
204  }
```

File: FixedSwap.sol

The FixedSwap.sol file has a function called ‘safePull’, which if executed could drain all funds. We however understand that it's considered as a safety option to pause the contract if an issue is encountered. But these points are something to take into consideration.

```
353      /* Safe Pull function */
354      function safePull() payable external onlyOwner whenPaused {
355          msg.sender.transfer(address(this).balance);
356          erc20.transfer(msg.sender, erc20.balanceOf(address(this)));
357      }
```

Conclusion

We were given a contract file and have used all possible tests based on the given object. The contract is written systematically, so it is ready to go for production.

Since possible test cases can be unlimited and developer level documentation (code flow diagram with function level description) not provided, for such an extensive smart contract protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

The security state of the reviewed contract is “secure”

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyse the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

The smart contracts given for audit have been analysed in accordance with the best industry practices at the date of this report, in relation to: 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).

Because the total number of test cases are unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

RD
AUDITORS

Email: info@rdauditors.com
Website: www.rdauditors.com