



Neon Nexus

Interim Smart Contract Audit Report

DATE: 22 April 2024 | **PREPARED BY:** Entersoft Pte Ltd

Contents

Revision History & Version Control	3
1.0 Disclaimer	4
2.0 Overview	5
2.1 Project Overview	5
2.2 Scope	5
2.3 Audit Summary	6
2.4 Security Level References	6
2.5 Vulnerability Summary	6
3.0 Executive Summary	7
4.0 Technical Analysis	9
4.1 Centralization Risk for trusted owners	9
4.2 Optimization of Unused Public Functions in ChainlinkAggregator4Supra.sol Contract	9
4.3 Utilization of Constants for Numeric Values in ChainlinkAggregator4Supra.sol Contract	10
4.4 Optimizing Declaration of `sValueFeed` in AdapterSupra.sol Contract	10
4.5 Solidity Pragma Specificity Issue	10
4.6 Reentrancy	11
4.7 Integer Underflow and Overflow Vulnerabilities	11
4.8 Precision Loss Due To Rounding	11
4.9 Danger of Single Oracle Dependence So Usage Of Multiple Oracles	12
4.10 Front-Running	12
4.11 Unsafe Casting	12
5.0 Auditing Approach and Methodologies Applied	12
5.1 Code Review / Manual Analysis	12
5.2 Tools Used for Audit	13
6.0 Limitations on Disclosure and Use of this Report	14

Revision History & Version Control

Version	Date	Author(s)	Description
1.0	22 April 2024	Raj Dixit Gurkirat Singh	Interim report

Entersoft was commissioned by Neon Nexus (hereinafter referred to as "Neon Nexus") to perform a Security Audit on their smart contracts. The review was conducted from 3 April 2024 to 13 April 2024, to ensure overall code quality, security, and correctness, and ensure that the code will work as intended.

The report is structured into two main sections:

- Executive Summary which provides a high-level overview of the audit findings.
- Technical Analysis which offers a detailed analysis of the smart contract code.

Please note that the analysis is static and entirely limited to the smart contract code. The information provided in this report should be used to understand the security and quality of the code, as well as its expected behavior.

Standards followed include:

- OWASP (partially, for instance, role validations, input validations, etc.)
- Solidity Best Security Practices

1.0 Disclaimer

This is a limited audit report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to (i) smart contract best coding practices and issues in the framework and algorithms based on white paper, code, the details of which are set out in this report, (Smart Contract audit). To get a full view of our analysis, you must read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us based on what it says or does not say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Entersoft Australia and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Entersoft) owe no duty of care towards you or any other person, nor does Entersoft make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and Entersoft hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose, and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Entersoft hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Entersoft, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the Smart contract is purely based on the smart contract code shared with us alone.

2.0 Overview

2.1 Project Overview

Entersoft has meticulously audited the Neon Nexus smart contract project from 3rd April 2024 to 13rd April 2024, with a primary focus on Solidity code files integral to blockchain functionality, emphasizing vulnerabilities in associated gas claiming and Burn functionalities. The working of basic functionalities was also tested during the review.

2.2 Scope

The audit scope covers the Neon Nexus contracts available in the GitHub private repository. The audit focused on the checklist provided for the smart contract code audit.

Commits under scope: d2df076a86afb1dedc800b5325968238af4a31e2

Files in Examination:

- AdapterSupra.sol
- ChainlinkAggregator.sol
- IAdapterSupra.sol
- ISupraSValueFeed.sol

