RD AUDITORS

Life Games, Smart Contract, Code Review and Security Analysis Report

Customer: Life Games
Prepared on: 1st May 2022
Platform: BSC
Language: Solidity

rdauditors.com

Table of Contents

Disclaimer 2
Document 3
Introduction 4
Project Scope 5
Executive Summary 6
Code Quality 6
Documentation 8
Use of Dependencies 8
AS-IS Overview 9
Code Flow Diagram - Life Games 13
Code Flow Diagram - Slither Results Log 14
Severity Definitions 19
Audit Findings 20
Discussion 21
Conclusion 22
Note For Contract Users 23
Our Methodology 25
Disclaimers 28

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.

Document

Name	Smart Contract Code Review and Security Analysis Report of Life Games
Platform	BSC / Solidity
File	LifeGames
MD5 hash	C76AB6EF8E0E08F5258202CA91518F02
SHA256 hash	59F8AB073713FDC8F1A1CD1897DAFAA2B0226CCBB35EBC4B42C384B3C8BCF14B
Date	1/5/2022

Introduction

RD Auditors (Consultant) were contracted by Life Games (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 21st April - 1st May 2022.

This contract consists of one file.

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.**

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	0
Critical	0
High	0
Medium	0
Low	0
Very Low	0

Code Quality

The library within this smart contract is 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 Life Games team has provided scenario and unit test scripts, which helped to determine the integrity of the code in an automated way.

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

Documentation

We were given the LifeGames code as a github link:

https://github.com/Life-Games-LFGM/LFGM-Token/blob/8d482241652e56bf808d813796f28c2f3a509322/contracts/LIFEGAMES.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

Life Games

File And Function Level Report

Contract: LIFEGAMES

Inherit: ERC20Upgradeable

Observation: Passed

Test Report: Passed

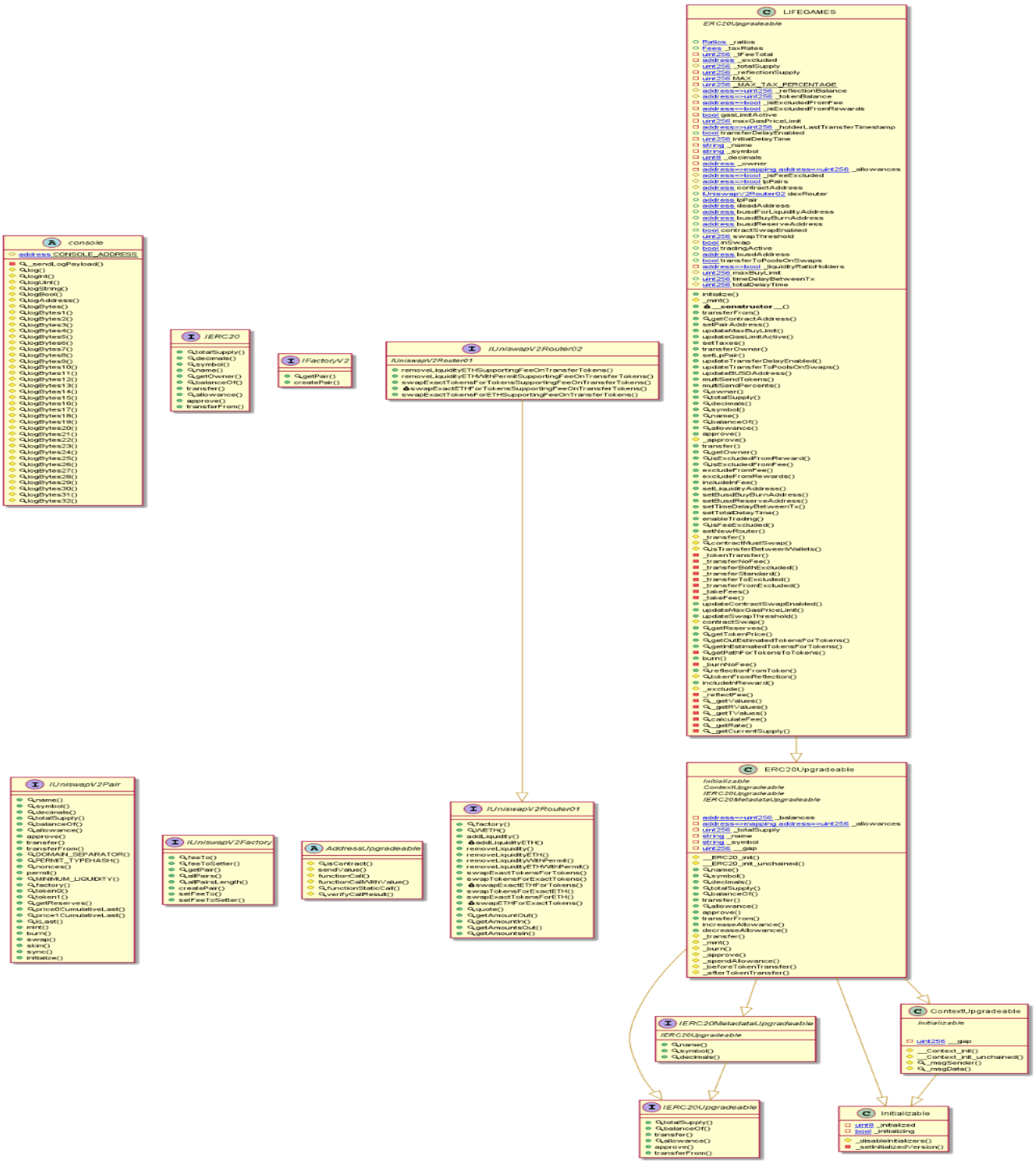
Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	Initialize	write	Passed	All Passed	No Issue	Passed
2	_mint	internal	Passed	All Passed	No Issue	Passed
3	transferFrom	write	Passed	All Passed	No Issue	Passed
4	getContractAddress	read	Passed	All Passed	No Issue	Passed
5	setPairAddress	onlyowner	Passed	All Passed	No Issue	Passed
6	UpdateMaxBuyLimit	onlyowner	Passed	All Passed	No Issue	Passed
7	UpdateGasLimitActive	onlyowner	Passed	All Passed	No Issue	Passed
8	setTaxes	onlyowner	Passed	All Passed	No Issue	Passed

