AuditBase

LlamaLend: Smart Contract Audit & Code Review

Prepared for: 0xngmi, LlamaLend

Date: October 18, 2022

Table of Contents

Disclaimer

Audit Information

Name	Security audit and analysis of LlamaLend Smart Contracts
Type of Contracts	NFT Lending
Platform	Ethereum
Language	Solidity
Testing Methods	Architecture Review, Static Analysis, Functional Analysis, Manual Review
Website	https://github.com/LlamaLend/contracts
Date	October 18, 2022

Executive Summary

CryptoGuardian has completed an audit for LlamaLend, an NFT lending protocol on Ethereum. The scope of this audit was limited to the smart contracts powering the NFT lending platform.

Scope

Deployed Contract	N/A
Documentation	https://github.com/LlamaLend/contracts
Unit Tests	https://github.com/LlamaLend/contracts/tree/master/test
Contracts	ListNfts.sol, LendingPool.sol, LlamaLendFactory.sol

Audit Opinion

In our opinion, the audited contracts comply with Solidity standards and intended functionality. The implementation and deployment of the audited contracts provides a reasonably safe execution environment and does not pose imminent risks to users.

Certain protocol design decisions do expose Lending Pool owners to a high degree of price risk and bad debt exposure. However, this is the intent of the platform given the niche use-case.

Security Definitions

Risk	**Description**
Critical	Critical flaws can result in the loss of assets or the alteration of data and are typically simple to exploit.
High	High-level vulnerabilities are challenging to attack, but they can have a big impact on how smart contracts work, like giving the public access to essential features.
Medium	Although medium-level vulnerabilities should be fixed, they cannot result in the loss of assets or the manipulation of data.
Low	Low-level flaws are typically caused by bits of unneeded, old code that don't have a big influence on the execution.

Findings

Critical

No critical severity issues were found.

High

No high severity issues were found.

Medium

- Protocol Design:
    - High likelihood of bad debt based on protocol design.
    - Oracle manipulation.
- Integration test FAILING: “ListNfts > 1) partial range, erc721”

Low

- Protocol Design: Time-based loan repayment has mild vulnerability to miner manipulation.

Attack Vectors

High likelihood of bad debt based on protocol design.

Given that liquidation can only be executed after loan expiry, borrowers are incentivized to not make payments and walk away from their open positions given a high degree of volatility.

This leaves the lending pool owner open to a high degree of bad debt for a potentially long time.

Bad debt exposure could be minimized by allowing loans to be liquidated based on loan length *or* loan to value (LTV).

Oracle Manipulation

By relying on a centralized oracle system, the lending pool owners are vulnerable to rapid price manipulation, potentially leading to undesirable loans being executed on chain.

Time-based loan repayment has mild vulnerability to miner manipulation.

This low-level vulnerability requires trust in miners to include loan repayments in blocks. For desirable NFT collections, there is MEV to be had in not including a transaction that is within seconds from expiry.

Medium & High Level Reported in Static Analysis

LendingPool.sol

LendingPool.currentAnnualInterest(uint256) (contracts/LendingPool.sol#267-275) uses a dangerous strict equality:

- address(this).balance + totalBorrowed == 0 (contracts/LendingPool.sol#269)

Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Safe to ignore this error because the pool legitimately will be zero at times and cannot be below zero due to being set as an unsigned integer.

ListNfts.sol

ListNfts.getOwnedNfts(address,IERC721,uint256,uint256) (contracts/ListNfts.sol#9-75) ignores return value by nftContract.ownerOf(start) (contracts/ListNfts.sol#53-61)

Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

This is a parsing issue in Slither - the return value is included as a return statement in the try/catch block.

LLamaLendFactory.sol

LlamaLendFactory.repay(LlamaLendFactory.LoanRepayment[]) (contracts/LlamaLendFactory.sol#41-51) sends eth to arbitrary user

Dangerous calls:
- LendingPool(loansToRepay[i].pool).repay{value: address(this).balance}(loansToRepay[i].loans,msg.sender) (contracts/LlamaLendFactory.sol#45)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations

Safe to ignore this error because the address being sent Ether is the LendingPool, which is verified upon contract creation.

Disclaimers

AuditBase Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with 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).

Regarding the code's security, the audit offers no claims or guarantees. It also cannot be taken into account as an adequate evaluation of the code's usefulness and safety, its bug-free status, or any other contract clauses. Despite the fact that we did our best in conducting the analysis and putting together this report, it is crucial to note that you shouldn't rely solely on it. Instead, we advise moving forward with a number of independent audits and a public bug bounty program to ensure the security of smart contracts.

This audit does not constitute a formal partnership or relationship with the Company and should not be used to assume a business relationship.

Technical Disclaimer

On a blockchain network, smart contracts are set up and carried out. Hacking vulnerabilities may exist in the platform, the programming language used, and other applications used in conjunction with the smart contract. The explicit security of the audited smart contracts cannot therefore be guaranteed.