Chain Integrity

The Web 3.0 Ethical Hacking Company

Revolve Games

Smart Contract Audit Deliverable

Date: Sep 29, 2021

Version: 2

Table of Contents

1.1 Summary

Project

Name	Revolve Games
Description	LP Staking, Bridge, Token Vesting, RNG
Platform	Binance Smart Chain (BSC)
Codebase	*
Commit	*

Engagement

Delivered	Sep 29th, 2021 (Updated Codebase)
Methods	Dynamic/Static Analysis, Manual Review
Consultants	2
Timeline	3 Days (Review Update)

Observations

Total	18	**Status**
Critical	2	Fixed (2/2)
High	6	Fixed (2/6), Improved (3/6), Unchanged (1/6)
Medium	1	Unchanged (1/1)
Low	9	Fixed (8/9), Unchanged (1/9)
Undetermined	0	

Executive

This document has been prepared for Revolve Games (Client) to discover and analyze the codebase provided by the team for security vulnerabilities, code correctness, and risk of investment. The codebase has been comprehensively examined using structural analysis, behavioral analysis, and manual review techniques.

Throughout the audit, caution was taken to ensure that the smart contract(s):

- Implements robust functions which are safe from well-known attack vectors.
- The logic and behavior adheres to the associated documentation and code comments.
- Transfer flows are designed in a sustainable way, safeguarded from i.e. infinite loops.
- Cross-comparison with similar smart contracts based on functionality.
- Does not hide potential back doors and implements sanity checks where suitable.

1.2 Scope

File	Fingerprint (SHA-256 Checksum)
ethBridge.sol	1eb84e9860a0ddc28d329c31500c79172c065c94196f7aeba69c97320cc046d4
random.sol	d25a981dbed17855a79e4d00c8a6115a6bf2df7bda45352f6eead3cfa363b8c7
LPStaking.sol	a96ad9ba3d6e10db512c38a4bc8caaee697c7464e87a2cbc29dc37c33a91398f
TokenTimelock.sol	d4784ded65e400168a963c0053eae45f45a4435140e85afa5d396a856049b501

1.3 Documentation

The smart contracts in scope are lacking code documentation. Appropriate documentation will improve the overall quality of the project and aid the codebase in terms of clarity of understanding. Finally, all observations are explicitly based on the information available in the provided codebase and whitepaper.

1.4 Review Notes

The majority of the code optimization recommendations highlighted in the observations list refer to best practices and inefficiencies. The observations of type 'Overpowered Design' can place increased risk on investors as a result of centralized access permissions and transfer flow controls (if trust issues were to commence).

2 critical issues were found, one with the potential to disrupt the transfer flow and one that can result in unexpected compilation behavior.

The latest version of the codebase has been optimized to follow best practices and a lot of inefficiencies have been fixed. The observations of type 'Overpowered Design' now include additional guards to reduce investor risk. The critical issues have been resolved, and only a few development observations have been left unchanged.

1.5 Recommendations

The codebase in scope should be fixed to reflect the recommendations presented in this document. Take advantage of linters and a styling guide to creating uniformity throughout the codebase. Next, order the smart contract layouts properly to achieve a high code quality standard. Finally, always favor code simplicity over complexity where possible.

The latest version of the codebase now follows the solidity styling guide and the smart contracts are layered properly for clarity of understanding. Finally, previously complex or over-engineered functions have been made simple, increasing the overall quality.

1.6 Disclaimer

It should be noted that this document is not an endorsement of the effectiveness of the smart contracts, rather limited to an assessment of the logic and implementation. This audit should be seen as an informative practice with the intent of raising awareness on the due diligence involved in secure development and make no material statements or guarantees in regards to the operational state of the smart contract(s) post-deployment. Chain Integrity (Consultant) do not undertake responsibility for potential consequences of the deployment or use of the smart contract(s) related to the audit.

For clarity of understanding, observations are arranged from critical to informational. The severity of each issue is evaluated based on the risk of exploitation or other unexpected behavior.

Critical

An issue flagged as critical means that it can affect the smart contract in a way that can cause serious financial implications, catastrophic impact on reputation, or disruption of core functionality.

High

An Issue flagged as high means that it can affect the ability of the smart contract to function in a significant way i.e. lead to broken execution flows or cause financial implicants.

Medium

An Issue flagged as medium means that the risk is relatively small and that the issue can not be exploited to disrupt execution flows or lead to unexpected financial implications.