9	transferOwner	onlyowner	Passed	All Passed	No Issue	Passed
10	setLpPair	onlyowner	Passed	All Passed	No Issue	Passed
11	UpdateTransferDelayEnabled	onlyowner	Passed	All Passed	No Issue	Passed
12	updateTransferToPoolsonSwaps	onlyowner	Passed	All Passed	No Issue	Passed
13	updateBusdAddress	onlyowner	Passed	All Passed	No Issue	Passed
14	multisendTokens	external	Passed	All Passed	No Issue	Passed
15	multisendPercents	external	Passed	All Passed	No Issue	Passed
16	owner	read	Passed	All Passed	No Issue	Passed
17	totalSupply	read	Passed	All Passed	No Issue	Passed
18	decimals	read	Passed	All Passed	No Issue	Passed
19	symbol	read	Passed	All Passed	No Issue	Passed
20	name	read	Passed	All Passed	No Issue	Passed
21	balanceOf	read	Passed	All Passed	No Issue	Passed
22	allowance	read	Passed	All Passed	No Issue	Passed
23	approve	write	Passed	All Passed	No Issue	Passed
24	_approve	internal	Passed	All Passed	No Issue	Passed
25	transfer	write	Passed	All Passed	No Issue	Passed
26	getOwner	read	Passed	All Passed	No Issue	Passed
27	isExcludedFromReward	read	Passed	All Passed	No Issue	Passed
28	isExcludedFromFee	read	Passed	All Passed	No Issue	Passed
29	excludeFromFee	onlyowner	Passed	All Passed	No Issue	Passed
30	excludeFromRewards	onlyowner	Passed	All Passed	No Issue	Passed

