QuillAudits

AUDIT REPORT

March, 2025

For

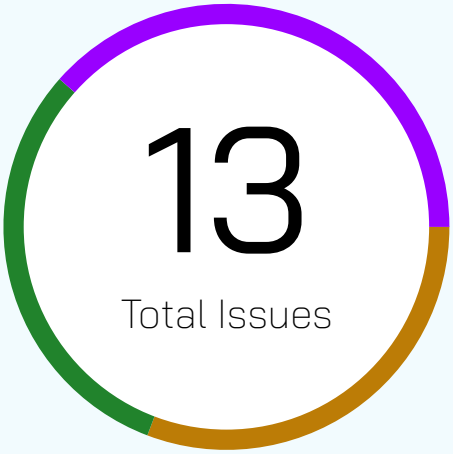
Table of Content

Executive Summary

Project name	Nex Labs
Project URL	https://www.nexlabs.io/
Overview	Nex Labs Stock Index Smart Contracts provide an on-chain infrastructure for managing tokenized stock index portfolios. At its core, the IndexFactory facilitates buying and selling underlying assets, minting and burning IndexTokens that represent a holder's share in the portfolio. The OrderManager creates buy/sell orders, routing them through the Dinari Order Processor, with final execution handled by the IndexFactoryProcessor where the ThirdWeb Engine watches for events on when orders are filled and afterwards invokes complete request functions. FactoryBalancer exists for the purpose of reweighting and rebalancing of assets, leveraging price data from FactoryStorage, which communicates with oracles. The Vault securely holds underlying assets, ensuring transparency and liquidity. Ownership and governance are managed through ProposableOwnable, enabling decentralized control.
Audit Scope	The scope of this Audit was to analyze the Nex Labs Smart Contracts for quality, security, and correctness.
Source Code	https://github.com/nexlabs22/Nex-Stock-Index-Contracts/tree/main/contracts
Commit Hash	Branch: Main 0c8d723185d2bd3070f7d699ebdb78f8f1f30611
Language	Solidity

Contracts in Scope	chainlink/ConfirmedOwner.sol chainlink/ConfirmedOwnerWithProposal.sol chainlink/FunctionsClient.sol coa/ContractOwnedAccounts.sol factory/IndexFactory.sol factory/IndexFactoryBalancer.sol factory/IndexFactoryProcessor.sol factory/IndexFactoryStorage.sol factory/OrderManager.sol proposable/ProposableOwnable.sol proposable/ProposableOwnableUpgradeable.sol vault/NexVault.sol
Blockchain	Arbitrum
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	14th January 2025 - 30th January 2025
Review 2	March 12 2025 to March 20 2025
Fixed In	Branch: Audit Change 2149a6e0c6f0f289ea8213f1feae2830bc4db4b0

Number of Issues per Severity



High	0 (0.00%)
Medium	4 (30.77%)
Low	4 (30.77%)
Informational	5 (38.46%)

Severity

Issues

	High	Medium	Low	Informational
Open	0	0	0	0
Resolved	0	**4**	**4**	**4**
Acknowledged	0	0	0	0
Partially Resolved	0	0	0	**1**

Checked Vulnerabilities

- ☑ Access Management
- ☑ Arbitrary write to storage
- ☑ Centralization of control
- ☑ Ether theft
- ☑ Improper or missing events
- ☑ Logical issues and flaws
- ☑ Arithmetic Computations Correctness
- ☑ Race conditions/front running
- ☑ SWC Registry
- ☑ Re-entrancy
- ☑ Timestamp Dependence
- ☑ Gas Limit and Loops
- ☑ Exception Disorder
- ☑ Gasless Send
- ☑ Use of tx.origin
- ☑ Malicious libraries
- ☑ Compiler version not fixed
- ☑ Address hardcoded
- ☑ Divide before multiply
- ☑ Integer overflow/underflow
- ☑ ERC's conformance
- ☑ Dangerous strict equalities
- ☑ Tautology or contradiction
- ☑ Return values of low-level calls
- ☑ Missing Zero Address Validation
- ☑ Private modifier
- ☑ Revert/require functions
- ☑ Multiple Sends
- ☑ Using suicide
- ☑ Using delegatecall
- ☑ Upgradeable safety
- ☑ Using throw

- [x] Using inline assembly
- [x] Unsafe type inference
- [x] Style guide violation
- [x] Implicit visibility level.

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools And Platforms Used For Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistical analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational Issues

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

Security vulnerabilities identified that must be resolved and are currently unresolved.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/ impact of the security issue, but are not completely resolved.

