# QuillAudits

# AUDIT REPORT

---

September 2025

For

cyberPerp

# Table of Content

# Executive Summary

| | |
|---|---|
| **Project Name** | CyberPerp |
| **Protocol Type** | Prep Dex |
| **Project URL** | https://cyberperp.io/ |
| **Overview** | Cyberperp is a decentralized spot and perpetual exchange built on the Iota EVM, tailored for traders seeking low swap fees and minimal price impact on trades. Cyberperp inherits robust features and introduces specialized functionalities optimized for the Iota EVM ecosystem. This platform uniquely supports trading via a multi-asset liquidity pool, which not only facilitates trades but also rewards liquidity providers with fees from market making, swaps, and leveraged trading.

The protocol is live on IOTA EVM. |
| **Audit Scope** | The scope of this Audit was to analyze the CyberPerp Smart Contracts for quality, security, and correctness. |
| **Source Code link** | https://github.com/ttbbio/cyberperp_contracts/tree/main/contracts_v2 |
| **Contracts in Scope** | contracts/mock/MockPriceFeed.sol<br>contracts/oracle/ChainlinkPriceFeedUtils.sol<br>contracts/oracle/Oracle.sol<br>scripts/update/updatePricesToPriceFeed.ts<br>config/oracle.ts<br>== core contracts ==<br>DataStore.sol - main app storage<br>ExchangeRouter.sol - main ops router<br>MarketFactory.sol - new markets creation<br>MarketToken.sol - lp tokens for the markets<br>== handlers ==<br>DepositHandler.sol<br>WithdrawalHandler.sol<br>OrderHandler.sol<br>== managment ==<br>RoleStore.sol<br>Config.sol |

| | |
|---|---|
| **Language** | Solidity |
| **Blockchain** | Iota EVM |
| **Branch** | Main |
| **Commit Hash** | 3f02208bdbbdc00253426dbe9e0b308be03b90fc |
| **Method** | Manual Analysis, Functional Testing, Automated Testing |
| **Review 1** | 22nd August 2025 - 10th September 2025 |
| **Updated Code Recieved** | 22nd September 2025 |
| **Review 2** | 23rd September 2025 |
| **Note** | The CyberPerp codebase was migrated from the Quantara codebase. The source code for this report includes the fixes implemented within the code of Quantara, which is why the "Fixed in Commit Hash" is not provided. To clarify further, the issues identified in CyberPerp are similar to those in Quantara, as the initial Quantara codebase was used as the foundation for CyberPerp's audit. |

## Verify the Authenticity of Report on QuillAudits Leaderboard:

https://www.quillaudits.com/leaderboard

# Number of Issues per Severity

**3**

Total Issues
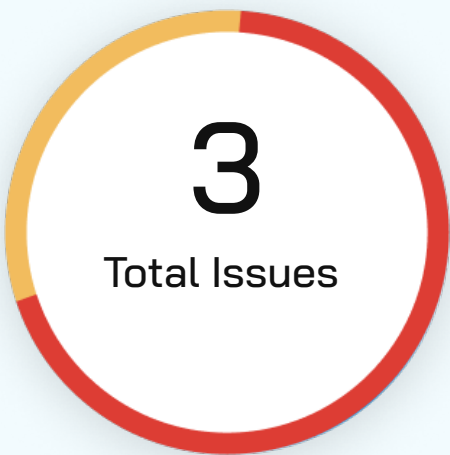
| | Severity | |
|---|---|---|
| ■ Critical | 0 (0.0%) | |
| ■ High | 2 (66.6%) | |
| ■ Medium | 0 (0.0%) | |
| ■ Low | 1 (33.3%) | |
| ■ Informational | 0 (0.0%) | |

Severity

| Issues | Critical | High | Medium | Low | Informational |
|---|---|---|---|---|---|
| Open | 0 | 0 | 0 | 0 | 0 |
| Acknowledged | 0 | 0 | 0 | 0 | 0 |
| Partially Resolved | 0 | 0 | 0 | 0 | 0 |
| Resolved | 0 | **2** | 0 | **1** | 0 |