31	setLiquidityAddress	onlyowner	Passed	All Passed	No Issue	Passed
32	SetBusdBuyBurnAddress	onlyowner	Passed	All Passed	No Issue	Passed
33	setBusdReserveAddress	onlyowner	Passed	All Passed	No Issue	Passed
34	setTimeDelayBetweenTx	onlyowner	Passed	All Passed	No Issue	Passed
35	setTotalDelayTime	onlyowner	Passed	All Passed	No Issue	Passed
36	enableTrading	onlyowner	Passed	All Passed	No Issue	Passed
37	isFeeExcluded	read	Passed	All Passed	No Issue	Passed
38	setNewRouter	onlyOwner	Passed	All Passed	No Issue	Passed
39	_transfer	internal	Passed with the client's consent	Passed with the client's consent	Passed with client's consent	Passed with client's consent
40	contractMustSwap	internal	Passed	All Passed	No Issue	Passed
41	isTransferBetweenWallets	internal	Passed	All Passed	No Issue	Passed
42	_tokenTransfer	write	Passed	All Passed	No Issue	Passed
43	_transfer	write	Passed	All Passed	No Issue	Passed
44	contractMustSwap	write	Passed	All Passed	No Issue	Passed
45	isTransferBetweenWallets	write	Passed	All Passed	No Issue	Passed
46	_tokenTransfer	write	Passed	All Passed	No Issue	Passed
47	_transferNoFee	write	Passed	All Passed	No Issue	Passed
48	_transferBothExcluded	write	Passed	All Passed	No Issue	Passed
49	_transferStandard	write	Passed	All Passed	No Issue	Passed
50	_transferToExcluded	write	Passed	All Passed	No Issue	Passed

51	_transferFromExcluded	write	Passed	All Passed	No Issue	Passed
52	_takeFee	write	Passed	All Passed	No Issue	Passed
53	updateContractSwapEnabled	onlyOwner	Passed	All Passed	No Issue	Passed
54	updateMaxGasPriceLimit	onlyowner	Passed	All Passed	No Issue	Passed
55	updateSwapThreshold	onlyowner	Passed	All Passed	No Issue	Passed
56	contractSwap	internal	Passed	All Passed	No Issue	Passed
57	getReserves	write	Passed	All Passed	No Issue	Passed
58	getTokenPrice	read	Passed	All Passed	No Issue	Passed
59	getOutEstimatedTokensForTokens	read	Passed	All Passed	No Issue	Passed
60	getInEstimatedTokensToTokens	read	Passed	All Passed	No Issue	Passed
61	burn	write	Passed	All Passed	No Issue	Passed
62	_burnNoFee	write	Passed	All Passed	No Issue	Passed
63	reflectionFromToken	read	Passed	All Passed	No Issue	Passed
64	tokenFromReflection	internal	Passed	All Passed	No Issue	Passed
65	IncludeInReward	onlyowner	Passed	All Passed	No Issue	Passed
66	_exclude	internal	Passed	All Passed	No Issue	Passed
67	_reflectFee	write	Passed	All Passed	No Issue	Passed
68	_getValues	read	Passed	All Passed	No Issue	Passed
69	_getRValues	read	Passed	All Passed	No Issue	Passed
70	_getTValues	read	Passed	All Passed	No Issue	Passed
71	calculateFee	read	Passed	All Passed	No Issue	Passed
72	_getRate	read	Passed	All Passed	No Issue	Passed
73	_getCurrentSupply	read	Passed	All Passed	No Issue	Passed

Code Flow Diagram - Life Games



Code Flow Diagram - Slither Results Log

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 2081:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in LIFEGAMES.contractSwap(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 3535:4:

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: 3594:16:

Gas & Economy

Gas costs:

Gas requirement of function ERC20Upgradeable.name 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: 2387:4:

Gas costs:

Gas requirement of function LIFEGAMES.reflectionFromToken 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: 3668:4:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more
Pos: 3806:8:

ERC

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type
more
Pos: 1543:4:

Miscellaneous

Constant/View/Pure functions:

console._sendLogPayload(bytes) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 7:1:

Constant/View/Pure functions:

LIFEGAMES._getTValues(uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 3763:4:

Similar variable names:

LIFEGAMES._getValues(uint256) : Variables have very similar names "rAmount" and "tAmount". Note: Modifiers are currently not considered by this static analysis.
Pos: 3735:9:

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: 3517:8:

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: 3811:22:

