



**UniLayer**

**LAYERx v4**

**SMART CONTRACT AUDIT**

**11.02.2021**

**Made in Germany by Chainsulting.de**



## Table of contents

1. Disclaimer.....	4
2. About the Project and Company .....	5
2.1 Project Overview.....	6
3. Vulnerability & Risk Level .....	6
4. Auditing Strategy and Techniques Applied.....	8
4.1 Methodology .....	8
4.2 Used Code from other Frameworks/Smart Contracts .....	9
4.3 Tested Contract Files .....	10
4.4 Metrics / CallGraph.....	11
4.5 Metrics / Source Lines .....	12
4.6 Metrics / Capabilities .....	13
4.7 Metrics / Source Unites in Scope .....	13
5. Scope of Work .....	14
5.1 Manual and Automated Vulnerability Test.....	15
5.1.1 Selfdestruct owner .....	15
5.1.2 Wrong import of OpenZeppelin library .....	16
5.1.3 Ownable OpenZeppelin library is not correctly implemented .....	17
5.1.4 uint vs. uint256 readability and consistency.....	18
5.1.5 Fix Spelling and Grammatical Errors .....	19
5.1.6 Pragma version not fixed .....	20
5.2. SWC Attacks & Special Checks.....	21
5.3. Testnet Deployment .....	25



5.3.1 Deployment LAYER token contract.....	25
5.3.2 Deployment LAYERx contract.....	26
5.3.3 Default contract variables .....	27
5.3.4 Reducing staking period for testing .....	28
5.3.5 Approve LAYERx contract to spend token.....	28
5.3.6 Lock tokens for staking.....	29
5.3.7 Approve LAYER token contract for staking.....	30
5.3.8 Lock tokens for staking.....	30
5.3.9 Unlock staked tokens from pool.....	31
5.3.10 Check stake count variable .....	31
5.3.11 Finalize the staking by closing without ether rewards .....	32
5.3.12 Send Ether to contract.....	33
5.3.13 Finalize staking with ether rewards .....	33
5.3.14 Withdraw LAYERx tokens and ether.....	34
5.3.15 Total stakes count.....	34
6. Executive Summary.....	35
7. Deployed Smart Contract .....	36

## 1. Disclaimer

The audit makes no statements or warranties 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
1.5 (11.02.2021)	Re-check of contract

## 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\\_](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 not effecting any of the code.	An observation that does not determine a level of risk

## 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>

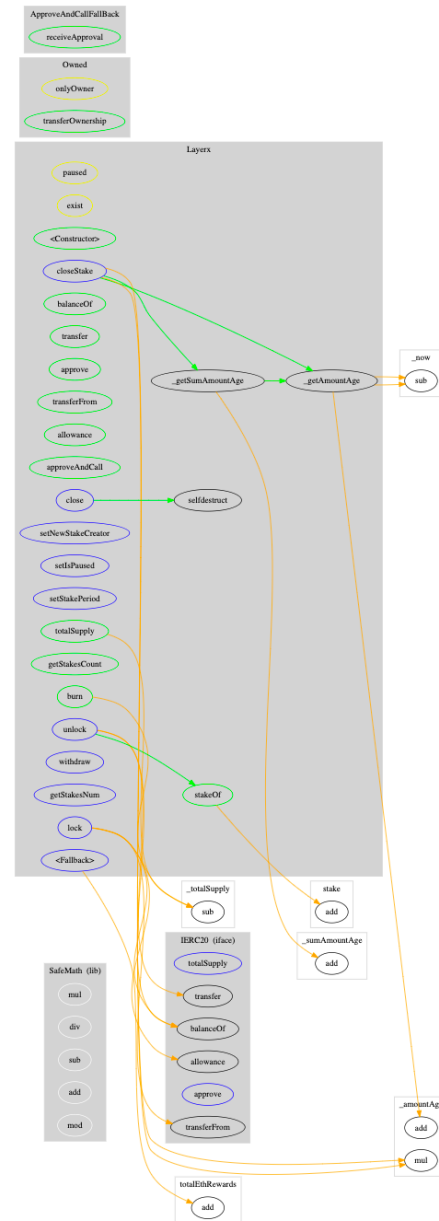


## 4.3 Tested Contract Files

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

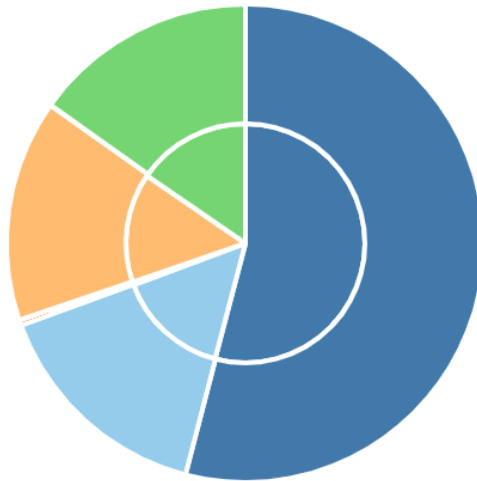
File	Fingerprint (MD5)
LAYERx_v4.sol	6e3653fe901e4e12c7f8d3bf8221e04b