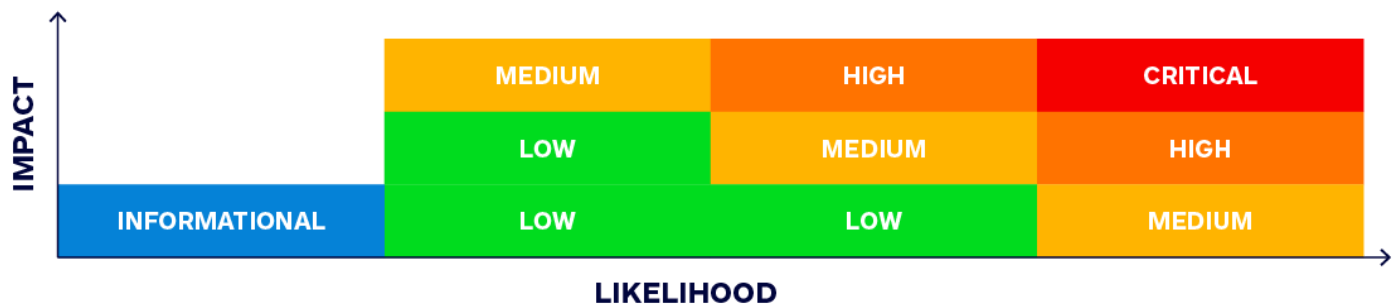
OUT-OF-SCOPE: External Solidity smart contract, other imported smart contracts.

2.3 Audit Summary

Delivery Date	Method of Audit	Team Members Engaged
22 April 2024	<p>Static Analysis: The static part of a smart contract audit refers to the process of reviewing the code of a smart contract to identify security vulnerabilities, coding errors, and adherence to best practices without executing the code and without interacting with it in a live environment. The smart contract is reviewed as is. In this we will cover logic vulnerabilities, dependency security flaws and more.</p> <p>Dynamic Analysis: The Dynamic audit of a smart contract involves analysing and testing the contract's code by executing it in a controlled environment to identify vulnerabilities, anomalies, and other security or quality issues that might not be evident through static analysis alone. Dynamic analysis involves interacting with the contract to observe its behavior and response to various inputs and conditions (we write both negative and positive test cases to test edge scenarios). This approach is crucial for uncovering security flaws that only occur during execution. This can include runtime operations, contract logic under specific conditions, and interactions with other contracts or external calls, etc.</p>	2

2.4 Security Level References

Every vulnerability in this report was assigned a severity level from the following classification table:



2.5 Vulnerability Summary

● Critical	● High	● Medium	● Low	● Informational
0	1	0	3	0

3.0 Executive Summary

Entersoft has conducted a comprehensive technical audit of the Neon Nexus smart contract through a comprehensive smart contract audit approach. The primary objective was to identify potential vulnerabilities and security risks within the codebase, ensuring adherence to industry-leading standards while prioritizing security, reliability, and performance. Our focus was on prompt and efficient identification and resolution of vulnerabilities to enhance the overall robustness of the solidity smart contract.

Testing Methodology:

We have leveraged static analysis techniques extensively to identify potential vulnerabilities automatically with the aid of cutting-edge tools such as Myhtril, Solhint, and Solium. Apart from this, we carried out extensive manual testing to iron out vulnerabilities that could slip through an automated check. This included a variety of attack vectors like reentrancy attacks, overflow and underflow attacks, timestamp dependency attacks, and more.

While going through the due course of this audit, we also ensured to cover edge cases, and built a combination of scenarios to assess the contracts' resilience. Our attempt to leave no stone unturned involved coming up with both negative and positive test cases for the system, and grace handling of stressed scenarios.

Our testing methodology in Solidity adhered to industry standards and best practices, integrating partially implemented OWASP and NIST SP 800 standards for encryption and signatures. Solidity's renowned security practices were complemented by tools such as Solhint for linting, and the Solidity compiler for code optimization. Sol-profiler, Sol-coverage, and Sol-sec were employed to ensure code readability and eliminate unnecessary dependencies.


Findings and Security Posture:













Our primary focus was on Access Control Policies, Reentrancy, Default Visibility, Outdated Compiler Version, Input Validation, DoS possibilities, Shadowing State Variables, Presence of Unused Variables, Overflow and Underflow Conditions, Assets Integrity, Errors and Exceptions, Re-initialization with Cross-instance Confusion, and Casting Truncation.

Importantly, our audit process intentionally avoided reliance solely on automated tools, emphasizing a more in-depth and nuanced approach to security analysis. Conducted from April 3, 2024, to April 13, 2024, our team diligently assessed and validated the security posture of the solidity smart contract, ultimately finding a number of vulnerabilities as per vulnerability summary table.

