Chain Integrity

The Web 3.0 Ethical Hacking Company

Revolve Games

Smart Contract Audit Deliverable

Date: Sep 17, 2021

Version: 1

Executive Summary

This document outlines the security posture of the smart contracts associated with the following Revolve Games (customer) products; LERC20RPG.sol - as evaluated by Adis Begic and Abderrahmen Hanafi, security researchers at Chain Integrity (consultant). The scope of the audit was to analyze and document the codebase provided by the Revolve Games team for security vulnerabilities, code correctness, and risk of investment. The engagement was conducted over 10 business days and the analysis pertains explicitly to the smart contracts in scope.

Contract(s) Status

Passed

Scope of Engagement

The scope of the engagement was to assess the security of the LERC20RPG.sol smart contract. Great emphasis was placed on the inheritance of functionality and erroneous arithmetic calculations. The engagement sought to answer the following:

1	**Is it possible for a malicious actor to cause financial implications or catastrophic impact on the reputation by disrupting the token transfer flow?**
2	**Can the arithmetic calculations be trusted to present the correct values in regards to transferring tokens between external accounts?**
3	**Are investors putting their funds at risk of exploitation by trusting a centralized actor while using the product, and what is the potential risk?**

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.

Table of Contents

Audit Strategy & Applied Techniques

Throughout the audit, prudence was exercised to assess that the smart contract(s):

- Follows best practices with respect to efficient gas consumption, while avoiding unnecessary costs.
- Implements robust functions which are safe from well-known attack vectors i.e. reentrancy.
- The logic and behavior adheres to the associated documentation and code comments.
- Distributions of tokens are designed in a sustainable way and safeguarded from i.e. infinite loops.
- Does not hide potential back doors and implements thorough sanity checks where suitable.

General best practices and industry-standard techniques have been followed to scrutinize the implementation of the revolve codebase during my assessment.

The smart contract has been reviewed line-by-line and concerning security issues have been documented as discovered.

To summarize, the auditing strategy consists largely of manual expertise.

1	Due diligence in evaluating the overall quality of the codebase.
2	Cross-comparison with other, similar smart contracts in respect to wrapped up functionality.
3	Testing the functionality against common and uncommon attack vectors.
4	Comprehensive, manual review of the codebase, line-by-line.
5	Deploying the smart contracts on testnet to run practical tests.

2.1 Summary

The LERC20RPG.sol smart contract in scope functions as a standard ERC20 token but with additional investor protection from the Lossless protocol, an integration that enables the transfer execution flow to be controlled to protect against malicious transfer activities.

Understand that the Lossless protocol is still at an early stage and no public audits have been released detailing the security posture of the project. Understand that there is risk present due to an overpowering design but the exploit motivation (likelihood) is minimal.

2.2 Behavioral Consistency

The codebase behaves as intended by the developer(s). The codebase presented to us can not be exploited by the public to disrupt the behavioral consistency nor the functional state of the smart contracts.

2.3 Overall Risk Severity

The risk of investment for this particular codebase is low.

In terms of threats for this codebase - low to non-existent. The project has a proper value proposition, the drivers behind the project are disclosed, and the team is consistent with delivering milestones.

In terms of risk for this codebase - low to non-existent. There is a risk of fee manipulation, however, it is not supported by exploit motivation. Finally, it is recommended to secure the owner with a strong key management strategy since it is used as a fixed fee endpoint.

Likelihood (0-3): 0
Financial Damage (0-3): 1

3.1 Threat Analysis

Anonymous Drivers:
The Revolve Games team is public, it would be easy to make the team accountable for any malicious activity. The project has reputable industry players as partners.

Overpowered Design:
The LERC20RPG.sol smart contract includes an overpowered owner which is seen as an anti-investment pattern. However, all token transfer activities are wrapped via hooks to the Lossless protocol, and disabling the protocol would require a proposal and a time delay before finality.

Exploit Motivation:
The use-case is relevant in terms of major adoption, decentralized finance, and a tokenized future. The gaming industry is continuously growing and the industry is marked as one that can truly derive value from tokenization.

3.2 Risk Assessment

Stolen:
The transfer fee can be increased to subtract more tokens than documented by exploiting the code from the perspective of an overpowered owner. It is important to put emphasis on the fact that strong walls have been built around all standard ERC20 transfer activities, making it very difficult for any malicious activities to be overlooked.

Frozen:
-

Drained:
Rational vesting schemes are defined in the project documentation.

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.

4.1 Detailed Analysis

Inconsistent Compiler & Inheritance Graph Versioning

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).

4.2 Detailed Analysis

Explicit Function Visibility Markings

Explanation:

Mark the visibility of the implemented functions explicitly (to ease the process of catching incorrect assumptions with respect to 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: removeAllowedContract, addAllowedContract, updateFee, deleteFreeOfChargeAddress, addFreeOfChargeAddress.

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.

4.3 Detailed Analysis

Explicit Global Storage Variable Markings

Explanation:

Declare the implemented variables explicitly (to ease the process of catching incorrect assumptions with respect to operational accessibility and to potentially reduce gas costs).

Based on the current implementation, the following global storage variables can be further restricted with the immutable keyword since they are never updated: percentConst, _pool. Additionally, the variable percentConst can be further restricted with the constant keyword, including it in the bytecode at deployment.

Recommendation:

Add the immutable keyword on both (or constant keyword on percentConst) the previously mentioned global storage variables for code correctness and for gas optimization.

4.4 Overall Considerations

Reasonable Sanity Checks

Explanation:

Full flexibility in regards to updating the fee percentage via the setFee function should be avoided if the updates can be limited to a certain percentage range. Reasonable sanity checks increase the quality of the code and in this particular scenario, it also reduces the financial damage in case of an exploit.

Recommendation:

Add a sanity check that requires the argument provided to the setFee function to be within a certain range (i.e. 1-15% if the maximum fee percentage to ever exist is 15%).

4.5 Overall Considerations

Consistent Styling Guide

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).

No security issues were found during the assessment 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 contracts in scope adhere to best practices in terms of security and it is evident that an effort has been made to deliver a codebase of high quality.

Source Code Fingerprints

FILE	FINGERPRINT
Address.sol	2a5a98e5f30a53b139a91f8491e0f6d666acddd9f3a619e9b2512f6244811f0c
SafeMath.sol	314afa0026de0a0604043759bcd89ffc9a70b519a6a015bd3250bd6ae05238cd
IERC20.sol	7efdd9d5042a4ab3a5f0cfa010488a01aabef75aef41554d4ca42a441eac0e3c
ERC20.sol	527d889dc93fa36dd7c9cfc8d2026db1d58411d974de1d6e381dc8bbdd2a1611
LERC20.sol	910a5909d8a055026d3c0ac632c3282afb1c401116c363c6e4162469e7defc2f
Context.sol	2db0ae7223ec5b069dc783e5c1234cd4d69772abdc02a3aa6ddb18a1dce66788
Ownable.sol	41720103d3e5154f61bd0f42ba6e0efa05a53755c2c643769ed9466dc9a32373
LERC20RPG.sol	5dfd866149dbfd89012176dda7f81e9bbdc9fde95b4e51dab1d1b5ce2b9bde5b