## 4.4 Metrics / CallGraph



## 4.5 Metrics / Source Lines





source comment single block mixed  
empty todo blockEmpty



## 4.6 Metrics / Capabilities

Solidity Versions observed		 Experimental Features		 Can Receive Funds		 Uses Assembly		 Has Destroyable Contracts			
<code>^0.5.0</code>				<code>yes</code>		**** (0 asm blocks)		<code>yes</code>			
 Transfers ETH		 Low-Level Calls		 DelegateCall		 Uses Hash Functions		 ECRrecover		 New/Create/Create2	
<code>yes</code>											

## 4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	Layerx_v4.sol	4	1	453	436	279	82	224	
	<b>Totals</b>	<b>4</b>	<b>1</b>	<b>453</b>	<b>436</b>	<b>279</b>	<b>82</b>	<b>224</b>	

## 5. Scope of Work

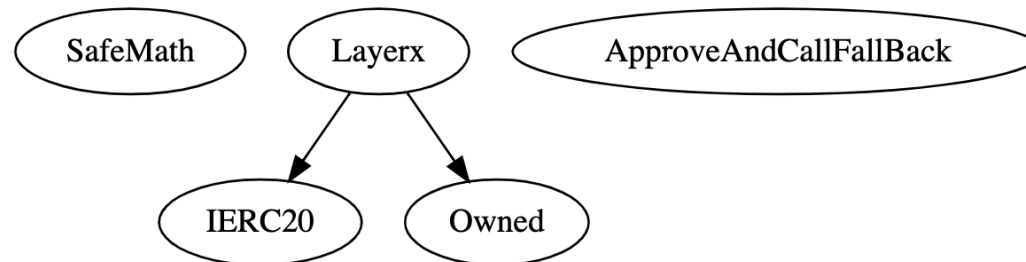
The Unilayer team provided us with the files that needs to be tested. The scope of the audit is the LayerX\_v3 contract.

Following contracts with the direct imports been tested  
LAYERx\_v4.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

File(s) affected: layerx\_v4.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 <pre>function close() external onlyOwner {     selfdestruct(owner); }</pre>	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

File(s) affected: layerx\_v4.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	<p>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.</p> <p><a href="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/math/SafeMath.sol</a></p> <p><a href="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/token/ERC20/IERC20.sol</a></p> <p><a href="https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.3.0/contracts/ownership/Ownable.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.3.0/contracts/ownership/Ownable.sol</a></p>



## 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

File(s) affected: layerx\_v4.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 <code>contract</code> 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.  <a href="https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.3.0/contracts/ownership/Ownable.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.3.0/contracts/ownership/Ownable.sol</a>

#### 5.1.4 uint vs. uint256 readability and consistency

Severity: LOW

Status: **ADDRESSED**

Update: Doesn't affect the overall security in that use case

File(s) affected: layerx\_v4.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 <pre>struct Stake {     uint start;     uint end;     uint layerLockedTotal;     uint layerxReward;     uint ethReward; }</pre>	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

File(s) affected: layerx\_v4.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 <pre>require((rwds.layerx &gt; 0    rwds.eth &gt; 0), 'You have no any rewards to withdraw');</pre>	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

File(s) affected: layerx\_v4.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 <code>pragma solidity ^0.5.0;</code>	As an example, we highly encourage to explicitly indicate the Solidity compiler version, e.g., <code>pragma solidity 0.5.0;</code> instead of <code>pragma solidity ^0.5.0;</code> .

## 5.2. SWC Attacks & Special Checks

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

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

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

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



## 5.3. Testnet Deployment

### 5.3.1 Deployment LAYER token contract

**CONTRACT**

Unilayer - browser/contracts/1\_Layer.sol

**DEPLOY**


NAME: Unilayer

SYMBOL: LAYER

DECIMALS: 18

TOTALSUPPLY: 40000000

OWNER: 0x2F1602FD37228b32Ad8D13137b1B620862771909

 **transact**

Tx: <https://ropsten.etherscan.io/tx/0x73f8f3f3b335ef68b35cd5332f372c5a147dbe48082ead5db24221fcb4854cfe>

Layer Token Contract address: [0x21bb8a338b68fe0e86eabccadc65e97a91520ef57](#)

### 5.3.2 Deployment LAYERx contract

CONTRACT

Layerx - browser/contracts/2\_Layerx.sol

DEPLOY

\_OWNER: 0x2F1602FD37228b32Ad8D13137b1B620862771909





LAYER\_TOKEN: 0x21bb8a338b68fe0e86eabcad65e97a91520ef57

transact

Tx: <https://ropsten.etherscan.io/tx/0x322d57158d509eb1301d9fc288e38e47c7a47c8d0abbaa05f984ae58a15a6fb8>

Layerx Contract Address: [0x5da2bbc48aa50421bca0a7d164c15c97ca4e1b56](https://ropsten.etherscan.io/address/0x5da2bbc48aa50421bca0a7d164c15c97ca4e1b56)