Result

The following table provides an overall summary of the findings and security posture of the smart contract code in scope.

-  **No Security vulnerabilities were identified**
-  **Security vulnerabilities were identified**

#	Neon Nexus Smart Contract Audit Attack Vectors	Result
1	Access Control Policies	
2	Transaction Signature Validations	
3	Default Visibility	
4	Address Validation	
5	Reentrancy attacks	
6	Presence of unused variables	
7	Overflow and Underflow Conditions	
8	Assets Integrity	
9	Gas claiming related issues	
10	Program Validation	
11	DoS	
12	Time Manipulation Vulnerability	
Overall Security Posture		NOT SECURE

4.0 Technical Analysis

The following results are the efforts of Static analysis and Dynamic analysis.

Note: The following values for “Result” mean:

- **PASS** indicates that there is no security risk.
- **FAIL** indicates that there is a security risk that needs to be remediated.
- **Informational** findings should be followed as a best practice, and they are not visible from the solidity smart contract.
- **Not Applicable** means the attack vector is Not applicable or Not available

4.1 Centralization Risk for trusted owners

Result	Fail
Severity	High
Description	In the ChainlinkAggregator4Supra.sol contract, centralization risks stemming from reliance on trusted owners are observed across various lines of code.
Location / Source File	Found in ChainlinkAggregator4Supra.sol Line: 8, 29, 33, 38
Observation	The ChainlinkAggregator4Supra contract inherits from Owned and implements AggregatorV2V3Interface. Functions setEnableAdapter, setAdapterInfo, and setLatestAnswer are defined as public and restricted to be called only by the contract owner.
Remediation	Consider decentralizing control mechanisms by implementing access control lists or permissioned functions to mitigate centralization risks associated with trusted owners.
Remark	Relying solely on the contract owner for critical functions may pose risks of centralization, which could lead to potential vulnerabilities and single points of failure. Decentralizing control can enhance security and resilience within the system.

4.2 Optimization of Unused Public Functions in ChainlinkAggregator4Supra.sol Contract

Result	Fail
Severity	Low
Description	Functions not used internally could be marked external
Location / Source File	Found in ChainlinkAggregator4Supra.sol Line: 29, 33, 38, 59, 63, 144, 163, 195
Observation	Several functions are declared as public but are not utilized internally within the contract.
Remediation	Marking functions as external when they are not used internally within the contract can improve code readability and reduce gas costs by preventing unnecessary state changes.

Remark	Marking functions as external when they are not used internally within the contract can improve code readability and reduce gas costs by preventing unnecessary state changes.
---------------	---

4.3 Utilization of Constants for Numeric Values in ChainlinkAggregator4Supra.sol Contract

Result	Fail
Severity	Low
Description	Constants should be defined and used instead of literals
Location / Source File	Found in ChainlinkAggregator4Supra.sol Line: 106, 117, 176
Observation	Numeric literals such as 0xFFFFFFFF are used directly in the code.
Remediation	Define constants for these numeric values to improve code readability, maintainability, and to adhere to best practices.
Remark	Using constants instead of literals enhances code readability and makes it easier to update and maintain. It also helps prevent errors and inconsistencies in the codebase.

4.4 Optimizing Declaration of `sValueFeed` in AdapterSupra.sol Contract

Result	Fail
Severity	Low
Description	The issue concerns the declaration of the sValueFeed variable within the AdapterSupra contract, specifically observed at line 8 in the AdapterSupra.sol file.
Location / Source File	AdapterSupra.sol Line: 8
Observation	The sValueFeed variable is declared as a state variable.
Remediation	Change the state variable declaration of sValueFeed to be immutable if its value is intended to be set only once during contract deployment and remain constant thereafter.
Remark	Making sValueFeed immutable ensures that its value cannot be changed after contract deployment, which can enhance security and reduce the risk of unintended modifications.

