



**RD
AUDITORS**

Liquidity Mining, Smart Contract, Code Review and Security Analysis Report

Customer: OVR Platform
Prepared on: 4th May, 2023
Platform: Ethereum
Language: Solidity

rdauditors.com

Table of Contents

Disclaimer	2
Documentation	3
Introduction	4
Project Scope	5
Executive Summary	6
Code Quality	6
Documentation	8
Use of Dependencies	9
AS-IS Overview	10
Code Flow Diagram - OVR.sol	12
Code Flow Diagram - LiquidityMining.sol	13
Code Flow Diagram - Slither Results Log	14
Severity Definitions	23
Audit Findings	24
Conclusion	25
Note For Contract Users	25
Our Methodology	27
Disclaimers	29

Disclaimer

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.

Documentation

Name	Smart Contract Code Review and Security Analysis Report of OVR Platform
Platform	Ethereum / Solidity
File	OVRToken.sol
MD5 hash	01cc8480f3d71c843a66b2aaaf4fae30
SHA256 hash	c0e7a37ecc10d48b8f297cd303c2400ae4ef4480ad0080b00774d4e7e4f3b013
File 2	LiquidityMining.sol
MD5 hash	9c052c653e9e5b8c1cf7c13ec3e0c17a
SHA256 hash	07fdd21298089b78cf0bcae5481d063b2c7851942d7245d3e61cb9af2de80f58
File 3	Store.sol
MD5 hash	701e59d7ac47b68822310e537716e089
SHA256 hash	07b643f42f4d4ebc26fe7de1736a4c8002072b20f2132195de3d0193bd54d971
Date	04/05/2023

Introduction

RD Auditors (Consultant) were contracted by OVR Platform (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report represents the findings of the security assessment of the customer`s smart contract and its code review conducted between 1st - 4th May, 2023.

This contract consists of three files.

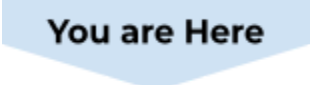
Project Scope

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 now **Well-Secured**.








You are Here

 **Insecure**  **Poorly Secured**  **Secure**  **Well-Secured**

Automated checks are with smartDec, Mythril, Slither and remix IDE. All issues were performed by our team, which included the analysis of code functionality, the 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 the following;

Total Issues	1
 Critical	0
 High	0
 Medium	0
 Low	0
 Very Low	1

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 OVR Platform team has not provided scenario and unit test scripts, which helped to determine the integrity of the code in an automated way.

Documentation

We were given a OVR Platform smart contract code in the form of a Github link:

<https://github.com/OVR-Platform/liquidity-mining-smart-contracts>

The hash of that code is mentioned above in the table. As mentioned above, It's recommended to write comments in the 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

OVR.sol

File And Function Level Report

File: OVR.sol
Contract: OVRToken
Inherit: Ownable, ERC20
Observation: Passed
Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	mint	write	Passed	All Passed	No Issue	Passed
2	burn	write	Passed	All Passed	No Issue	Passed