Undetermined

An issue flagged as undetermined means that the impact of the discovered issue is uncertain and needs to be studied further.

Low

An Issue flagged as informational does not pose an immediate threat to disruption of functionality, however, it should be considered for security best practices or code integrity.

2.2 Observations List

ID	Title	Type	Severity
OBSN-01	**Substraction Overflow On Withdrawal**	Arithmethic Operations	⌃⌃
OBSN-02	**Inconsistent Compiler Versioning**	Compiler Error	⌃⌃
OBSN-03	**Time-locked Investor Allocations Pull**	Overpowered Design	⌃
OBSN-04	**Staking Reward Allocations Pull**	Overpowered Design	⌃
OBSN-05	**Staking Reward Nullification**	Overpowered Design	⌃
OBSN-06	**Staking Reward Manipulation**	Overpowered Design	⌃
OBSN-07	**Staking Reward Multiplier Manipulation**	Overpowered Design	⌃
OBSN-08	**Time-locked Allocations Token Dust**	Arithmethic Operations	⌃
OBSN-09	**Global Time-lock Duration Fixation**	Logical Issue	—
OBSN-10	**Improper Naming Or Implementation**	Duplicate Code	⌄
OBSN-11	**Unused Global Storage Variable**	Dead Code	⌄
OBSN-12	**Inconsistent Context Helper Use**	Code Optimization	⌄
OBSN-13	**Library Code Bloat**	Code Optimization	⌄
OBSN-14	**Improper Error Handling**	Code Optimization	⌄
OBSN-15	**Explicit Function Visibility Markings**	Code Optimization	⌄
OBSN-16	**Reasonable Sanity Checks**	Code Optimization	⌄
OBSN-17	**Avoidable Code Complexity**	Code Optimization	⌄
OBSN-18	**Inconsistent Styling**	Code Optimization	⌄

Unchanged **Improved** **Fixed**

2.3 Observations Review

OBSN-01

Location: TokenTimelock.sol

Explanation:

When an allocation has been deposited for a particular beneficiary with a portion being unvested, and when a beneficiary wants to withdraw tokens, then the unvested amount is added on top of the total deposit causing a subtraction overflow, this ultimately results in phantom balancing (more tokens needed than available).

Recommendation:

When the unvested amount is based on the total deposit for a particular beneficiary,, then make sure to include the unvested withdrawal in primary calculations, otherwise, the allocated tokens will be frozen until more tokens have been added to the balance of the contract or until the owner executes an emergency withdrawal.

OBSN-02

Location: TokenTimelock.sol

Explanation:

The inheritance graph should be tested with the same compiler version and fixed to ensure that the smart contract does not unexpectedly become affected by undiscovered bugs from other compiler versions.

The compiler versioning has a large range (i.e. >=0.6.0 <0.8.0) and several earlier versions that should be supported in this particular range, will fail when compiling the code.

Recommendation:

Lock the pragmas to the version which has been used for the development of the smart contract (preferably the version which has been used to test the functionality of the smart contract the most). Some of the inherited contracts are outdated based on the chosen compiler version resulting in codebase bloat from unnecessary libraries (i.e. SafeMath).

OBSN-03

Location: TokenTimelock.sol

Explanation:

The owner is capable of withdrawing all the tokens which should be allocated to vested beneficiaries. This means that a certain amount of trust is necessary due to centralized access permissions and control of the transfer flow.

There is nothing wrong with the implementation from an inventors perspective, however, it does bring an increased risk of investment which is the primary reason for the highlight.

OBSN-04

Location: LPStaking.sol

Explanation:

The owner is capable of withdrawing all the tokens which should be allocated to staking rewards. This means that a certain amount of trust is necessary due to centralized access permissions and control of the token transfer flow.

There is nothing wrong with the implementation from an inventors perspective, however, it does bring an increased risk of investment which is the primary reason for the highlight.

OBSN-05

Location: LPStaking.sol

Explanation:

The emergency withdrawal will nullify unclaimed staking rewards. There is no reason to use the emergency withdrawal, however, if the owner were to withdraw all the tokens allocated to staking rewards, then the emergency withdrawal would be the only way to return staked LP tokens.

There is nothing wrong with the implementation from an inventors perspective, however, it does bring an increased risk of investment which is the primary reason for the highlight.

OBSN-06

Location: LPStaking.sol

Explanation:

The owner is capable of updating (and nullify) the staking rewards allocated per block. This means that a certain amount of trust is necessary due to centralized access permissions and control of the token transfer flow.