Medium Severity Issues

Validate that incoming Chainlink price data is not stale

Resolved

Description

The Index smart contract integrates Chainlink oracle for price derivation. However, when it fetches the price of a token, it fails to check that the price value is not stale as it is possible for an attacker to manipulate this.

```
    * @dev Returns the price in Wei.
    * @return The price in Wei.
    */
    ftrace | funcSig
    function priceInWei() public view returns (uint256) {
        (, int price, , , ) = toUsdPriceFeed.latestRoundData();
        uint8 priceFeedDecimals = toUsdPriceFeed.decimals();
        price = _toWei(price, priceFeedDecimals, 18);
        return uint256(price);
    }
```

Recommendation

Add check for stale price

```
- (, int price, , , ) = toUsdPriceFeed.latestRoundData();
+   (uint80 _roundId, int256 price, , uint256 _updatedAt, ) = toUsdPriceFeed.
latestRoundData();
+   if(_roundId == 0) revert InvalidRoundId();
+   if(price == 0) revert InvalidPrice();
+   if(_updatedAt == 0 || _updatedAt > block.timestamp) revert InvalidUpdate();
+   if(block.timestamp - _updatedAt > TIMEOUT) revert StalePrice();
```

Missing whenNotPaused Modifier on critical entry points

Resolved

Path

IndexFactory.sol#L206

Description

These contracts implement the pausable functionality that prevents users from calling important entry points in time of emergency. In the IndexFactory, the internal functions (_pause and _unpause) of the Pausable contract were appropriately invoked and the external functions had onlyOwner modifier. However, the implementation is incomplete as the "whenNotPaused" modifier was not attached to any critical functions. This implies that in time of emergency when the admin calls the pause function, the activity of the contract does not exactly pause. Users will proceed with their activity. Though there are no external/public functions present in other contracts where the PausableUpgradable contracts were inherited, the internal functions were not even called at all. This will form the bytecode of the contracts but never to be used. The contracts are:

- OrderManager
- IndexFactoryProcessor

Recommendation

Add the whenNotPaused modifier to critical functions that are meant to halt when the protocol faces an emergency situation. If not required for contracts where it was inherited but unimplemented, remove.

Missing 'disableInitializers` in Proxy Upgradable Contract Constructors

Resolved

Path

IndexFactory.sol#L206

Description

The failure to disable initializers at the constructor level of the proxy upgradeable contract raises concern as it is susceptible to exploit. This oversight presents the potential of attackers initializing the implementation contract itself.

Recommendation

Invoke the _disableInitializers() at the constructor.

```
5
6   // @custom:oz-upgrades-unsafe-allow constructor
7   constructor() {
8       _disableInitializers();
9   }
```

POC

Invoke the _disableInitializers() at the constructor.

Reference

https://forum.openzeppelin.com/t/what-does-disableinitializers-function-mean/28730

Owner can mint tokens above the supply ceiling as this check does not exist in mintToFeeReceiver()

Resolved

Path

token/IndexToken.sol

Function

mintToFeeReceiver()

Description

The mint function of the IndexToken contract has 5 require statements to check for address validity and the supply cap being less or equal to the supplyCeiling, however mintToFeeReceiver() function does not have a similar check for token supply minted. This would make the owner able to mint new tokens above the supplyCeiling set and also cause a DOS the next time an address with MINTER role calls mint() because totalSupply() + amount would already have exceeded the supplyCeiling.
require(totalSupply() + amount <= supplyCeiling, "will exceed supply ceiling");

If the owner address gets hijacked at any point, the compromised account would be able to mint new tokens without any restrictions leading to economic loss for legitimate users of the protocol.

Recommendation

Include the same require check in the mintToFeeReceiver function.

Low Severity Issues

setFeeRate requires more time than 12 hours to reset fee rate

Resolved

Path

IndexfactoryStorage.sol#L4

Function

setFeeRate()

Description

The setFeeRate is conditioned to be called after 12 hours from the last time of update. After 12 hours and 59 minutes, it's impossible to reset the fee rate until 13 hours.

Recommendation

Use >= notation rather than just > in order to reopen the ability to reset fee at the 12th hours.

Initialize does not invoke the ReentrancyGuard_init()

Resolved

Path

factory/OrderManagerL#61
factory/IndexFactoryProcessorL#732
factory/IndexFactoryProcessorL#732

Function

initialize()

Description

It is expected that initialize functions get invoked before normal contract interactions begin to ensure the contract state is as expected. The initialize functions in the contracts listed above do not call ReentrancyGuard's init function and could lead to potential inconsistencies in state if the codeblock contains instructions to set up the contract state.

