UniLayer

LAYERx v3

SMART CONTRACT AUDIT

06.02.2021

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of UniLayer. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (02.02.2021)	Layout
0.5 (02.02.2021)	Automated Security Testing Manual Security Testing
0.8 (03.02.2021)	Testing SWC Checks
0.9 (04.02.2021)	Summary and Recommendation
1.0 (04.02.2021)	Final document
1.2 (06.02.2021)	Deployed Contract added

2. About the Project and Company

Company address:

The Unilayer Project LTD
44 Church St.
St. John’s
Antigua & Barbuda

Website: https://unilayer.info
DApp: https://unilayer.app
GitHub: NA
Twitter: https://twitter.com/unilayer_
Telegram: https://t.me/Unilayer
Etherscan (LAYER Token): https://etherscan.io/token/0x0fF6ffcFDa92c53F615a4A75D982f399C989366b
Medium: https://medium.com/@UniLayer/unilayer-next-generation-decentralised-trading-platform-524e458ec7ff
Discord: https://discord.com/invite/BV5y3dd

2.1 Project Overview

UniLayer is a next generation decentralised trading platform built on top of Uniswap that enables key features for professional-level trading with it’s LAYER utility token, focusing on automated swaps and liquidity management, flash staking, charts and analytics, live order books, and a lot more. On top of these features, the LAYER token is used to facilitate transactions on UniLayer where all transaction fees are transferred to a token pool. 92% of fees will be distributed to stakers of the platform in addition to liquidity providers to the ETH/LAYER liquidity pool, with the remaining 8% going to the foundation as a reserve.

What is LAYERx?

A new token that will have 0 premine, governance features integrated in the Unilayer platform, and can only be minted while staking LAYER in the staking platform or by Providing liquidity on Uniswap.

LAYERx Tokenomics

- Total Supply 40,000
- Inflation rate 10,000 year for a period of 4 years
- Can be minted while staking LAYER in the staking platform
- Can be minted while providing Liquidity to both LAYER/USDT & LAYER/ETH LPs in Uniswap

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is	An observation that does not determine a level of risk

		not effecting any of the code.	

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

4.2 Used Code from other Frameworks/Smart Contracts (direct imports)

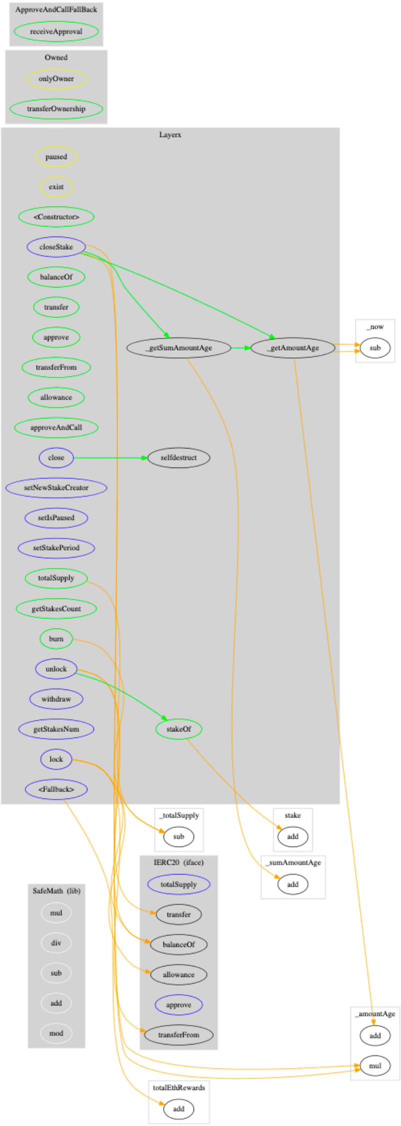
1. SafeMath.sol (0.5.0)
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.3.0/contracts/math/SafeMath.sol
2. IERC20.sol (0.5.0)
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.3.0/contracts/token/ERC20/IERC20.sol

