d3ploy

d3ploy

SECURITY ASSESSMENT

LEND

August 16th 2022

TABLE OF

Contents

LEGAL

Disclaimer

D3ploy audits are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts d3ploy to perform a security review. D3ploy does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

D3ploy audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort. The report is provided only for the contract(s) mentioned in the report and does not include any other potential additions and/or contracts deployed by Owner. The report does not provide a review for contract(s), applications and/or operations, that are out of this report scope.

D3ploy’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

D3ploy represents an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. D3ploy’s position is that each company and individual are responsible for their own due diligence and continuous security. The security audit is not meant to replace functional testing done before a software release. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent manual audits and a public bug bounty program to ensure the security of the smart contracts.

D3PLOY

Introduction

D3ploy is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

Secure your project with d3ploy

We offer field-proven audits with in-depth reporting and a range of suggestions to improve and avoid contract vulnerabilities. Industry-leading comprehensive and transparent smart contract auditing on all public and private blockchains.

Vunerability checking

A crucial manual inspection carried out to eliminate any code flaws and security loopholes. This is vital to avoid vulnerabilities and exposures incurring costly errors at a later stage.

Contract verification

A thorough and comprehensive review in order to verify the safety of a smart contract and ensure it is ready for launch and built to protect the end-user

Risk assessment

Analyse the architecture of the blockchain system to evaluate, assess and eliminate probable security breaches. This includes a full assessment of risk and a list of expert suggestions.

In-depth reporting

A truly custom exhaustive report that is transparent and depicts details of any identified threats and vulnerabilities and classifies those by severity.

Fast turnaround

We know that your time is valuable and therefore provide you with the fastest turnaround times in the industry to ensure that both your project and community are at ease.

Best-of-class blockchain engineers

Our engineers combine both experience and knowledge stemming from a large pool of developers at our disposal. We work with some of the brightest minds that have audited countless smart contracts over the last 4 years.

PROJECT

Introduction

LEND is simply lending & borrowing, amplified, with real yield value extraction, from protocol, to holder. A lending protocol that generates revenue through facilitating swaps and interest rate differentials, generating revenue on the spread.

LEND will establish pools of algorithmically derived interest rate model, based on current supply and demand of each respective asset. Suppliers and Borrowers of assets interact directly with the protocol in earning and paying a floating interest rate. This is all done without the need to negotiate terms of maturity, interest rate or collateral with any peer or counterparty!

Project Name	*LEND*
Contract Name	*LEND Token*
Contract Address	*-*
Contract Chain	*Mainnet*
Contract Type	*Smart Contract*
Platform	*EVM*
Language	*Solidity*
Network	*BNB Chain (BEP20) Ethereum (ERC20) Polygon (Matic)*
Codebase	*Private GitHub Repository*
Total Token Supply	*1,000.000.000*

INFO

Social

https://www.lend.finance/

https://twitter.com/LEND_finance

https://t.me/lendfinance

-

https://medium.com/lendfinance

https://github.com/tenfinance

contact@lend.finance

94

PASS

AUDIT

Score

Issues	11
Critical	0
Major	1
Medium	3
Minor	3
Informational	4
Discussion	0

All issues are described in further detail on the following pages.

AUDIT Scope

CODEBASE FILES	LOCATION
JumpRateModelV2c.sol	✦ Private Repository
JumpRateModelV2s.sol	✦ Private Repository
Unitroller.sol	✦ Private Repository

REVIEW Methodology

TECHNIQUES

This report has been prepared for LEND to discover issues and vulnerabilities in the source code of the LEND project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic, Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts producedby industry leaders.

The security assessment resulted in findings that ranged from major to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective in the comments below.

TIMESTAMP

Version	*v1.0*
Date	*2022/08/16*
Descrption	*Layout project*
	Architecture / Static security testing
	Summary

KEY Finding

TITLE	SEVERITY	STATUS
Locked Ether Inside A Contract	✦ Major	Pending
Missing Exception On BEP20 Transfer Failure	✦ Medium	Pending
BEP20 Approve Font-Running Attack	✦ Medium	Pending
Array Length Manipulation	✦ Medium	Pending
Use Of SafeMath Library	✦ Minor	Pending
Cheaper Inequalities In Require()	✦ Minor	Pending
Use Of Floating Pragma	✦ Minor	Pending
Block Values As A Proxy For Time	✦ Informational	Pending
In-Line Assembly Detected	✦ Informational	Pending
Function Should Return Struct	✦ Informational	Pending

KEY Finding

TITLE	SEVERITY	STATUS
Presence Of Overpowered Role	✦ Informational	Pending

IN - DEPTH Vulnerabilities

DESCRIPTION

The contract is programmed to receive Ether, but no method was found that allowed the Ether to be withdrawn, i.e., call, transfer, transferFrom, send, or call.value at least once.
Without a withdrawal function, the Ethers will forever be locked inside the contract if the contract's code is not upgradeable leading to loss of funds.