Solidity Static Analysis

```
INFO:Detectors:
LIFEGAMES.updateMaxBuyLimit(uint256) (LIFEGAMES.sol#2987-2989) should emit an event for:
        - maxBuyLimit = newVal (LIFEGAMES.sol#2988)
LIFEGAMES.setTimeDelayBetweenTx(uint256) (LIFEGAMES.sol#3220-3222) should emit an event for:
        - timeDelayBetweenTx = time (LIFEGAMES.sol#3221)
LIFEGAMES.setTotalDelayTime(uint256) (LIFEGAMES.sol#3228-3230) should emit an event for:
        - totalDelayTime = time (LIFEGAMES.sol#3229)
LIFEGAMES.updateMaxGasPriceLimit(uint256) (LIFEGAMES.sol#3516-3519) should emit an event for:
        - maxGasPriceLimit = newValue (LIFEGAMES.sol#3518)
LIFEGAMES.updateSwapThreshold(uint256) (LIFEGAMES.sol#3525-3527) should emit an event for:
        - swapThreshold = newThreshold (LIFEGAMES.sol#3526)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
LIFEGAMES.setPairAddress(address).add1 (LIFEGAMES.sol#2977) lacks a zero-check on :
                - lpPair = add1 (LIFEGAMES.sol#2978)
LIFEGAMES.transferOwner(address).newOwner (LIFEGAMES.sol#3022) lacks a zero-check on :
                - _owner = newOwner (LIFEGAMES.sol#3025)
LIFEGAMES.updateBUSDAddress(address).newAddress (LIFEGAMES.sol#3059) lacks a zero-check on :
                - busdAddress = address(newAddress) (LIFEGAMES.sol#3060)
LIFEGAMES.setLiquidityAddress(address)._busdForLiquidityAddress (LIFEGAMES.sol#3196) lacks a zero-check on :
                - busdForLiquidityAddress = address(_busdForLiquidityAddress) (LIFEGAMES.sol#3197)
LIFEGAMES.setBusdBuyBurnAddress(address)._busdBuyBurnAddress (LIFEGAMES.sol#3204) lacks a zero-check on :
                - busdBuyBurnAddress = address(_busdBuyBurnAddress) (LIFEGAMES.sol#3205)
LIFEGAMES.setBusdReserveAddress(address)._busdReserveAddress (LIFEGAMES.sol#3212) lacks a zero-check on :
                - busdReserveAddress = address(_busdReserveAddress) (LIFEGAMES.sol#3213)
LIFEGAMES.setNewRouter(address,address).busd (LIFEGAMES.sol#3260) lacks a zero-check on :
                - lpPair = IFactoryV2(_newRouter.factory()).createPair(address(busd),address(this)) (LIFEGAMES.sol#3266)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in LIFEGAMES._transfer(address,address,uint256) (LIFEGAMES.sol#3288-3353):
        External calls:
        - contractSwap() (LIFEGAMES.sol#3335)
                - IERC20(busdAddress).approve(address(dexRouter),type()(uint256).max) (LIFEGAMES.sol#3546)
                - IERC20(busdAddress).approve(address(dexRouter),numTokensToSwap) (LIFEGAMES.sol#3557)
```

```
busdBuyBurnAddress,block.timestamp + 600) (LIFEGAMES.sol#3574-3580)
                - IERC20(busdAddress).approve(address(dexRouter),numTokensToSwap) (LIFEGAMES.sol#3587)
                - dexRouter.swapExactTokensForTokensSupportingFeeOnTransferTokens(busdReserveAmount,0,tokensBusdPath_scope_1,bu
ReserveAddress,block.timestamp + 600) (LIFEGAMES.sol#3589-3595)
        State variables written after the call(s):
        - _tokenTransfer(from,to,amount) (LIFEGAMES.sol#3352)
                - _tFeeTotal += tFee (LIFEGAMES.sol#3723)
Reentrancy in LIFEGAMES.setNewRouter(address,address) (LIFEGAMES.sol#3260-3276):
        External calls:
        - lpPair = IFactoryV2(_newRouter.factory()).createPair(address(busd),address(this)) (LIFEGAMES.sol#3266)
        State variables written after the call(s):
        - _approve(address(this),address(dexRouter),type()(uint256).max) (LIFEGAMES.sol#3275)
                - _allowances[tokenOwner][spender] = amount (LIFEGAMES.sol#3134)
        - dexRouter = _newRouter (LIFEGAMES.sol#3274)
        - lpPairs[lpPair] = true (LIFEGAMES.sol#3273)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in LIFEGAMES._transfer(address,address,uint256) (LIFEGAMES.sol#3288-3353):
        External calls:
        - contractSwap() (LIFEGAMES.sol#3335)
                - IERC20(busdAddress).approve(address(dexRouter),type()(uint256).max) (LIFEGAMES.sol#3546)
                - IERC20(busdAddress).approve(address(dexRouter),numTokensToSwap) (LIFEGAMES.sol#3557)
                - dexRouter.swapExactTokensForTokensSupportingFeeOnTransferTokens(tokensToliquidityAmount,0,tokensBusdPath,busd
rLiquidityAddress,block.timestamp + 600) (LIFEGAMES.sol#3559-3565)
                - IERC20(busdAddress).approve(address(dexRouter),numTokensToSwap) (LIFEGAMES.sol#3572)
                - dexRouter.swapExactTokensForTokensSupportingFeeOnTransferTokens(tokensToBuyBurnAmount,0,tokensBusdPath_scope_
```

```
busdBuyBurnAddress,block.timestamp + 600) (LIFEGAMES.sol#3574-3580)
                - IERC20(busdAddress).approve(address(dexRouter),numTokensToSwap) (LIFEGAMES.sol#3587)
                - dexRouter.swapExactTokensForTokensSupportingFeeOnTransferTokens(busdReserveAmount,0,tokensBusdPath_scope_1,bu
ReserveAddress,block.timestamp + 600) (LIFEGAMES.sol#3589-3595)
        Event emitted after the call(s):
        - Approval(tokenOwner,spender,amount) (LIFEGAMES.sol#3135)
                - _approve(sender,msg.sender,_allowances[sender][msg.sender] - amount) (LIFEGAMES.sol#2965)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
LIFEGAMES._transfer(address,address,uint256) (LIFEGAMES.sol#3288-3353) uses timestamp for comparisons
        Dangerous comparisons:
        - transferDelayEnabled && block.timestamp < (initialDelayTime + totalDelayTime) (LIFEGAMES.sol#3301)
        - require(bool,string)(_holderLastTransferTimestamp[msg.sender] <= block.timestamp,_transfer:: Transfer Delay enabled.)
LIFEGAMES.sol#3317)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
console._sendLogPayload(bytes) (LIFEGAMES.sol#7-14) uses assembly
        - INLINE ASM (LIFEGAMES.sol#10-13)
AddressUpgradeable.verifyCallResult(bool,bytes,string) (LIFEGAMES.sol#2127-2147) uses assembly
        - INLINE ASM (LIFEGAMES.sol#2139-2142)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
AddressUpgradeable.functionCall(address,bytes) (LIFEGAMES.sol#2038-2040) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes,string) (LIFEGAMES.sol#2048-2054) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (LIFEGAMES.sol#2067-2073) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (LIFEGAMES.sol#2081-2092) is never used and should be re
ved
AddressUpgradeable.functionStaticCall(address,bytes) (LIFEGAMES.sol#2100-2102) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (LIFEGAMES.sol#2110-2119) is never used and should be removed
AddressUpgradeable.sendValue(address,uint256) (LIFEGAMES.sol#2013-2018) is never used and should be removed
AddressUpgradeable.verifyCallResult(bool,bytes,string) (LIFEGAMES.sol#2127-2147) is never used and should be removed
ContextUpgradeable.__Context_init() (LIFEGAMES.sol#2335-2336) is never used and should be removed
ContextUpgradeable.__Context_init_unchained() (LIFEGAMES.sol#2338-2339) is never used and should be removed
```

