Chain Integrity

The Web 3.0 Ethical Hacking Company

Revolve Games

Smart Contract Audit Deliverable

Date:	Jan 13, 2022
Version:	2

Table of Contents

1.1 Summary

Project

Name	Revolve Games
Description	Staking, NFTs, RNG, Oracles
Platform	Binance Smart Chain (BSC)
Codebase	*
Commit	*

Engagement

Delivered	Jan 13, 2021 (Updated Codebase)
Methods	Static Analysis, Manual Review, TM, RA.
Consultants	2
Timeline	3 Days (Review Update)

Observations

Total	21	**Status**
Critical	2	Fixed (2/2)
High	9	Fixed (7/9), Unchanged (2/9)
Medium	3	Fixed (2/3), Unchanged (1/3)
Low	7	Fixed (4/7), Improved (1/7), Unchanged (2/7)
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.
- Does not hide potential back doors and implements sanity checks where suitable.

1.2 Scope

File	Fingerprint (SHA-256 Checksum)
Controller.sol	849786cc6eaac2fedc93714e38744a6a8bfa351fadfbc557712dcdcd446d811c
Staking.sol	624d9f2210b1eefa07bbfc931c83726fe5606447aa78ef7073c1f03d769dcbb1
Oracle.sol	4066191d989d36d74de8477826c081b26e8bf3e2735c29a13494193fc966c82d
NFT.sol	0b7de4de8ebb74dcc267dd06a5bd1bc8e492b353c6a57ca967e690890704176a

1.3 Documentation

The smart contract in scope is documented partially but the naming style makes the remaining code easy to comprehend. Finally, all observations are explicitly based on the information available in the provided codebase and whitepaper.

1.4 Review Notes

The gas optimization recommendations emphasized in the observations list refers to best practices and code inefficiencies. Furthermore, notice the classification type 'Overpowered Design' in the appendix A - this particular type places an increased risk on investors as a result of the design patterns used throughout the codebase .

1 observation ranked as critical was found related to the controller design pattern due to an overpowered design and **2** observations ranked as high were found leading to behavioral inconsistensies and cross-chain calls which might result in composability issues.

1.5 Recommendations

The codebase in scope should be fixed to conform with the recommendations presented in this assessment. Optimize the code to lower the gas costs associated with the use of the products. Furthermore, review and fix the compiler version used throughout the codebase to avoid unexpected behavior, and add appropriate error handling to sensitive segments of the codebase.

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.

Detailed Analysis

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	**Supply Manipulation**	Overpowered Design	⌃⌃
OBSN-02	**Breakable Contract State**	Control Flow	⌃⌃
OBSN-03	**Potential Reentrancy**	Control Flow	⌃
OBSN-04	**Potential Reentrancy**	Control Flow	⌃
OBSN-05	**Potential Reentrancy**	Control Flow	⌃
OBSN-06	**Potential Reentrancy**	Control Flow	⌃
OBSN-07	**Potential Reentrancy**	Control Flow	⌃
OBSN-08	**Lazy Functions**	Logical Issue	⌃
OBSN-09	**Mathematical Inconsistency**	Logical Issue	⌃
OBSN-10	**Centralized Dependency**	Overpowered Design	⌃
OBSN-11	**Owner Manipulation**	Overpowered Design	⌃
OBSN-12	**Unlocked Compiler Version**	Code Correctness	–
OBSN-13	**Unchecked Transfer**	Volatile Code	–
OBSN-14	**Calculation Dusting**	Mathematical Operations	–
OBSN-15	**Overengineered Logic**	Gas Optimisation	⌄
OBSN-16	**Lack Of Sanity Checks**	Volatile Code	⌄
OBSN-17	**Unused Contract Variables**	Dead Code	⌄
OBSN-18	**Duplicated Code**	Dead Code	⌄
OBSN-19	**Naming Conventions**	Coding Style	⌄
OBSN-20	**Explicit Function Mutability**	Gas Optimisation	⌄
OBSN-21	**Duplicated Code**	Gas Optimisation	⌄

⬜ **Unchanged** 🟦 **Improved** 🟩 **Fixed**

2.3 Observations Review

OBSN-01

Location: NFT.sol

Explanation:

The owner can manipulate the supply by changing the controller contract storage value using changeController(...) and point it to another smart contract or external account, allowing for manipulation of the token supply via minting and burning.