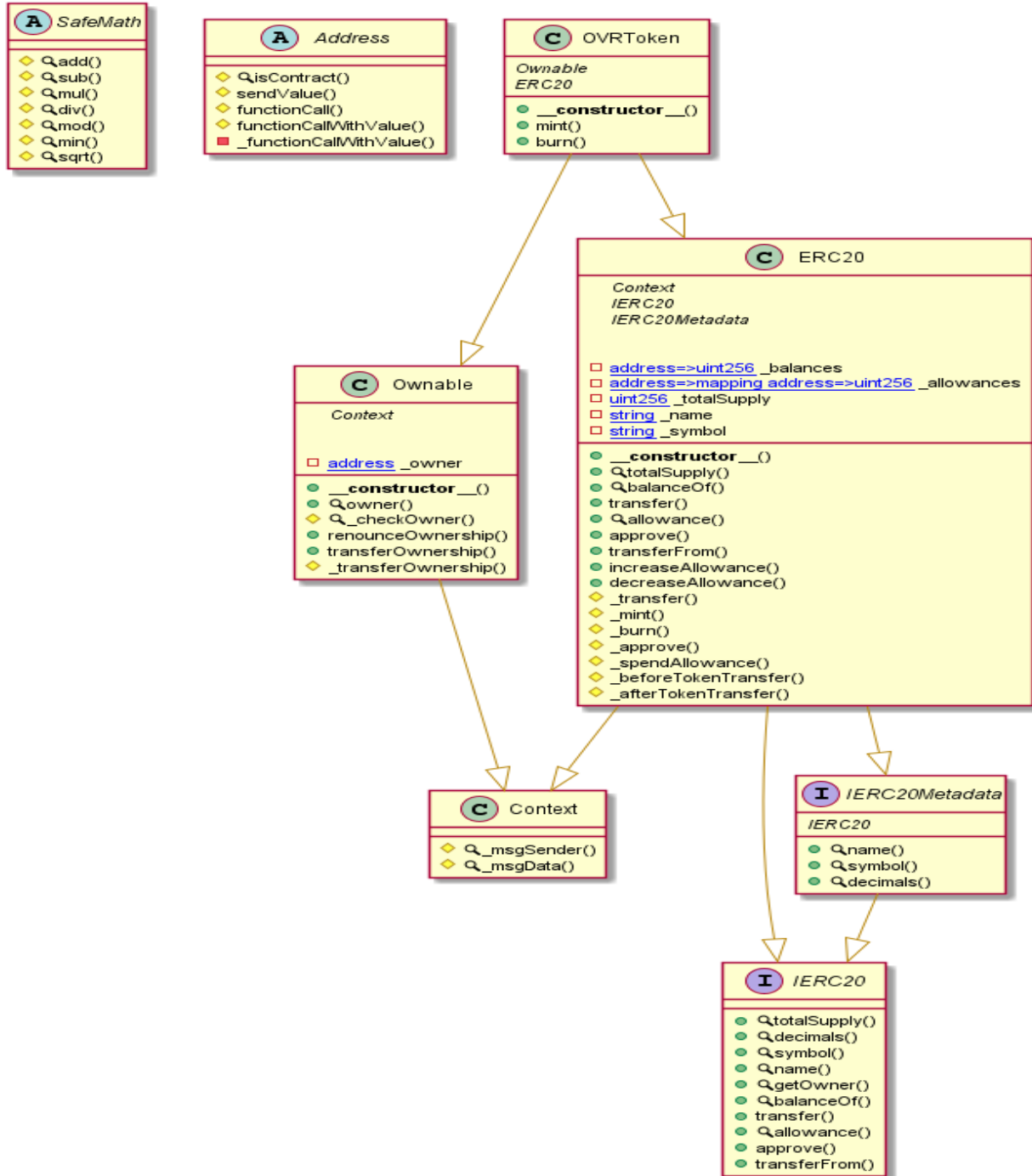
LiquidityMining.sol

File And Function Level Report

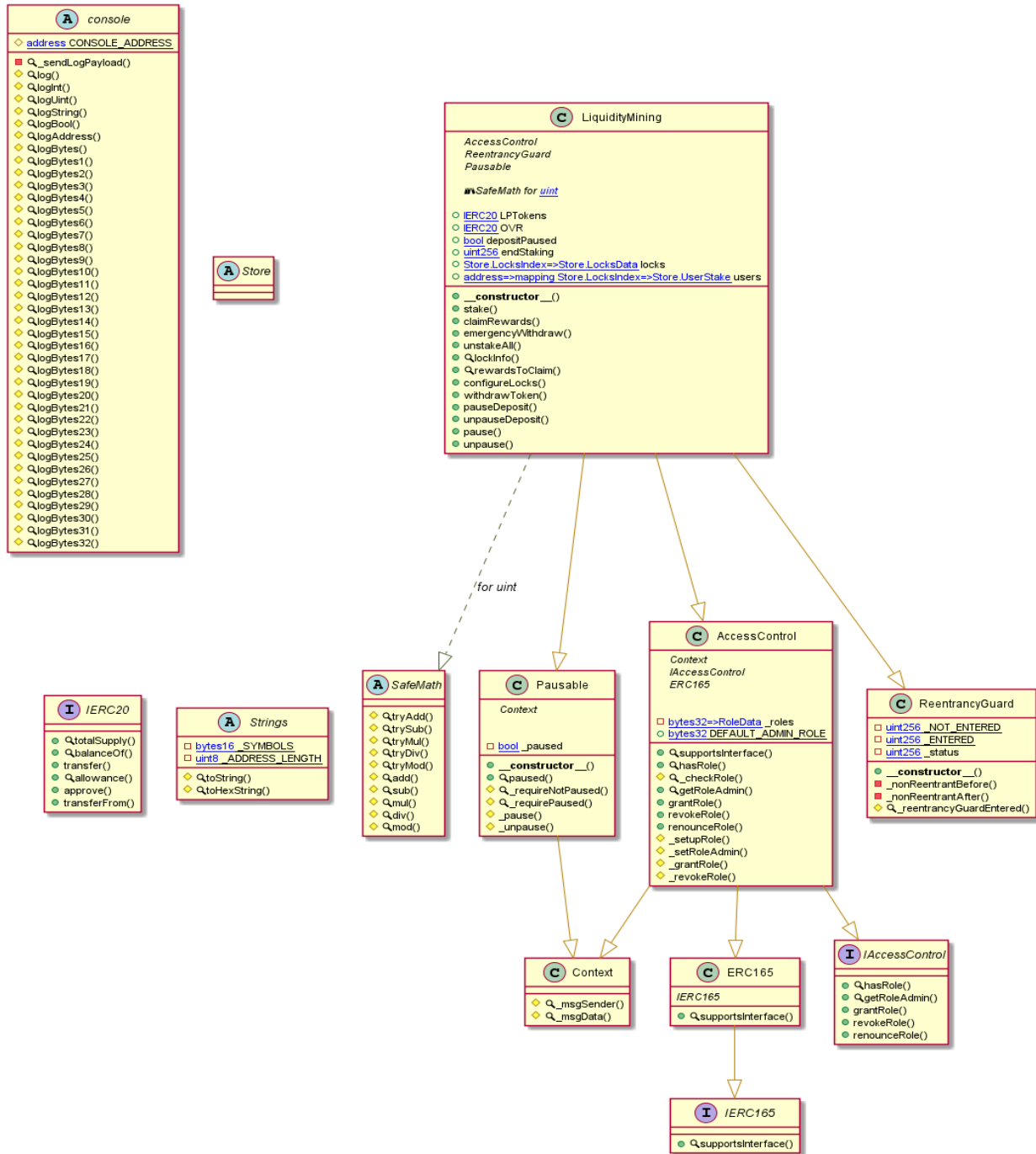
File: LiquidityMining.sol
Contract: LiquidityMining
Inherit: AccessControl, ReentrancyGuard, Pausable
Observation: Passed
Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	stake	external	Passed	All Passed	No Issue	Passed
2	claimRewards	public	Passed	All Passed	No Issue	Passed
3	emergencyWithdraw	external	Passed	All Passed	No Issue	Passed
4	unstakeAll	external	Passed	All Passed	No Issue	Passed
5	lockInfo	public	Passed	All Passed	No Issue	Passed
6	rewardsToClaim	public	Passed	All Passed	No Issue	Passed
7	configureLocks	external	Passed	All Passed	No Issue	Passed
8	withdrawToken	external	Passed	All Passed	No Issue	Passed
9	pauseDeposit	external	Passed	All Passed	No Issue	Passed
10	unpauseDeposit	external	Passed	All Passed	No Issue	Passed
11	pause	external	Passed	All Passed	No Issue	Passed
12	unpause	external	Passed	All Passed	No Issue	Passed