4.5 Solidity Pragma Specificity Issue

Result	Fail
Severity	Low
Description	The pragma directive in the contracts specifies a wide range of Solidity versions using the caret (^) symbol. This can potentially introduce compatibility issues with future compiler versions, as it allows

	any version of Solidity greater than or equal to 0.8.0 in ChainlinkAggregator4Supra.sol and 0.8.13 in AdapterSupra.sol, including major updates that may introduce breaking changes.
Location / Source File	ChainlinkAggregator4Supra.sol, AdapterSupra.sol
Observation	The use of ^ in the pragma directive allows for flexibility but can lead to unexpected behavior if there are breaking changes in future compiler versions.
Remediation	Replace the pragma directive with a specific Solidity version, such as pragma solidity 0.8.20;

4.6 Reentrancy

Result	Pass
Description	Reentrancy attack is a kind of malicious action which allows an attacker to repeatedly call a function of a smart contract before the first function call has ended, potentially draining funds or causing other harm.
Location / Source File	ChainlinkAggregator4Supra.sol, AdapterSupra.sol

4.7 Integer Underflow and Overflow Vulnerabilities

Result	Pass
Description	These occur when an arithmetic operation reaches a value outside the range that can be represented within the fixed bit-size of the integer type, causing the value to wrap around the range.
Location / Source File	ChainlinkAggregator4Supra.sol, AdapterSupra.sol

4.8 Precision Loss Due To Rounding

Result	Pass
Description	This lack of precision can silently erode token values, skew distributive fairness, create systemic vulnerabilities, and even unlock doors to potential DDoS attacks.
Location / Source File	ChainlinkAggregator4Supra.sol, AdapterSupra.sol

4.9 Danger of Single Oracle Dependence So Usage Of Multiple Oracles

Result	Pass
Description	Relying exclusively on a single oracle source presents a considerable risk for any DeFi protocol. This dependency can expose the system to various threats, such as inaccurate data due to oracle manipulation, service disruptions in the event of oracle failures, or outdated information if the oracle data becomes stale.
Location / Source File	ChainlinkAggregator4Supra.sol, AdapterSupra.sol

4.10 Front-Running

Result	Pass
Description	Front running exploits the transparent and deterministic nature of transaction execution in Ethereum.
Location / Source File	ChainlinkAggregator4Supra.sol, AdapterSupra.sol

4.11 Unsafe Casting

Result	Pass
Description	Unsafe type casting vulnerabilities occur when the type conversion process encounters a value that doesn't fit within the bounds of the target type, which can lead to unexpected results.
Location / Source File	ChainlinkAggregator4Supra.sol, AdapterSupra.sol

5.0 Auditing Approach and Methodologies Applied

The solidity smart contract was audited in a comprehensive approach to ensure the highest level of security and reliability. Careful attention was given to the following key areas to ensure the overall quality of code:

- **Code quality and structure:** We conducted a detailed review of the codebase to identify any potential issues related to code structure, readability, and maintainability. This included analyzing the overall architecture of the solidity smart contract and reviewing the code to ensure it follows best practices and coding standards.

- **Security vulnerabilities:** Our team used manual techniques to identify any potential security vulnerabilities that could be exploited by attackers. This involved a thorough analysis of the code to identify any potential weaknesses, such as buffer overflows, injection vulnerabilities, Signatures, and deprecated functions.
- **Documentation and comments:** Our team reviewed the code documentation and comments to ensure they accurately describe the code's intended behavior and logic. This helps developers to better understand the codebase and make modifications without introducing new issues.
- **Compliance with best practices:** We checked that the code follows best practices and coding standards that are recommended by the solidity community and industry experts. This ensures that the solidity smart contract is secure, reliable, and efficient.

Our audit team followed OWASP and Ethereum(Solidity) community security guidelines for this audit. As a result, we were able to identify potential issues and provide recommendations to improve the smart contract security and performance. Throughout the audit of the smart contracts, our team placed great emphasis on ensuring the overall quality of the code and the use of industry best practices. We meticulously reviewed the codebase to ensure that it was thoroughly documented and that all comments and logic aligned with the intended behavior. Our approach to the audit was comprehensive, methodical, and aimed at ensuring that the smart contract was secure, reliable, and optimized for performance.