There is nothing wrong with the implementation from an inventors perspective, however, it does bring an increased risk of investment which is the primary reason for the highlight.

OBSN-07

Location: LPStaking.sol

Explanation:

The owner is capable of updating the rank of a player and in parallel also their tier and staking reward multiplier. This means that a certain amount of trust is necessary due to centralized access permissions and control of the token transfer flow.

There is nothing wrong with the implementation from an inventors perspective, however, it does bring an increased risk of investment which is the primary reason for the highlight.

OBSN-08

Location: TokenTimelock.sol

Explanation:

The interplay between percentage calculations and EVM limitations (i.e. rounding) results in token dust accumulation, a pool of tokens that can not be withdrawn by the beneficiary, resulting in an inaccurate allocation of tokens.

Recommendation:

Clean up the total vested allocation during the last period. The beneficiary should withdraw the calculated proportion as well as token dust that might have been accumulated from percentage roundings.

OBSN-09

Location: TokenTimelock.sol

Explanation:

The time-locked vesting periods vary depending on the pool, however, all beneficiaries of a pool follow the same vesting scheme as they are global. This means that beneficiaries added the day before the end of the vesting period, will be able to claim the total deposit after just a day.

Recommendation:

When all beneficiaries are added to a particular pool during the same transaction, then this observation can be ignored, however, if new beneficiaries are added over time, then the correct approach would be to track the vested duration on an individual basis.

OBSN-10

Location: LPStaking.sol

Explanation:

The removeLiquidity() function is a duplicate of the withdrawRpg() function unless the former is supposed to remove staked LP tokens which would mean the implementation is incorrect as it removes staking rewards instead of staked LP tokens.

Recommendation:

Keep only one of the two before-mentioned functions, or fix the implementation of the removeLiquidity() function to correctly implement the assumption. Being able to remove staked LP tokens, places an increased risk on stakers.

OBSN-11

Location: LPStaking.sol

Explanation:

In the LPStaking.sol smart contract, there is a global storage variable named 'pull' which serves no functional purpose. The variable is never utilized and exists as dead code.

Recommendation:

Avoid source code bloating by removing the variable since the storage location is never being pointed to. If the variable will be used then assign it to the appropriate activities.

OBSN-12

Location: LPStaking.sol

Explanation:

The Context.sol smart contract is a well-known helper smart contract which protects from direct access to low-level calls such as msg.sender and it is primarily used for metadata-related libraries. When the Context.sol smart contract is inherited, then the _msgSender() function should be used instead of msg.sender whenever possible.

Recommendation:

The Context.sol is inherited in all the major smart contracts part of the codebase, however, the functionality is only implemented occasionally, and mixed with direct msg.sender pointers. Change the msg.sender pointers to _msgSender(), when a function access the msg.sender more than once in the same function call, then store _msgSender() in a local variable and use the variable for the remainder. Alternatively, remove the Context.sol inheritance.

OBSN-13

Location: *

Explanation:

The inheritance graph should be tested with a specific compiler version and ultimately fixed to ensure that the smart contract does not accidentally get affected by undiscovered bugs from other compiler versions.

Recommendation:

Lock the pragmas to the version which has been used for the development of the smart contract (preferably the version which has been used to test the functionality of the smart contract the most). **Some of the inherited contracts are outdated based on the chosen compiler version resulting in codebase bloat from unnecessary libraries** (SafeMath).

OBSN-14

Location: *

Explanation:

When handling errors, it is a best practice to fail as early as possible and not fail forcefully from built-in vulnerability safeguards. Exposed functions such as getBeneficiaryStructs() and getTokensAvailable() can fail earlier via proper implementation of sanity checks which ensures that structs, pools, et al. exists before continuing.

Recommendation:

Implement conditional structures such as require(...) to ensure that the data structure exists before continuing the following expressions in the function, especially for publically exposed functions.

OBSN-15

Location: *

Explanation:

Mark the visibility of the implemented functions explicitly (to ease the process of catching incorrect assumptions concerning operational accessibility and to potentially reduce gas costs). The visibility modifiers of the following functions can be further restricted from public to external because the functions are never accessed internally:

LPStaking.sol:
withdrawRpg()
deposit()
withdraw()
processRankingUpdate()
getUserInfo()
emergencyWithdraw()
removeLiquidity()

TokenTimelock.sol
initPools()
addPool()
bulkUploadDeposits()
withdraw()
emergencyWithdraw()
getBeneficiaryStructs()
getSingleBeneficiaryStruct()
getPools()
getTokensAvailable()

ethBridge.sol:
setOwnersForFee()
updateBaseFee()
writeTransaction()
withdrawTokens()
withdrawETHer()

Recommendation:

Change the visibility modifiers of the previously mentioned functions from being public to external for code correctness and to enforce call data on function parameters which will reduce gas costs.