Code Flow Diagram - OVR.sol



Code Flow Diagram - LiquidityMining.sol



Code Flow Diagram - Slither Results Log

OVR.sol

```
Address.isContract(address) (OVR.sol#85-92) uses assembly
- INLINE ASM (OVR.sol#88-90)
Address._functionCallWithValue(address,bytes,uint256,string) (OVR.sol#131-153) uses assembly
- INLINE ASM (OVR.sol#145-148)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Address.functionCallWithValue(address,bytes,uint256,string) (OVR.sol#131-153) is never used and should be removed
Address.functionCall(address,bytes) (OVR.sol#101-103) is never used and should be removed
Address.functionCall(address,bytes,string) (OVR.sol#105-111) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (OVR.sol#113-119) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (OVR.sol#121-129) is never used and should be removed
Address.isContract(address) (OVR.sol#85-92) is never used and should be removed
Address.sendValue(address,uint256) (OVR.sol#94-99) is never used and should be removed
Context._msgData() (OVR.sol#190-193) is never used and should be removed
SafeMath.add(uint256,uint256) (OVR.sol#6-11) is never used and should be removed
SafeMath.div(uint256,uint256) (OVR.sol#39-41) is never used and should be removed
SafeMath.div(uint256,uint256,string) (OVR.sol#43-52) is never used and should be removed
SafeMath.min(uint256,uint256) (OVR.sol#67-69) is never used and should be removed
SafeMath.mod(uint256,uint256) (OVR.sol#54-56) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (OVR.sol#58-65) is never used and should be removed
SafeMath.mul(uint256,uint256) (OVR.sol#28-37) is never used and should be removed
SafeMath.sqrt(uint256) (OVR.sol#71-82) is never used and should be removed
SafeMath.sub(uint256,uint256) (OVR.sol#13-15) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (OVR.sol#17-26) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version0.8.13 (OVR.sol#2) allows old versions
solc-0.8.13 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (OVR.sol#94-99):
- (success) = recipient.call{value: amount}() (OVR.sol#97)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (OVR.sol#131-153):
- (success,returndata) = target.call{value: weiValue}(data) (OVR.sol#139)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Parameter OVRToken.mint(address,uint256)._to (OVR.sol#596) is not in mixedCase
Parameter OVRToken.mint(address,uint256)._amount (OVR.sol#596) is not in mixedCase
Parameter OVRToken.burn(address,uint256)._from (OVR.sol#605) is not in mixedCase
Parameter OVRToken.burn(address,uint256)._amount (OVR.sol#605) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (OVR.sol#191)" inContext (OVR.sol#185-194)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

OVRToken.constructor(string,string) (OVR.sol#584-589) uses literals with too many digits:
- mint(msg.sender,8168815500000000000000000000) (OVR.sol#588)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

OVRToken (OVR.sol#582-608) does not implement functions:
- IERC20Metadata.decimals() (OVR.sol#210)
- IERC20.getOwner() (OVR.sol#164)
- IERC20Metadata.name() (OVR.sol#200)
- IERC20Metadata.symbol() (OVR.sol#205)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions

OVR.sol analyzed (8 contracts with 84 detectors), 31 result(s) found
```