# Summary of Issues

| Issue No. | Issue Title | Severity | Status |
|---|---|---|---|
| **1** | Missing execution fee capping mechanism enables unlimited fee extraction attacks | **High** | **Resolved** |
| **2** | Relaxed funding rate validation enables excessive market manipulation through funding parameters | **High** | **Resolved** |
| **3** | A missing spread reduction mechanism increases trading costs for users | **Low** | **Resolved** |

# 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**          ✅ **Upgradeable safety**

✅ **Private modifier**                         ✅ **Using throw**

✅ **Revert/require functions**                 ✅ **Using inline assembly**

✅ **Multiple Sends**                           ✅ **Style guide violation**

✅ **Using suicide**                            ✅ **Unsafe type inference**

✅ **Using delegatecall**                       ✅ **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 Static Analysis.

# Types of Severity

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

### ■ Critical: Immediate and Catastrophic Impact

Critical issues are the ones that an attacker could exploit with relative ease,  potentially leading to an immediate and complete loss of user funds, a total  takeover of the protocol's functionality, or other catastrophic failures. Critical vulnerabilities are non-negotiable; they absolutely must be fixed.

### ■ High (H): Significant Risk of Major Loss or Compromise

High-severity issues represent serious weaknesses that could result in significant financial losses for users, major  malfunctions within the protocol, or substantial compromise of its intended operations. While exploiting these vulnerabilities might require specific conditions to be met or a moderate level of technical skill, the potential damage is considerable. These findings are critical and should be addressed and resolved thoroughly before the contract is put into the Mainnet.

### ■ Medium (M): Potential for Moderate Harm Under Specific Circumstances

Medium-severity bugs are loopholes in the protocol that could lead to moderate financial losses or partial disruptions of the protocol's intended behavior. However, exploiting these vulnerabilities typically requires more specific and less common conditions to occur, and the overall impact is generally lower compared to high or critical issues. While not as immediately threatening, it's still highly recommended to address these findings to enhance the contract's robustness and prevent potential problems down the line.

### ■ Low (L): Minor Imperfections with Limited Repercussions

Low-severity issues are essentially minor imperfections in the smart contract that have a limited impact on user funds or the core functionality of the protocol. Exploiting these would usually require very specific and unlikely scenarios and would yield minimal gain for an attacker. While these findings don't pose an immediate threat, addressing them when feasible can contribute to a more polished and well-maintained codebase.

### ■ Informational (I): Opportunities for Improvement, Not Immediate Risks

Informational findings aren't security vulnerabilities in the traditional sense. Instead, they highlight areas related to the clarity and efficiency of the code, gas optimization, the quality of documentation, or adherence to best development practices. These findings don't represent any immediate risk to the security or functionality of the contract but offer valuable insights for improving its overall quality and maintainability. Addressing these is optional but often beneficial for long-term health and clarity.

# Types of Issues

**Open**

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

**Resolved**

These are the issues identified in the initial audit and have been successfully fixed.

**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.

# Severity Matrix

Impact

| | 🟥 High | 🟧 Medium | 🟨 Low |
|---|---|---|---|
| 🟥 **High** | Critical | High | Medium |
| 🟧 **Medium** | High | Medium | Low |
| 🟨 **Low** | Medium | Low | Low |

*Likelihood* (vertical axis label)

**Impact**

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.

- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.

- **Low** - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

**Likelihood**

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.

- Medium - only a conditionally incentivized attack vector, but still relatively likely.

- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

# Risks and Limitations

- **Limited Scope / Diff-Only Audit**
  This audit covered only the differences introduced between the original GMX codebase and this project. Only the in-scope files and code segments were reviewed. Large portions of the inherited or modified codebase, including several thousand lines of logic, were not audited. Vulnerabilities may exist outside the reviewed diffs. Exploits in non-audited code are outside the responsibility of this audit.