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:

1. Line no 599 is not reachable due to conditions defined in line 592.

```
583     function _transfer(
584         address from,
585         address to,
586         uint256 amount
587     ) internal override virtual {
588         require(from != address(0), "ERC20: transfer from the zero address");
589         require(to != address(0), "ERC20: transfer to the zero address");
590         require(amount != 0, "Transfer amount cannot be zero");
591 
592         if (!tradingActive && from != _owner && to != address(dexRouter) && to != address(lpPair)) {
593             revert("Trading not yet enabled!");
594         }
595 
596         if (transferDelayEnabled && block.timestamp < (initialDelayTime + totalDelayTime)) {
597 
598             // at launch if the transfer delay is enabled, ensure the block timestamps for purchasers is set -- during launch.
599             if (from != _owner && to != address(dexRouter) && to != address(lpPair)) {
600 
601                 // in the first one hour, a maximum of XX BUSD purchase is adjustable (10000 BUSD is the default value)
602                 if (maxBuyLimit > 0) {
603                     require(amount <= maxBuyLimit, "Max Buy Limit.");
604                 }
605 
606                 // only use to prevent sniper buys in the first blocks.
607                 if (gasLimitActive) {
608                     require(tx.gasprice <= maxGasPriceLimit, "Gas price exceeds limit.");
609                 }
610 
611                 // delay between tx
612                 require(_holderLastTransferTimestamp[msg.sender] <= block.timestamp, "_transfer:: Transfer Delay enabled.");
613                 _holderLastTransferTimestamp[msg.sender] = block.timestamp + timeDelayBetweenTx;
614             }
```