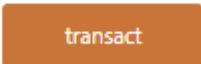
### 5.3.3 Default contract variables


Address	 0x5Da2bbc48aa50421bcA0a7D164c15C97cA4E1B56
ETH Balance	0
UNILAYER	 0x21bB8A338b68fE0E86eabcADc65E97a91520ef57
decimals	18
isPaused	X False
name	UNILAYERX
owner	 0x2F1602FD37228b32Ad8D13137b1B620862771909
stakeCreator	 0x2F1602FD37228b32Ad8D13137b1B620862771909
stakePeriod	604800
symbol	LAYERX
totalEthRewards	0
totalSupply	4000000000000000000000
getStakesNum	1

### 5.3.4 Reducing staking period for testing

**setStakePeriod**

newStakePeriod:



0: uint256: 30


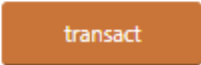
Tx: <https://ropsten.etherscan.io/tx/0x54059eaf8e4ac55a601488647a1536ec27c8700c0b66708f693e0b33c3f59b4f>

### 5.3.5 Approve LAYERx contract to spend token

**approve**

spender:

tokens:



 

Tx: <https://ropsten.etherscan.io/tx/0x4e54c241f10e69b364df8a4cb2e34db30a5cf546a204d9d6f151aaa5d092d83b>

**allowance**

tokenOwner:

spender:

0: uint256: remaining 2000000000000000000

### 5.3.6 Lock tokens for staking

lock

100000000000000000000



transact

Gas estimation failed

[illegible]

Send Transaction

Cancel Transaction



### 5.3.7 Approve LAYER token contract for staking

approve

spender: 0x5da2bbc48aa50421bca0a7d164c15c97ca4e1b56

value: 3000000000000000000



transact

Tx: <https://ropsten.etherscan.io/tx/0xe84906b5fef5231c9f2430355bb5e7944e9bf96a45bf6437c56386ceceb673f9>

### 5.3.8 Lock tokens for staking

lock

amount: 1000000000000000000



transact

Tx: <https://ropsten.etherscan.io/tx/0x0902e976ccaeab98ff03a4d945f6f8340667ae471ad5e52395f07cdc14c799e9>

stakeOf

account: "0x2F1602FD37228b32Ad8D13137b1B620862771909"



call

0: uint256: 1000000000000000000

### 5.3.9 Unlock staked tokens from pool

stakeOf

account: "0x2F1602FD37228b32Ad8D13137b18620862771909"



call

0: uint256: 0

Tx: <https://ropsten.etherscan.io/tx/0x13ccd9e863739478a4e7b529bde4cddc651187ed30381e0725c343a326b261b2>

### 5.3.10 Check stake count variable

getStakesCount

holder: "0x2F1602FD37228b32Ad8D13137b18620862771909"



call

0: uint256: 0

lock

amount: 1000000000000000000



transact

Tx: <https://ropsten.etherscan.io/tx/0x475d2ba349c72a866379ab74768c3d9982764d32f4f453cb9259eb37efb12759>

getStakesCount

holder: "0x2F1602FD37228b32Ad8D13137b1B620862771909"



call

0: uint256: 1

stakeOf

account: "0x2F1602FD37228b32Ad8D13137b1B620862771909"



call

0: uint256: 1000000000000000000

### 5.3.11 Finalize the staking by closing without ether rewards

rewards

0x2F1602FD37228b32Ad8D13137b1B620862771909

0: uint256: layerx 836187214612708332

1: uint256: eth 0

Tx: <https://ropsten.etherscan.io/tx/0xaa1040ded46d95cba6f70c295d717bd47c0f8e1281da373770265ab495993a64>





### 5.3.12 Send Ether to contract

Sending 0.1 Ether to the Layerx smart contract for staking rewards.

totalEthRewards	
0:	uint256: 1000000000000000000

Tx: <https://ropsten.etherscan.io/tx/0x86b08161d7384f9e45dcfe7a4d8d9b413e1ab8e1e956b9be5210808b118a3306>

### 5.3.13 Finalize staking with ether rewards

rewards	0x2F1602FD37228b32Ad8D13137b18620862771909
0:	uint256: layerx 1268391679351851849
1:	uint256: eth 999999999999999999
totalEthRewards	
0:	uint256: 0

Tx: <https://ropsten.etherscan.io/tx/0x6407d2cdcf55d394fdf1081076e47bd89db95145e484e025a491c13fa2153a91>

#### 5.3.14 Withdraw LAYERx tokens and ether



0.4516 ETH



39999999.9 LAYER



Tx: <https://ropsten.etherscan.io/tx/0x2a020f1a09d7a962fee697fc86104d855bc0087b6a6e91d73b45faa133f7c139>



0.5515 ETH



40000000 LAYER



#### 5.3.15 Total stakes count

getStakesNum

0: uint256: 3

## 6. 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.

## 7. Deployed Smart Contract

PENDING

LAYERx\_v4

<https://etherscan.io/0x>