LiquidityMining.sol

```
LiquidityMining.claimRewards(Store.LocksIndex) (LiquidityMining.sol#2250-2270) uses timestamp for comparisons
  Dangerous comparisons:
  - rewards > 0 (LiquidityMining.sol#2259)
LiquidityMining.emergencyWithdraw(Store.LocksIndex) (LiquidityMining.sol#2276-2305) uses timestamp for comparisons
  Dangerous comparisons:
  - require(bool,string)(block.timestamp >= users[_msgSender()][_timeLockIndex].lastStake + locks[_timeLockIndex].timeLo
cked,LiquidityMining: Cannot unstake yet) (LiquidityMining.sol#2283-2288)
LiquidityMining.unstakeAll(Store.LocksIndex) (LiquidityMining.sol#2311-2341) uses timestamp for comparisons
  Dangerous comparisons:
  - require(bool,string)(block.timestamp >= users[_msgSender()][_timeLockIndex].lastStake + locks[_timeLockIndex].timeLo
cked,LiquidityMining: Cannot unstake yet) (LiquidityMining.sol#2314-2319)
  - rewards > 0 (LiquidityMining.sol#2325)
LiquidityMining.rewardsToClaim(address,Store.LocksIndex) (LiquidityMining.sol#2361-2427) uses timestamp for comparisons
  Dangerous comparisons:
  - block.timestamp < lockExpiration (LiquidityMining.sol#2386)
  - block.timestamp >= lockExpiration && lastClaim < lockExpiration (LiquidityMining.sol#2397)
  - block.timestamp > endStaking && endStakingExist (LiquidityMining.sol#2387-2389)
  - block.timestamp > endStaking && endStakingExist (LiquidityMining.sol#2410-2412)
  - block.timestamp > endStaking && endStakingExist (LiquidityMining.sol#2419-2421)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

console._sendLogPayload(bytes) (LiquidityMining.sol#7-14) uses assembly
  - INLINE ASM (LiquidityMining.sol#10-13)
Strings.toString(uint256) (LiquidityMining.sol#1867-1885) uses assembly
  - INLINE ASM (LiquidityMining.sol#1872-1874)
  - INLINE ASM (LiquidityMining.sol#1877-1879)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
Pragma version0.8.13 (LiquidityMining.sol#2) allows old versions
solc-0.8.13 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Contract console (LiquidityMining.sol#4-1532) is not in CapWords
Event Storestake(address,uint256,uint256,Store.LocksIndex) (LiquidityMining.sol#1564-1569) is not in CapWords
Event StoreclaimRewards(address,uint256,uint256,Store.LocksIndex) (LiquidityMining.sol#1570-1575) is not in CapWords
Event StoreemergencyWithdraw(address,uint256,uint256,Store.LocksIndex) (LiquidityMining.sol#1576-1581) is not in CapWords
Event Storewithdraw(address,uint256,uint256,Store.LocksIndex) (LiquidityMining.sol#1582-1587) is not in CapWords
Parameter LiquidityMining.stake(uint256,Store.LocksIndex)._amount (LiquidityMining.sol#2210) is not in mixedCase
Parameter LiquidityMining.stake(uint256,Store.LocksIndex)._timeLockIndex (LiquidityMining.sol#2211) is not in mixedCase
Parameter LiquidityMining.claimRewards(Store.LocksIndex)._timeLockIndex (LiquidityMining.sol#2251) is not in mixedCase
Parameter LiquidityMining.emergencyWithdraw(Store.LocksIndex)._timeLockIndex (LiquidityMining.sol#2277) is not in mixedCase
Parameter LiquidityMining.unstakeAll(Store.LocksIndex)._timeLockIndex (LiquidityMining.sol#2312) is not in mixedCase
Parameter LiquidityMining.lockInfo(Store.LocksIndex)._timeLockIndex (LiquidityMining.sol#2346) is not in mixedCase
Parameter LiquidityMining.rewardsToClaim(address,Store.LocksIndex)._user (LiquidityMining.sol#2362) is not in mixedCase
Parameter LiquidityMining.rewardsToClaim(address,Store.LocksIndex)._timeLockIndex (LiquidityMining.sol#2363) is not in mixedCase
Parameter LiquidityMining.configureLocks(uint128,Store.LocksIndex,uint128,uint256)._emissionPerSecond (LiquidityMining.sol#2440) is not in mixedCase
Parameter LiquidityMining.configureLocks(uint128,Store.LocksIndex,uint128,uint256)._timeLockIndex (LiquidityMining.sol#2441) is not in mixedCase
Parameter LiquidityMining.configureLocks(uint128,Store.LocksIndex,uint128,uint256)._timeLocked (LiquidityMining.sol#2442) is not in mixedCase
Parameter LiquidityMining.configureLocks(uint128,Store.LocksIndex,uint128,uint256)._maxOvrStakable (LiquidityMining.sol#2443) is not in mixedCase
Parameter LiquidityMining.withdrawToken(address,uint256,address)._token (LiquidityMining.sol#2463) is not in mixedCase
Parameter LiquidityMining.withdrawToken(address,uint256,address)._amount (LiquidityMining.sol#2464) is not in mixedCase
Parameter LiquidityMining.withdrawToken(address,uint256,address)._to (LiquidityMining.sol#2465) is not in mixedCase
Variable LiquidityMining.LPTokens (LiquidityMining.sol#2180) is not in mixedCase
Variable LiquidityMining.OVR (LiquidityMining.sol#2181) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
Variable LiquidityMining.rewardsToClaim(address,Store.LocksIndex)._rewards_scope_1 (LiquidityMining.sol#2414) is too similar to LiquidityMining.rewardsToClaim(address,Store.LocksIndex)._rewards_scope_3 (LiquidityMining.sol#2423)
Variable LiquidityMining.rewardsToClaim(address,Store.LocksIndex).timeDelta_scope_0 (LiquidityMining.sol#2410-2412) is too similar to LiquidityMining.rewardsToClaim(address,Store.LocksIndex).timeDelta_scope_2 (LiquidityMining.sol#2419-2421)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

LiquidityMining.LPTokens (LiquidityMining.sol#2180) should be immutable
LiquidityMining.OVR (LiquidityMining.sol#2181) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
LiquidityMining.sol analyzed (13 contracts with 84 detectors), 444 result(s) found
```