Update (01/05/2022): passed with the client's consent.

Medium:

No medium severity vulnerabilities were found.

Low:

No low severity vulnerabilities were found.

Very Low:

No very low severity vulnerabilities were found.

Discussion

1. Line no 4, the contract is inherited by Openzepplin ERC20Upgradeable.sol, the contract functionality can be re-written/diverted by the authorised accounts, which is against the core concept of the immutable ledger. This upgradability functionality could result in a rug-pull or similar hacks, hence not recommended. We do not guarantee any future outcomes if the contract is upgraded.

2. Double check all hard coded values before deploying to the mainnet.

3. Ownership transfer without acceptOwnership, this will result in no roll back if ownership is transferred to the wrong address.

4. multiSendTokens and multiSendPercents may fail sometimes due to the presence of a malicious address (contract). However, the list of addresses is an input parameter of the function, so the caller can check and remove that malicious address.

5. _transfer function defined in line 583 is limited by the specific conditions/address.

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.

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 now secure.

Note For Contract Users

There are several owner only functions. Those can be called by the owner's wallet only. So, if the owner's wallet is compromised, it carries the risk of the contract becoming vulnerable.

The contract is inherited by Openzepplin ERC20Upgradeable.sol, the contract functionality can be re-written/diverted by the authorised accounts, which is against the core concept of the immutable ledger. This upgradability functionality could result in a rug-pull or similar hacks, hence not recommended. We do not guarantee any future outcomes if the contract is upgraded.

setNewRouter: The owner can set the router address.

```
555     function setNewRouter(address newRouter, address busd) public onlyOwner() {
556         IUniswapV2Router02 _newRouter = IUniswapV2Router02(newRouter);
557         address getPair = IFactoryV2(_newRouter.factory()).getPair(address(busd), address(this));
558 
559         // if not exists, create pair, otherwise get pair address from factory contract
560         if (getPair == address(0)) {
561             lpPair = IFactoryV2(_newRouter.factory()).createPair(address(busd), address(this));
562         }
563         else {
564             lpPair = getPair;
565         }
566 
567         // set lp address on automatic market maker list
568         lpPairs[lpPair] = true;
569         dexRouter = _newRouter;
570         _approve(address(this), address(dexRouter), type(uint256).max);
571     }
```

updateMaxGasPriceLimit: The owner can update the max gas price limit.

```
811     function updateMaxGasPriceLimit(uint256 newValue) public onlyOwner {
812         require(newValue >= 10000000000, "max gas price cant be lower than 10 gWei");
813         maxGasPriceLimit = newValue;
814     }
815
```

updateSwapThreshold: The owner can set a swap threshold limit amount.

```
820     function updateSwapThreshold(uint256 newThreshold) public onlyOwner {
821         swapThreshold = newThreshold;
822     }
```

includeInReward: The owner can include an address in reward.

```
985     function includeInReward(address account) external onlyOwner() {
986         require(_isExcludedFromRewards[account], "Account is not excluded");
987         for (uint256 i = 0; i < _excluded.length; i++) {
988             if (_excluded[i] == account) {
989                 _excluded[i] = _excluded[_excluded.length - 1];
990                 _tokenBalance[account] = 0;
991                 _isExcludedFromRewards[account] = false;
992                 _excluded.pop();
993                 break;
994             }
995         }
996     }
```

updateMaxBuyLimit: Update the max amount of tokens that can be bought in one transaction

```
282     function updateMaxBuyLimit(uint256 newVal) public onlyOwner() {
283         maxBuyLimit = newVal;
284     }
```

The owner has full control over the smart contract. Thus, technical auditing does not guarantee the project's ethical side.

Please do your due diligence before investing. Our audit report is never an investment advice.

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