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

i

OBSN-02

Location: NFT.sol

Explanation:

DoS when the mint(...) function in Controller.sol reaches an assetCounter value that is already used by the bulkMint(...) function in NFT.sol. The assetCounter data is managed internally in an incremental way that would make it impossible to skip the already used token Id.

Recommendation:

Remove the bulkMint(...) function as defined in **OBSN-01** and **OBSN-04** or manage the tokenId in Controller.sol internally.

OBSN-03

Location: Controller.sol, Staking.sol

Explanation:

There are unresolved effects before an external call in Controller.activateBoosters(...) which will execeiute an external call to another vulnerable function Staking.activateBooster(...).

The Controller.activateBoosters(...) function should revert when the value of AssetsToIds[_globalId].boostersMults = 0 and it is being updated to 0 only after performing an external call to Staking.activateBooster(...) which executes another external call to ERC20.transfer(...) in the Staking.claim(...) function → resulting in a potential reentrancy attack.

Furthermore, the Staking.activateBooster(...) function should revert when the value of assetBoostersToAssetsId[_globalId].endTime < block.timestamp, and it is updating the endTime data in the _weightCorrectorForBoosters(...) function only after performing another ERC20.transfer(...) external call in the claim(...) function → resulting in a potential reentrancy attack.

Recommendation:

Perform the external call only after resolving the internal effects or using the OpenZeppelin reentrancy guard by applying the nonReentrant() modifier to suitable functions.

OBSN-04

Location: Controller.sol, Staking.sol, NFT.sol

Explanation:

There are unresolved effects before an external call in the Controller.unstakeAsset(...) function which will execute **3** more external calls to **3** functions:

Staking.withdraw(...)
NFT.burn(...)
ERC20.transfer(...)

Controller.unstakeAsset(...) depends on the values of AssetsToIds[_globalId].amount and AssetsToIds[_globalId].isExist but the state of the data gets updated only after performing the before-mentioned external calls → resulting in a potential reentrancy attack.

Also, in the Staking.withdraw(...) function, an external ERC20.transfer(...) call is executed in advance of internal effect changes → resulting in a potential reentrancy attack.

Recommendation:

Perform the external call only after resolving the internal effects or using the OpenZeppelin reentrancy guard by applying the nonReentrant() modifier to suitable functions.

OBSN-05

Location: Controller.sol

Explanation:

Multiple unresolved effects before an external call in the Controller.emergencyWithdraw(...) function which will trigger **3** more external calls to **3** functions:

Staking.weightCorrector(...)
Staking.emergencyBoosterClearer(...)
NFT.burn(...)

The Controller.emergencyWithdraw(...) function also depends on the values of AssetsToIds[_globalId].amount and AssetsToIds[_globalId].isExist, contract storage data which gets updated only after performing the before-mentioned external calls → resulting in a potential reentrancy attack.

Recommendation:

Perform the external call only after resolving the internal effects or using the OpenZeppelin reentrancy guard by applying the nonReentrant() modifier to suitable functions.

OBSN-06

Location: Controller.sol

Explanation:

There are unresolved effects before an external call in the Controller.fulfillAsset(...) function where logic depends on the values of AssetsToIds[_globalId].isStaked, data which gets updated only after performing an external call to Staking.deposit(...) → resulting in a potential reentrancy attack.

Recommendation:

Perform the external call only after resolving the internal effects or using the OpenZeppelin reentrancy guard by applying the nonReentrant() modifier to suitable functions.

OBSN-07

Location: Controller.sol

Explanation:

Unresolved effects before an external call in the Controller.syncAsset(...) function, the deposit function depends on the values of AssetsToIds[_globalId].levelsUpdateAmount and AssetsToIds[_globalId].boostersUpdateAmount, data which gets updated only after performing an external call to Staking.weightCorrector(...) and a WETH transfer → resulting in a potential reentrancy attack.

Recommendation:

Perform the external call only after resolving the internal effects or using the OpenZeppelin reentrancy guard by applying the nonReentrant() modifier to suitable functions.

OBSN-08

Location: NFT.sol

Explanation:

The external functions bulkMint(...) and bulkBurn(...) are access restricted and limited to Controller.sol, however, the before-mentioned functions are never executed or consumed.