4.3 Tested Contract Files

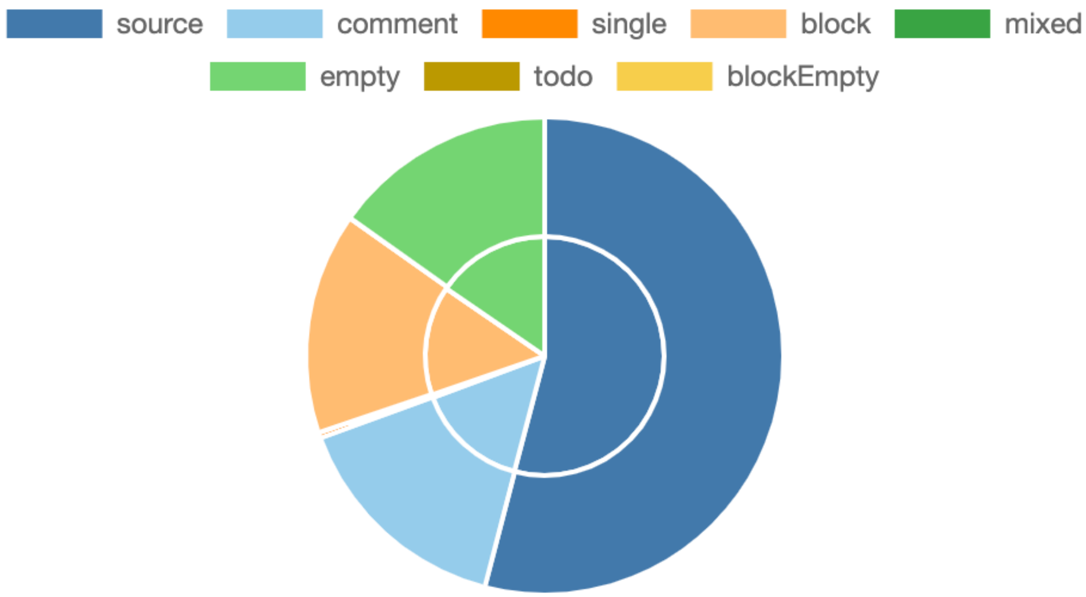
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

File	Fingerprint (MD5)
LayerX_v3.sol	27d8e523beb41e91a005853f4d792ee7

4.4 Metrics / CallGraph



4.5 Metrics / Source Lines



4.6 Metrics / Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`^0.5.0`		`yes`	**** (0 asm blocks)	`yes`

📤 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧮 Uses Hash Functions	🔖 ECRecover	🌀 New/Create/Create2
`yes`					

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝📚🔍	LayerX_v3.sol	4	1	451	434	277	82	224	💰💣📤🔆
📝📚🔍	**Totals**	**4**	**1**	**451**	**434**	**277**	**82**	**224**	💰💣📤🔆

5. Scope of Work

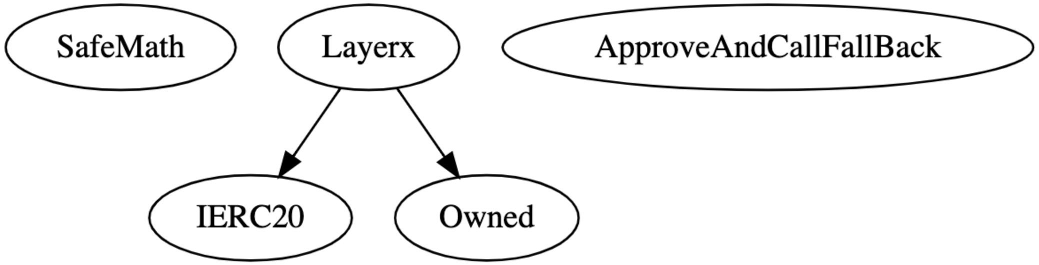
The Unilayer team provided us with the files that needs to be tested. The scope of the audit is the LayerX_v3 contract.

Following contracts with the direct imports been tested
layerx_v3.sol

The team put forward the following assumptions regarding the security, usage of the contracts:

- Checking the overall security of the contracts

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.



5.1 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, Chainsulting's experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES

5.1.1 Selfdestruct owner

Severity: HIGH
Status: ADDRESSED
Update: Doesn’t affect the overall security in that use case too much
File(s) affected: layerx_v3.sol

Attack / Description	Code Snippet	Result/Recommendation

Use of selfdestruct: Can block calling contracts unexpectedly. Be especially careful if this contract is planned to be used by other contracts (i.e. library contracts, interactions). Selfdestruction of the callee contract can leave callers in an inoperable state.	`Line: 447 - 449`
	`function close() external onlyOwner {`
	`        selfdestruct(owner);`
	`    }`
		The function close is not called for any other reason, only with the reason to remove the owner, which leaves the contract without owner and can cause problems for current staker. Consider removing that function or modify that a new owner needs to be assigned before selfdestruct.