Recommendation

Properly initialize all libraries and contracts to avoid inconsistencies in the contract state.

Use safeTransferLib for safety transfer of tokens

Resolved

Path

factory/IndexFactory.sol

Function

transferFrom()

Description

With the use of safeTransferLib, there is assurance of safety transfer of tokens.

Recommendation

Use safeTransferLib and replace transferFrom() with safeTransferFrom().

Owner is allowed to mint tokens to the feeReceiver even when the contract is paused

Resolved

Path

token/IndexToken.sol

Function

mintToFeeReceiver()

Description

The functionality of the pausable contract does not apply to the mintToFeeReceiver function and allows for the owner to continue token minting to the fee receiver while the contract is paused.

Recommendation

Add whenNotPaused to the mintToFeeReceiver().

Informational Severity Issues

Use the DRY principle to minimize the repetition of code

Resolved

Path

factory/OrderManager.sol

Function

calculateFees

Description

There's a function called calculateFees where it gets the flat fee and percentage fee rate and then accumulates the overall fees. Similar implementation is present in the requestBuyOrder and requestBuyOrderFromCurrentBalance. Rather than replicating these logics in both functions, call the calculateFees in its place to avoid code repetition.

Recommendation

Use the calculateFees function in requestBuyOrder and requestBuyOrderFromCurrentBalance.

Unused state variables and functions

Partially Resolved

Path

factory/IndexFactoryStorage.sol
factory/IndexFactoryBalancer.sol
factory/IndexFactoryProcessor.sol

Function

concatenation(),getTimestamp(), compareStrings(), isEmptyString()

Description

There are some defined state variables that were never used in the smart contracts.

-Unused ActionInfo struct in IndexFactoryProcessor
-Unused orderInstanceById mapping in IndexFactoryBalancer
-Unused getOrderInstanceById function in IndexFactoryBalancer
There were also a couple of functions that had onlyFactory modifiers but none of these functions were called across other contracts
-setRedemptionTokenPrimaryBalance
-setRedemptionIndexTokenPrimaryTotalSupply

Recommendation

Use unused state variables and functions.

Remove commented lines of code

Resolved

Path

factory/IndexFactoryStorage.sol
factory/IndexFactoryBalancer.sol
factory/IndexFactoryProcessor.sol

Function

concatenation(),getTimestamp(), compareStrings(), isEmptyString()

Description

There are a couple of commented lines of code across all contracts. Some were imports which were commented out, some were functions.

Recommendation

Remove commented lines of code not needed to keep code clean.

Array length mismatch in setWrappedDShareAddresses() and setPriceFeedAddress() can cause issues.

Resolved

Path

factory/IndexFactoryStorage.sol

Function

setWrappedDShareAddresses(), setPriceFeedAddress()

Description

In contract IndexFactoryStorage you can see that there are 2 functions, setWrappedDShareAddresses() and setPriceFeedAddresses(). In both they are checking dShares array with wrappedDShares and priceFeedAddress array.
Even though they are called by the owner and the owner is trusted and won't make a mistake still, say the owner called setWappedDShareAddresses with 10 addresses in both arrays and in setPriceFeedAddresses you just called with 9.
And for the last one if while buying/selling if the price feed is not available then it might fail/revert until they are set correctly again.

Also there are chances of messing the order of the priceFeed or wrappedDshares.

Recommendation

To resolve the issue make sure that every time both the functions are called they are called with all the required addresses correctly.

Minting tokens to the feeReceiver becomes very expensive if not done frequently

Resolved

Description

The current arithmetic implementation for minting tokens to the fee receiver is heavily gas intensive because it performs the arithmetic for compounding with a loop. This would become considerably expensive when the number of days grows to a large number.

With current tests, it costs ~500k gas units to call this function when 300 days have passed from the initial feeTimestamp value. Conversely, using a logarithmic approximation for compounding is just as precise with ~80% gas savings.

Functional Tests

Some of the tests performed are mentioned below:

- Should test for simultaneous creation of issuance and redemption request orders
- Test should fail for same order with requestBuyOrder() function
- Test should fail for fulfillBuyOrder when user is blacklisted before calling createOrderStandardFees()
- Test should fail for FulfillSellOrder when user is blacklisted after calling createOrderStandardFees()
- Test should fail for cancelOrder() when the user is blacklisted
- Logarithmic compounding is more efficient.

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of Nex Labs. We performed our audit according to the procedure described above.

Some issues of Medium, Low and informational severity were found. Some suggestions, gas optimizations and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem

QuillAudits

7+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

March, 2025

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com