Solidity Static Analysis

Ovr.sol

Gas & Economy

Gas costs:

Gas requirement of function OVRToken.mint is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 20:4:

Gas costs:

Gas requirement of function OVRToken.burn is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 29:4:

Miscellaneous

Constant/View/Pure functions:

ERC20._afterTokenTransfer(address,address,uint256) :
Potentially should be constant/view/pure but is not. Note:
Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 384:4:

Similar variable names:

ERC20._burn(address,uint256) : Variables have very similar
names "account" and "amount". Note: Modifiers are currently
not considered by this static analysis.

Pos: 286:16:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any
circumstance (apart from a bug in your code). Use "require(x)"
if x can be false, due to e.g. invalid input or a failing external
component.

[more](#)

Pos: 237:8:

LiquidityMining.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in LiquidityMining.stake(uint256,enum Store.LocksIndex): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 45:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in LiquidityMining.claimRewards(enum Store.LocksIndex): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 86:4:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

[more](#)

Pos: 233:12:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

[more](#)

Pos: 248:18:

Gas & Economy

Gas costs:

Gas requirement of function LiquidityMining.unpauseDeposit is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 315:4:

Gas costs:

Gas requirement of function LiquidityMining.unpause is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 324:4:

Miscellaneous

Similar variable names:

LiquidityMining.rewardsToClaim(address,enum Store.LocksIndex) : Variables have very similar names "users" and "_user". Note: Modifiers are currently not considered by this static analysis.

Pos: 208:12:

Similar variable names:

LiquidityMining.rewardsToClaim(address,enum Store.LocksIndex) : Variables have very similar names "users" and "_user". Note: Modifiers are currently not considered by this static analysis.

Pos: 211:39:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

[more](#)

Pos: 223:12:

Data truncated:

Division of integer values yields an integer value again. That means e.g. $10 / 100 = 0$ instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Pos: 250:31:

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:

SafeMath Library:

SafeMath Library is used in this contract code, but the compiler version is greater than or equal to 0.8.0. Solidity automatically handles overflow/underflow.

Resolution: Remove the SafeMath library and use normal math operators, it will improve code size, and less gas consumption.

Conclusion

We were given a contract code in the form of a bscscan.com link and have used all possible tests based on the given object. So it is ready to go for production. 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 "**well-secured**".

Note For Contract Users

Technical auditing does not guarantee the project's ethical side.

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

RD Auditors Disclaimer

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.



Email: info@rdauditors.com

Website: www.rdauditors.com