LOCATION

Unitroller.sol L2964-L3102

Issue : Locked Ether Inside A Contract

Level : Major

Remediation : Implement a withdraw function or reject payments (contracts without a fallback function do it automatically).

Alleviation / Retest :

IN - DEPTH Vulnerabilities

DESCRIPTION

Functions of BEP-20 Token Standard should throw in special cases:

- transfer should throw if the _from account balance does not have enough tokens to spend.

- transferFrom should throw unless the _from account has deliberately authorized the sender of the message via some mechanism.

LOCATION

Unitroller.sol L720-L1001; L1112-L1246

Issue : Missing Exception On BEP20 Transfer Failure

Level : Medium

Remediation : The BEP20 standard recommends throwing exceptions in functions transfer and transferFrom.

SafeBEP20 standard can also be used that automatically throws on failure

Alleviation / Retest :

IN - DEPTH Vulnerabilities

DESCRIPTION

The approve() method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account.
This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account.
Meanwhile, if the sender decides to change the amount and sends another approve transaction, the receiver can notice this transaction before it's mined and can extract tokens from both the transactions, therefore, ending up with tokens from both the transactions. This is a front-running attack affecting the BEP20 Approve function.

LOCATION

Unitroller.sol L1440-L1445

Issue : BEP20 Approve Font-Running Attack

Level : Medium

Remediation : Only use the approve function of the BEP20 standard to change the allowed amount to 0 or from 0 (wait till transaction is mined and approved).

Token owner just needs to make sure that the first transaction actually changed allowance from N to 0, i.e., that the spender didn't manage to transfer some of N allowed tokens before the first transaction was mined. Such checking is possible using advanced blockchain explorers such as [bscscan.io] (https://bscscan.io/).

Another way to mitigate the threat is to approve token transfers only to smart contracts with verified source code that does not contain logic for performing attacks like described above, and to accounts owned by the people you may trust.

Alleviation / Retest :

IN - DEPTH Vulnerabilities

DESCRIPTION

A smart contract's data (e.g., storing the owner of the contract) is persistently stored at some storage location (i.e., a key or address) on the EVM level. The contract is responsible for ensuring that only authorized user or contract accounts may write to sensitive storage locations. If an attacker is able to write to arbitrary storage locations of a contract, the authorization checks may easily be circumvented. This can allow an attacker to corrupt the storage; for instance, by overwriting a field that stores the address of the contract owner.

The length of the dynamic array is changed directly. In this case, the appearance of gigantic arrays is possible and it can lead to a storage overlap attack (collisions with other data in storage)

LOCATION

Unitroller.sol L1812; L1921; L 1939; L2171; L2181; L2183: L2197; L2395-L2396; L2401-L2404

Issue : Array Length Manipulation

Level : Medium

Remediation : Only use the approve function of the BEP20 standard to change allowed amount to 0 or from 0 (wait till transaction is mined and approved).

Alleviation / Retest :

IN - DEPTH Vulnerabilities

DESCRIPTION

SafeMath library is found to be used in the contract. This increases gas consumption than traditional methods and validations if done manually.

Also, Solidity 0.8 includes checked arithmetic operations by default, and this renders SafeMath unnecessary.

LOCATION

JumpRateModelV2c.sol L233
JumpRateModelV2s.sol L233

Issue : Use Of SafeMath Library

Level : Minor

Remediation : We do not recommend using SafeMath library for all arithmetic operations. It is good practice to use explicit checks where it is really needed and to avoid extra checks where overflow/underflow is impossible.

The compiler should be upgraded to Solidity version 0.8.0+ which automatically checks for overflows and underflows.

Alleviation / Retest :

IN - DEPTH Vulnerabilities

DESCRIPTION

The contract was found to be doing comparisons using inequalities inside the require statement. When inside the require statements, non-strict inequalities (>=, <=) are usually costlier than the strict equalities (>, <).

LOCATION

JumpRateModelV2c.sol L65; L80
JumpRateModelV2s.sol L65; L80

Issue : Cheaper Inequalities In Require()

Level : Minor

Remediation : It is recommended to go through the code logic, and, if possible, modify the non-strict inequalities with the strict ones to save ~3 gas as long as the logic of the code is not affected.

Alleviation / Retest :

IN - DEPTH Vulnerabilities

DESCRIPTION

Solidity source files indicate the versions of the compiler they can be compiled with using a pragma directive at the top of the solidity file. This can either be a floating pragma or a specific compiler version.