- **Inherited Vulnerabilities from GMX**
  Much of the codebase is inherited from GMX. Any vulnerabilities, misconfigurations, or design flaws present in GMX may persist in this project. This audit did not verify the correctness of the original GMX implementation.

- **Integration Risks**
  Interactions with external systems such as price oracles, cross-chain bridges, and third-party protocols were not audited. A compromise in any external dependency may affect the security and solvency of the protocol, even if the audited contracts are correct.

- **Operational and Deployment Risks**
  Deployment parameters, such as oracle addresses, fee settings, or collateral configurations, may introduce risks if set incorrectly. Operational security of private keys, multisigs, and keeper infrastructure was not reviewed.

- **Neura-Specific Risks**
  As of the time of this audit, the Neura blockchain is only available on testnet and the mainnet has not been publicly released. This introduces additional uncertainty, as the network has not undergone live mainnet.

# High Severity Issues

**Missing execution fee capping mechanism enables unlimited fee extraction attacks**

<span style="background:green;color:white">Resolved</span>

### Path

contracts/gas/GasUtils.sol

### Function

`validateExecutionFee`

### Description

In contracts/gas/GasUtils.sol:178-184, the validateExecutionFee function only validates that the provided execution fee meets the minimum threshold by calculating the gas limit and minimum fee, then reverting if the execution fee is insufficient. However, the original GMX V2 implementation includes a validateAndCapExecutionFee function that provides critical protection against execution fee abuse by enforcing a maximum execution fee cap based on gas limit, base fee, and a configurable multiplier factor. The absence of this maximum validation allows users to submit orders with arbitrarily high execution fees, which are later refunded through the callback mechanism or directly to the user, creating an opportunity for fund extraction. The missing MAX_EXECUTION_FEE_MULTIPLIER_FACTOR configuration parameter and associated validation logic remove this essential economic security control.

Consequently, malicious users can exploit the unlimited execution fee acceptance to extract funds from the protocol through the refund mechanism. An attacker can create orders with execution fees far exceeding the actual gas costs, such as 1000 ETH for a transaction that costs 0.01 ETH to execute, and receive the excess 999.99 ETH as a refund through callback contracts or direct transfers. This vulnerability is particularly severe for subaccount orders with callback contracts, where the excess execution fee refund is sent to the callback contract under the attacker's control. The cumulative effect of such attacks could systematically drain protocol funds and compromise the economic security of the entire system, as there are no upper bounds on the execution fees that users can specify or receive as refunds.

### Recommendation

We recommend implementing the missing validateAndCapExecutionFee function with proper maximum execution fee validation logic that caps execution fees using a configurable multiplier factor applied to the gas limit and base fee, and adding the MAX_EXECUTION_FEE_MULTIPLIER_FACTOR configuration parameter with appropriate validation ranges to prevent execution fee abuse attacks.

### Fixed In

8b2af585d8f771ae368904293145b033c24fcc95

## Relaxed funding rate validation enables excessive market manipulation through funding parameters

**Resolved**

### Path

contracts/config/Config.sol

### Function

`_validateRange`

### Description

In contracts/config/Config.sol:656-662, the _validateRange function groups FUNDING_INCREASE_FACTOR_PER_SECOND and FUNDING_DECREASE_FACTOR_PER_SECOND with other parameters under a single validation check that limits values to 1% of FLOAT_PRECISION (1 × 10^28). However, GMX V2 implements specific granular validation for these parameters with MAX_ALLOWED_FUNDING_INCREASE_FACTOR_PER_SECOND (approximately 2.777 × 10^19) and MAX_ALLOWED_FUNDING_DECREASE_FACTOR_PER_SECOND (approximately 1.157 × 10^18), representing 360,000-fold and 8.64 million-fold stricter limits, respectively.