5.1 Code Review / Manual Analysis

Our team conducted a manual analysis of the Solidity smart contracts to identify new vulnerabilities or to verify vulnerabilities found during static and manual analysis. We carefully analyzed every line of code and made sure that all instructions provided during the onboarding phase were followed. Through our manual analysis, we were able to identify potential vulnerabilities that may have been missed by automated tools and ensure that the smart contract was secure and reliable.

Throughout this comprehensive audit process, we have adhered to a meticulous and well-structured plan that allowed us to efficiently deliver a deep analysis of the Neon Nexus smart contracts.

1. Oracle Mechanism Analysis

Our journey started with the Oracle Mechanism Analysis, where we delved deeper into understanding the logic that drives the integration of the specified oracle within the protocol. We held intensive discussions on how the oracle is leveraged to bring in external price data into the system. This phase provided us critical insights on the well-thought-out design of Oracle Mechanism that catered to Neon Nexus's business needs.

2. Review of Oracle Vulnerabilities

The next step involved a rigorous exploration of potential vulnerabilities within the oracle mechanism. This crucial stage ensured our awareness of common pitfalls, as well as more intricate security flaws associated with oracle designs. Given our rich experience and the wide spectrum of known and hypothetical threats, we closely examined every possibility.

3. Code Examination and Vulnerability Assessment

Our next crucial activity framed the most extensive and important part of our project. We carried out an exhaustive line-by-line audit of the source code of each solidity file under the audit scope:

- AdapterSupra.sol: We noted how the contract interfaced with external data sources and validated the interaction paradigm to ensure absolute data integrity.
- ChainlinkAggregator4Supra.sol: We critically evaluated its utilization of Chainlink's decentralized oracle network for price feeds to confirm reliability and real-time data inputs.
- IAdapterSupra.sol: Our in-depth dive into this interface offered valuable insights into how it defined the structure for the AdapterSupra contract integration.
- ISupraSValueFeed.sol: By methodically scrutinizing this interface, we thoroughly grasped how it structured Supra's value feed oracle's scheme.

During the code examination, attempted attacks were orchestrated from various perspectives, playing around with potential manipulation possibilities, aimed at exploiting the possible weak points. This allowed us to uncover any hidden vulnerabilities that could potentially harm the price data reliability.

5.2 Tools Used for Audit

In the course of our audit, we leveraged a suite of tools to bolster the security and performance of our program. While our team drew on their expertise and industry best practices, we also integrated various tools into our development environment. Noteworthy among them are Slither, ethlint, and testing using hardhat. This holistic approach ensures a thorough analysis, uncovering potential issues that automated tools alone might overlook. Entersoft takes pride in utilizing these tools, which significantly contribute to the quality, security, and maintainability of our codebase.

6.0 Limitations on Disclosure and Use of this Report

This report contains information concerning potential details of Neon Nexus Project and methods for exploiting them. Entersoft recommends that special precautions be taken to protect the confidentiality of both this document and the information contained herein. Security Assessment is an uncertain process, based on past experiences, currently available information, and known threats. All information security systems, which by their nature are dependent on human beings, are vulnerable to some degree. Therefore, while Entersoft considers the major security vulnerabilities of the analyzed systems to have been identified, there can be no assurance that any exercise of this nature will identify all possible vulnerabilities or propose exhaustive and operationally viable recommendations to mitigate those exposures. In addition, the analysis set forth herein is based on the technologies and known threats as of the date of this report. As technologies

and risks change over time, the vulnerabilities associated with the operation of the Neon Nexus solidity smart contract described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities will also change. Entersoft makes no undertaking to supplement or update this report based on changed circumstances or facts of which Entersoft becomes aware after the date hereof, absent a specific written agreement to perform the supplemental or updated analysis. This report may recommend that Entersoft use certain software or hardware products manufactured or maintained by other vendors. Entersoft bases these recommendations upon its prior experience with the capabilities of those products. Nonetheless, Entersoft does not and cannot warrant that a particular product will work as advertised by the vendor, nor that it will operate in the manner intended. This report was prepared by Entersoft for the exclusive benefit of Neon Nexus and is proprietary information. The Non-Disclosure Agreement (NDA) in effect between Entersoft and Neon Nexus governs the disclosure of this report to all other parties including product vendors and suppliers.