The contract was found to be using a floating pragma which is not considered safe as it can be compiled with all the versions described

LOCATION

Unitroller.sol L04
JumpRateModelV2c.sol L04
JumpRateModelV2s.sol L04

Issue : Use Of Floating Pragma

Level : Minor

Remediation : It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee. The developers should always use the exact Solidity compiler version when designing their contracts as it may break the changes in the future.

pragma solidity ^0.4.17; not recommended -> compiles with 0.4.17 and above
pragma solidity 0.8.4; recommended -> compiles with 0.8.4 only

Alleviation / Retest :

IN - DEPTH Vulnerabilities

DESCRIPTION

Contracts often need access to time values to perform certain types of functionality. Values such as block.timestamp and block.number can be used to determine the current time or the time delta. However, they are not recommended for most use cases.

For block.number, as Ethereum block times are generally around 14 seconds, the delta between blocks can be predicted. The block times, on the other hand, do not remain constant and are subject to change for a number of reasons, e.g., fork reorganizations and the difficulty bomb.

Due to variable block times, block.number should not be relied on for precise calculations of time

LOCATION

Unitroller.sol L1510

Issue : Block Values As A Proxy For Time

Level : Informational

Remediation : Smart contracts should be written with the idea that block values are not precise, and their use can have unexpected results. Alternatively, oracles can be used.

Alleviation / Retest :

IN - DEPTH Vulnerabilities

DESCRIPTION

Inline assembly is a way to access the Ethereum Virtual Machine at a low level. This bypasses several important safety features and checks of Solidity. This should only be used for tasks that need it and if there is confidence in using it.

Multiple vulnerabilities have been detected previously when the assembly is not properly used within the Solidity code; therefore, caution should be exercised while using them.

LOCATION

Unitroller.sol L288-L300; L305-L311; L316-L348; L370-L532; L971; L974; L1232-L1245; L1272-1303; L1564-L1594; L1621-L1651; L2239-L2254: L2265-L2334; L2504-L2521; L2708-L2716; L2724-L2752

Issue : In-Line Assembly Detected

Level : Informational

Remediation : Avoid using inline assembly instructions if possible because it might introduce certain issues in the code if not dealt with properly because it bypasses several safety features that are already implemented in Solidity.

Alleviation / Retest :

IN - DEPTH Vulnerabilities

DESCRIPTION

Internal or private functions were detected to be returning multiple values.

Consider using a struct instead of multiple return values for internal or private functions. It can improve code readability.

Most decentralized applications and games need to store data on the blockchain, so they have to interact with the storage.

Minimizing storage costs is a major part of Gas optimization for your smart contracts.

LOCATION

Unitroller.sol L1494; L1499; L1502

Issue : Function Should Return Struct

Level : Informational

Remediation : Use struct for internal or private function, which returns several parameters and improves code readability. This also helps in reducing the Gas used.

Alleviation / Retest :

IN - DEPTH Vulnerabilities

DESCRIPTION

The overpowered owner (i.e., the person who has too much power) is a project design where the contract is tightly coupled to their owner (or owners); only they can manually invoke critical functions.

Due to the fact that this function is only accessible from a single address, the system is heavily dependent on the address of the owner. In this case, there are scenarios that may lead to undesirable consequences for investors, e.g., if the private key of this address is compromised, then an attacker can take control of the contract.

LOCATION

Unitroller.sol L1278

Issue : Presence Of Overpowered Role

Level : Informational

Remediation : We recommend designing contracts in a trust-less manner. For instance, this functionality can be implemented in the contract's constructor. Another option is to use a MultiSig wallet for this address.

For systems that are provisioned for a single user, you can use [Ownable.sol](https://github.com/OpenZeppelin/openzeppelincontracts/blob/release-v2.5.0/contracts/ownership/Ownable.sol).

For systems that require provisioning users in a group, you can use [@openzeppelin/Roles.sol] (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v2.5.0/contracts/access/Roles.sol) or [@hq20/Whitelist.sol] (https://github.com/HQ20/contracts/blob/v0.0.2/contracts/access/Whitelist.sol)

Alleviation / Retest :

SOURCE Code

Private GitHub Repository

REPORT Appendix

FINDING CATEGORIES

The assessment process will utilize a mixture of static analysis, dynamic analysis, in-depth manual review and/or other security techniques.

This report has been prepared for LEND project using the above techniques to examine and discover vulnerabilities and safe coding practices in LEND's smart contract including the libraries used by the contract that are not officially recognized.

A comprehensive static and dynamic analysis has been performed on the solidity code in order to find vulnerabilities ranging from minor gas optimizations to major vulnerabilities leading to the loss of funds.

Various common and uncommon attack vectors will be investigated to ensure that the smart contracts are secure from malicious actors. The testing methods find and flag issues related to gas optimizations that help in reducing the overall gas cost It scans and evaluates the codebase against industry best practices and standards to ensure compliance It makes sure that the officially recognized libraries used in the code are secure and up to date.

AUDIT SCORES

D3ploy Audit Score is not a live dynamic score. It is a fixed value determined at the time of the report issuance date.

D3ploy Audit Score is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports and scores are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts d3ploy to perform a security review.

d3ploy

WEBSITE d3ploy.co

@d3ploy_ TWITTER