Recommendation:

Remove the bulkMint(...) and bulkBurn(...) functions or change their accessibility to be a reflection of their use.

OBSN-09

Location: Controller.sol, Oracle.sol

Explanation:

The getPrice(...) function returns a default value of 0 for empty price mappings and it can have an unexpected impact on the calculations performed by functions in Controller.sol.

Recommendation:

Revert the function call when the price is not mapped to a non-zero value.

OBSN-10

Location: Controller.sol

Explanation:

The rerollAsset(...) function won't reroll the asset on-chain but it will emit an event that will be picked up by a server maintained by the client to reroll the asset in a centralized way.

Recommendation:

Implement an appropriate key management strategy to protect the private key associated with the manager's wallet and to ensure high availability, security, and maintainability of the backend-server.

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

OBSN-11

Location: Controller.sol

Explanation:

The owner has the power to change the fees associated with asset generation, updating levels, and boosters using the changeTrxFee(...) function. The manager has the power to change the level of an asset to any value using the updateAssetLevel(...) function.

Recommendation:

Implement an appropriate key management strategy to protect the private key associated with the owner's wallet

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

OBSN-12

Location: Controller.sol, Staking.sol, NFT.sol, Oracle.sol

Explanation:

Marking a compiler as wild ^ might lead to unexpected behavior from undiscovered bugs in other compiler versions and break the integrity of the codebase.

Recommendation:

Avoid using a wild compiler version by locking the pragma to the version which has been used for the development of the smart contract (preferably the version used to test the functionality of the smart contract the most).

OBSN-13

Location: Controller.sol

Explanation:

Some token implementations do not revert on a failed transfer, and will return a false value. To prevent friction, it is highly recommended to avoid leaving functions such as transfer(...) and transferFrom(...) unchecked.

Recommendation:

Add sanity checks that require the transfer(...) and transferFrom(...) return values to be true.

OBSN-14

Location: Staking.sol

Explanation:

Performing multiplication after division is not recommended as it can result in truncation and roundings due to the limitations of the Solidity programming language. While calculating the return values in the getInfo(...) function, the Solidity limitations must be taken into consideration.

Recommendation:

Solidity integer division might truncate. As a result, performing multiplication before division can help increase numerical precisioning. Consider re-ordering the arithmetic operations to have division before multiplication whenever possible.

OBSN-15

Location: Controller.sol

Explanation:

The transferBeacon(...) function is only used to emit an external AssetTransfer(...) event call. The code can be further gas optimized by reorganizing the event call and by removing an external function call.

Recommendation:

The AssetTransfer(...) event can be emitted from NFT.sol instead of Controller.sol. This change would remove the external call to NFT.transferBeacon(...) and save gas costs while reducing code complexity.

OBSN-16

Location: Controller.sol

Explanation:

Integrate appropriate sanity checks in functions that might break the system in case of human error or backend issues. The following functions could be hardened with sanity checks for the zero address:

changeOracle(...)
changeRerollFeeReciever(...)
changeFeeReciever(...)

Recommendation:

Add sanity checks that validates the input arguments in the before-mentioned functions to prevent them from updating the associated state variables to the zero address.

OBSN-17

Location: NFT.sol

Explanation:

The global variable baseURI and the mapping type tokensURIs act as dead code since they are not utilized nor do they serve a functional purpose.

Recommendation:

Avoid code bloat by removing the variable and the mapping type since their storage locations are never being pointed to or updated. If the variables are not intended to be used, they might as well be removed.

OBSN-18

Location: NFT.sol

Explanation:

The getOwner(...) function is a duplicate of the same functionality provided by the ownerOf(...) function which is part of the ERC721-standard and already inherited by the contract.

Recommendation:

Keep the ownerOf(...) function and remove the getOwner(...) function to follow industry standards while avoiding code bloat.

OBSN-19

Location: Controller.sol, Staking.sol, NFT.sol

Explanation:

Several functions and variable names have grammatical errors (i.e. rerollFeeReciever instead of rerollFeeReceiver), and the same is true for multiple error-handling strings. Underscore prefixes are commonly used to differentiate between visibility and access levels (or argument inputs), however, in this codebase it is used primarily to differentiate input arguments and return values from contract variables. Next, the order of functions and modifiers should be consistent as well, as it improves the overall quality of the codebase. Finally, the struct data structures do not follow the solidity styling guide as well.