However, these funding rate adjustment parameters directly control market stability by determining how quickly funding rates change in response to position imbalances. GMX V2's conservative limits ensure that funding rates reach maximum levels over approximately 1 hour and return to zero within 24 hours, providing a predictable progression of funding costs for traders.

Consequently, the relaxed validation allows configuration of extreme funding rate adjustment speeds that could destabilise market operations. Malicious configuration could cause funding rates to spike to maximum levels within seconds rather than hours, creating unpredictable funding costs that exceed position values and force immediate liquidations regardless of market conditions.

### Recommendation

We recommend implementing separate validation blocks for FUNDING_INCREASE_FACTOR_PER_SECOND and FUNDING_DECREASE_FACTOR_PER_SECOND, with conservative upper bounds similar to those in GMX V2. Specifically, this involves limiting the increase factor to prevent funding rates from reaching maximum levels in under one hour and limiting the decrease factor to ensure a gradual return to equilibrium.

### Fixed In

dd1c6add48b6399295f9d56a81bb303bea4fc325

# Low Severity Issues

## A missing spread reduction mechanism increases trading costs for users

**Resolved**

### Path

contracts/oracle/ChainlinkDataStreamProvider.sol

### Function Name
**getOraclePrice**

### Description

In contracts/oracle/ChainlinkDataStreamProvider.sol:77-78, the getOraclePrice function directly uses raw bid and ask prices from Chainlink Data Streams without any spread reduction mechanism. The code sets adjustedBidPrice to the raw bid price and adjustedAskPrice to the raw ask price, then returns these values as the minimum and maximum prices respectively in the ValidatedPrice struct. However, the original GMX V2 implementation includes a configurable spread reduction factor that narrows the bid-ask spread by moving the bid price upward and ask price downward toward the mid-price, effectively reducing trading costs for users. The absence of this mechanism means users pay the full spread from Chainlink Data Streams, which includes market impact pricing designed for institutional trading rather than optimized retail execution.

Consequently, users executing trades will experience higher costs compared to the original GMX V2 protocol. For a typical ETH trade with a $1.00 spread, users would pay an additional $0.25-$0.50 per trade depending on the configured spread reduction factor in GMX V2. This impact becomes more significant for large trades, where a $100,000 position would incur an additional $125-$250 in spread costs compared to GMX V2 with 50% spread reduction enabled. The cumulative effect across all trading activity represents a measurable increase in protocol costs that may impact user adoption and trading volume, particularly for price-sensitive traders who compare execution quality across different decentralized exchanges.

### Recommendation
We recommend implementing a configurable spread reduction mechanism similar to GMX V2 that allows narrowing the bid-ask spread through governance-controlled parameters, enabling the protocol to balance between optimal user pricing and oracle price integrity based on market conditions and competitive requirements.

### Fixed In

ea272831031aac4e33eb7b9a5038397729a198b8

# 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

This engagement provided a focused review of the differences introduced in this project compared to the original GMX codebase. The audit identified two high severity vulnerabilities within the scoped changes, but it must be emphasized that large portions of the codebase, including inherited GMX logic and other modifications, were not reviewed. As such, vulnerabilities may exist outside the audited scope.

Additionally, deployment on Neura introduces environment-specific risks. Neura is currently only available on testnet, with its mainnet not yet publicly released, meaning that many of its features remain unproven under production conditions. Factors such as instant finality, RPCfi dynamics, bridge dependencies, and ecosystem immaturity present uncertainties that should be carefully considered prior to deployment.

# 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 leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With seven years of expertise, we've secured over 1400 projects globally, averting over $3 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

QuillAudits

| 7+ | 1M+ |
|---|---|
| Years of Expertise | Lines of Code Audited |
| 50+ | 1400+ |
| Chains Supported | Projects Secured |

**Follow Our Journey**

# AUDIT REPORT

---

September 2025

For

cyberPerp

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com        audits@quillaudits.com