MEDIUM ISSUES

5.1.2 Wrong import of OpenZeppelin library

Severity: MEDIUM
Status: ADDRESSED
Update: Doesn’t affect the overall security in that use case too much
File(s) affected: layerx_v3.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation, OpenZeppelin files are added directly into the code. This violates OpenZeppelin’s MIT license, which requires the license and copyright notice to be included if its code is used. Moreover, updating code manually is error-prone.	`SafeMath, IERC20, Owned`	We highly recommend using npm (import "@openzeppelin/contracts/..) in order to guarantee that original OpenZeppelin contracts are used with no modifications. This also allows for any bug-fixes to be easily integrated into the codebase. https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.3.0/contracts/math/SafeMath.sol https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.3.0/contracts/token/ERC20/IERC20.sol https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.3.0/contracts/ownership/Ownable.sol

LOW ISSUES

5.1.3 Ownable OpenZeppelin library is not correctly implemented

Severity: LOW
Status: ADDRESSED
Update: Doesn’t affect the overall security in that use case too much
File(s) affected: layerx_v3.sol

Attack / Description	Code Snippet	Result/Recommendation

In the current implementation, OpenZeppelin files are added directly into the code. This
violates OpenZeppelin’s MIT license, which requires the license and copyright notice to be included if its code is used. Moreover, updating code manually is error-prone. In the case of Ownable, not all necessary functions are implemented.
	`Line: 74 – 89`
	`contract Owned`
		We highly recommend using npm (import "@openzeppelin/contracts/..) in order to guarantee that original OpenZeppelin contracts are used with no modifications. This also allows for any bug-fixes to be easily integrated into the codebase.
		Contract module which provides a basic access control mechanism, where there is an account (an owner) that can be granted exclusive access to specific functions.
		https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.3.0/contracts/ownership/Ownable.sol

5.1.4 uint vs. uint256 readability and consistency

Severity: LOW
Status: ADDRESSED
Update: Doesn’t affect the overall security in that use case too much
File(s) affected: layerx_v3.sol

Attack / Description	Code Snippet	Result/Recommendation

Since uint is just an alias to uint256 it is not necessary to use uint256. The best practice is a consistent use of uint data types, together with the specific size, and also because making the size of the data explicit reminds the developer and the reader how much data they've got to play with, which may help prevent or detect bugs and improve the maintainability and auditability of the codebase.	`Line: 114 – 130 and more`
	`struct Stake {`
	`        uint start;`
	`        uint end;`
	`        uint layerLockedTotal;`
	`        uint layerxReward;`
	`        uint ethReward;`
	`    }`
		Consider declaring the integer with the right size for struct and arrays. It brings about readability and consistency in your code, and it allows you to adhere to best practices in smart contracts.

INFORMATIONAL ISSUES

5.1.5 Fix Spelling and Grammatical Errors

Severity: INFORMATIONAL
Status: ADDRESSED
Update: Doesn’t affect the overall security in that use case too much
File(s) affected: layerx_v3.sol

Attack / Description	Code Snippet	Result/Recommendation

Language mistakes were identified in the messages in the codebase. Fixing these mistakes can help improve the end-user experience by providing clear information on errors encountered, and improve the maintainability and auditability of the codebase.	`Line: 410`
	`require((rwds.layerx > 0 || rwds.eth > 0), 'You have no any rewards to withdraw');`
		`There aren't any rewards to withdraw`

5.1.6 Pragma version not fixed

Severity: INFORMATIONAL
Status: ADDRESSED
Update: Doesn’t affect the overall security in that use case too much
File(s) affected: layerx_v3.sol

Attack / Description	Code Snippet	Result/Recommendation

Due to the fact that compiler upgrades might bring unexpected compatibility or inter-version consistencies, it is always suggested to use fixed compiler versions whenever possible.	`Line: 1`
	`pragma solidity ^0.5.0;`
		As an example, we highly encourage to explicitly indicate the Solidity compiler version, e.g., pragma solidity 0.5.0; instead of pragma solidity ^0.5.0;.

5.2. SWC Attacks & Special Checks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	✅
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✅
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✅
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✅
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	✅
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✅
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✅
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✅

ID	Title	Relationships	Test Result
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	✅
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	✅
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	✅
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✅
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✅
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	✅
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✅

ID	Title	Relationships	Test Result
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	✅
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✅
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	✅
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	✅
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	✅
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	✅
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	✅
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	❌
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	✅
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✅

ID	Title	Relationships	Test Result
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	✗
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	✅
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	✅
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	✅

7. Executive Summary

The overall code quality of the project is good, not overloaded with unnecessary functions, these is greatly benefiting the security of the contract. It has not correctly implemented widely-used and reviewed contracts from OpenZeppelin and for safe mathematical operations and we recommend as well, to specific all integer types.

The main goal of the audit was to verify the claims regarding the security of the smart contract and the functions. During the audit, no critical issues were found after the manual and automated security testing, all other issues are outlined in the audit section 5.

8. Deployed Smart Contract

VERIFIED

Layerx_v3
https://etherscan.io/address/0x472e536c60dffa345513e80a1e09a8f95cdf79f3#code