Recommendation:

Implement 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). Check the code for grammatical errors.

OBSN-20

Location: Oracle.sol

Explanation:

The function signature type should be as explicit as possible. When a function is only executed to read a value it should be marked as view or pure depending on the provided input arguments.

Recommendation:

The functions isManager() and isAllowed() do not change the contract state and only read from contract storage, as a result, we recommend restricting the function type from write → read-only by marking them with the view keyword.

OBSN-21

Location: Controller.sol

Explanation:

The onlyAssetOwner(...) custom modifier has a sanity check for whether an asset exists prior to the continuation of the function execution, however, multiple functions with the modifier appended, also has the sanity check as an expression inside of the function, this goes against the nature of modifiers and decorator patterns → to reduce code redundancy.

Recommendation:

Remove the AssetsToIds[_globalId].isExist sanity check from the functions that already have the onlyAssetOwner(...) modifier appended to their function signature, this will reduce code bloat and lower gas costs as well.

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.

The statements made in this report do not constitute legal or investment advice and we should not be held accountable for decisions made based on them.

A. Classification List

TYPE	DESCRIPTION
Gas Optimization	Gas Optimization findings refer to code improvements that do not affect the functionality of the code but execute more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Mathematical Operations	Mathematical Operations details findings related to mishandling of mathematical formulas such as integer overflows, an incorrect order of operations, or percentage precisioning et al.
Logical Issue	Logical Issue findings defines faults in the logic of chained functions or particular expressions, such as an incorrect implementation of incentive designs, vesting schemes, and similar.
Control Flow	Control Flow findings concern the access control imposed on functions, such as overpowered access functions being executable by anyone under certain circumstances or control flow hijackings like reentrancy attacks.
Volatile Code	Volatile Code findings refer to code implementations that behave unexpectedly on particular edge cases that may result in exploitability or sensitive and unreliable code behavior.
Overpowered Design	Owerpowered Design findings describe code that entails a certain amount of trust in centralized entities such as an owner not behaving maliciously and to maintain code integrity.
Language Specific	Language Specific findings are issues that are related to the Solidity programming language such as incorrect usage of the delete keyword or conformity with language limitations.
Coding Style	Coding Style findings are primarily informational and they help to increase the quality of the codebase and easier maintainable by following a consistent styling guide.
Code Correctness	Code Correctness findings refer to functions that should seemingly behave similarly yet contain different code, legacy inheritance graph versioning, explicit visibility markings, etc.

A. Classification List

TYPE	DESCRIPTION
Magical Numbers	Magic Number findings refer to numerical values that are defined in the codebase in their raw format and should otherwise be specified as contract variables to increase code legibility and maintainability.
Compiler Error	Compiler Error findings refer to an issue in the implementation of a segment of the code that renders it impossible to compile using the specified version of the codebase.
Dead Code	Code that otherwise does not affect the functionality of the codebase and can be safely omitted to avoid code bloat and to increase the overall quality of the codebase.

B. Source Code Fingerprints

FILE	FINGERPRINT (SHA-256 Checksum)
IController.sol	2e98b7d67290ddaaf9df086edfc8169dad4d9c9fcf3064d61197aa06dd433c35
IStaking.sol	4cb8c29b5e6b3b67b2cc889ff8920cf0ffac37e9c2e3809c2b8f73eb6566cdde
IOracle.sol	4654c24b35711958ec4b9e6ca2431f5c3dcc526a6f6aadedab77dcc041a3ea7b
INFT.sol	216b9a4774b2d650a7cd368c37a1a93000f4332d38c1f2107939d12e0305ca20
Controller.sol	849786cc6eaac2fedc93714e38744a6a8bfa351fadfbc557712dcdcd446d811c
Staking.sol	624d9f2210b1eefa07bbfc931c83726fe5606447aa78ef7073c1f03d769dcbb1
Oracle.sol	4066191d989d36d74de8477826c081b26e8bf3e2735c29a13494193fc966c82d
NFT.sol	0b7de4de8ebb74dcc267dd06a5bd1bc8e492b353c6a57ca967e690890704176a