OBSN-16

Location: *

Explanation:

The use of sanity checks allows functions to fail early and to prevent human errors. Next, it improves the overall quality of the code and it is a best practice concerning code correctness and code integrity. The following functions can be further restricted with sanity checks:

LPStaking.sol
updateBaseFee(uint256 _feeWei)
getMapCount()
getSingleBeneficiaryStruct()
getBeneficiaryStruct()
getPools()
getTokensAvailable()
emergencyWithdraw()

TokenTimelock.sol
addManager()
removeManager()
changeTokenPerBlock()
deactivateFarm()
reactivateFarm()
getUserInfo()

Recommendation:

Add sanity checks to the previously mentioned functions for code correctness and code integrity and to prevent human errors. The sensitive nature of smart contracts encourages the use of a large number of sanity checks, especially when hosted on a network with low gas costs.

OBSN-17

Location: TokenTimelock.sol

Explanation:

Avoid complex data structures and loops where possible. The getPools() function creates a memory array for looping over indexes to return all the added pools but the array could just be returned with no looping and ease readability while also increasing performance.

Recommendation:

Return poolNamesArray instead of loading a memory array and looping over indexes.

OBSN-18

Location: *

Explanation:

The smart contracts do not implement a consistent styling guide and this affects the readability of the codebase. Underscore prefixes are commonly used to differentiate between visibility and access levels (or argument inputs), however, it is used inconsistently throughout the smart contracts. Finally, the order of functions and modifiers should be consistent as well, as it all adds to the overall quality of the codebase.

Recommendation:

Consider implementing the styling guide outlined in the Solidity documentation (not only in terms of naming conventions but also in regards to order of layout, order of functions, and order of function modifiers). Adhering to a styling guide makes it easier to mentally build an overview of the codebase and the exposed endpoints (attack surface).

All security issues found during the assessment have been corrected and the smart contracts in scope pass our auditing process.

Tokenization is one of the most important components of decentralized finance and the Revolve Games project contains multiple products that in synergy bring a new gamified decentralized finance experience.

The smart contract(s) in scope had some issues which have been fixed by the team and the overall quality of the codebase has been improved.

A. Source Code Fingerprints

FILE	FINGERPRINT (SHA-256 Checksum)
IERC20.sol	3823104da5fbc0ca11436b529e590c60278d5a7db3fd9568755c3a427d9ba2ee
SafeMath.sol	56eadee672961b0958e4f01d67fa823dd0b4796a5450da33839deda6cdd32b6e
IRandom.sol	e9a745edf8e3c92d4b5656d604e1dfdb87ae4b773139fe010feb96d7cb9c4b16
Context.sol	2db0ae7223ec5b069dc783e5c1234cd4d69772abdc02a3aa6ddb18a1dce66788
Ownable.sol	41720103d3e5154f61bd0f42ba6e0efa05a53755c2c643769ed9466dc9a32373
multiOwnable.sol	a9dc311c856c9e3255755e3b0693510677b2847d040342357944617e0ff2c141
SafeMathChainlink.sol	577f3b8f8f8a66cb67c1acb83d07c2420eef07ad68e13bbf14e4def316d1e0a4
LinkTokenInterface.sol	4d0f8cde26997b7976ba03fb8b34a1c057c327754dc25033a8dac73b24e5e097
VRFRequestIDBase.sol	546bdae2c0504b4eb9795061d711a491a4c80acb1e3839c57f7bbcbdc94a12da
VRFConsumerBase.sol	683f05e01ab7d5fc62f92fe2f4fac561d26410efef698aa3dc8aca6148fd4993
ethBridge.sol	1eb84e9860a0ddc28d329c31500c79172c065c94196f7aeba69c97320cc046d4
random.sol	d25a981dbed17855a79e4d00c8a6115a6bf2df7bda45352f6eead3cfa363b8c7
LPStaking.sol	a96ad9ba3d6e10db512c38a4bc8caaee697c7464e87a2cbc29dc37c33a91398f
TokenTimelock.sol	d4784ded65e400168a963c0053eae45f45a4435140e85afa5d396a856049b501