QuillAudits

AUDIT REPORT

July 2025

For

perpX

Table of Content

Executive Summary

Project Name	PerpX
Protocol Type	Perpetuals
Project URL	https://perpx.org/
Overview	Perpx is a perpetual futures trading protocol that allows users to trade leveraged positions on various tokens. The protocol centers around a Vault contract that manages liquidity pools, collateral, and position tracking, while the Perpetual contract provides the interface for opening/closing leveraged long/short positions. Users can deposit collateral and create positions with leverage through an order book system, with the protocol supporting both token-to-token swaps via the Router and ETH handling through WETH integration. The system includes comprehensive risk management features like liquidation mechanisms, funding rates, and fee structures. The architecture resembles GMX-style perpetual DEXs where traders get leveraged exposure while liquidity providers supply capital to a shared vault that acts as the counterparty to all trades.
Audit Scope	The scope of this Audit was to analyze the Womofi Smart Contracts for quality, security, and correctness
Source Code link	https://github.com/perpx-ai/perpx-smart-contract/tree/main/contracts
Contracts in Scope	Perpetual.sol, Router.sol
Branch	Main
Commit Hash	25c78add72d97332353980f926f7ee42d10d3214
Language	Solidity
Blockchain	EVM
Method	Manual Analysis, Functional Testing, Automated Testing

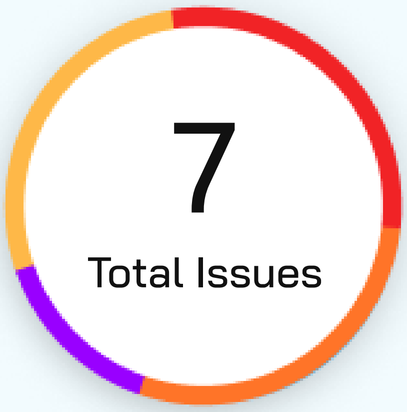
Review 1	13th June 2025 - 9th July 2025
Updated Code Received	14th July 2025
Review 2	14th July 2025 - 15th July 2025
Fixed In	https://github.com/perpx-ai/perpx-smart-contract/tree/master/contracts Commit hash: 98309fc21cca1e082848823704335e9ef26ef153

Notes: The Perpx codebase currently appears immature, lacking a clear purpose and proper integration with GMX. We strongly advise the Perpx team to prioritize thorough development over rushing to deployment. It's crucial to ensure the code is running perfectly and that test case coverage exceeds 90% before proceeding.

Verify the Authenticity of Report on QuillAudits Leaderboard:

https://www.quillaudits.com/leaderboard

Number of Issues per Severity



🟥	Critical	0 (0%)
🟥	High	2 (28.5%)
🟧	Medium	2 (28.5%)
🟨	Low	2 (28.5%)
🟪	Informational	1 (14.5%)

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	2	2	1

Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Incorrect Interface will lead to contract being non-operational	High	Resolved
2	Some critical functions won't work in the contract	High	Resolved
3	Funds might get locked in the contract	Medium	Resolved
4	Use safeTransferFrom instead of transferFrom	Medium	Resolved
5	GMX exchange router address might change	Low	Resolved
6	Increase position does not return position key	Low	Resolved
7	Remove redundant openPosition function.	Informational	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
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Multiple Sends
- ✓ Using suicide
- ✓ Using delegatecall
- ✓ Upgradeable safety
- ✓ Using throw
- ✓ Using inline assembly
- ✓ Style guide violation
- ✓ Unsafe type inference
- ✓ 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

Likelihood	🟥 High	🟧 Medium	🟨 Low
🟥 High	Critical	High	Medium
🟧 Medium	High	Medium	Low
🟨 Low	Medium	Low	Low

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.

High Severity Issues

Incorrect Interface will lead to contract being non-operational

Resolved

Path

Perpetual.sol

Description

There are two important interfaces uses by Perpetual.sol namely IOrderBook.sol and IExchangeRouter.sol Both of them are initialized in the constructor. The exchange router is responsible for calling `createIncreasePosition` in the gmx contract.
However if we take a look at ExchangeRouter.sol implementation, there is no such function as `createIncreasePosition`

The official implementation of this function is present in PositionRouter.sol . If we consider the possibility that IExchangeRouter will have address of position router, the name is highly misleading.

Moreover, if we look at IOrderbook.sol , it has functions like `createIncreasePosition` , `createIncreasePositionETH`, `createDecreasePosition` and many more functions that are absent in OrderBook.sol

That means we have two interfaces namely IOrderBook.sol and IExchangeRouter.sol and both of them have `createIncreasePosition` as an interface function.

Provided that both orderbook and exchange router will have different addresses, subsequent call to `createIncreasePosition` will not be able to execute since there is not function present in the actual implementation.

Recommendation

If the name is misleading, change the name otherwise change the implementation of interface.

Some critical functions won't work in the contract

Resolved

Path

Perpetual.sol

Description

There are several critical functions that in the current state will permanently revert.

Take a look at these functions:
1. getRequestQueueLengths
2. executeIncreasePosition
3. cancelIncreasePosition
4. executeDecreasePosition
5. cancelDecreasePosition
6. closePositionById

Notice how all of these function are being called on orderbook address. If we look at the actual implementation of IOrderBook.sol

https://github.com/gmx-io/gmx-contracts/blob/master/contracts/core/interfaces/IOrderBook.sol

These functions do not exist. Instead, these functions are present in PositionRouter.sol
getRequestQueueLengths
executeIncreasePosition
cancelIncreasePosition
executeDecreasePosition
cancelDecreasePosition
createDecreasePosition

Calling these functions on wrong contract will make them permanently. As a result, the contract will be useless since any user won't be able to execute them.

Recommendation

Ensure that the functions to be called are actually present in the external contract before. Also user given interfaces from GMX instead of creating your own.

Medium Severity Issues

Funds might get locked in the contract

Resolved

Path

src/Perpetual.sol

Description

When integrating the GMX platform in a protocol there are chances that deposits, withdrawals and orders may be cancelled due to requirements specified in the request which cannot be fulfilled leading to funds getting locked in the contract. Also in the Perpetual.sol contract there is no way to withdraw stuck funds.

Reference

Funds loss check

Recommendation

To resolve the issue do check where the funds and gas refunds are transferred on cancellations and it matches the expectations.

Use safeTransferFrom instead of transferFrom

Resolved

Path

src/Perpetual.sol

Function name

`openPosition()`, `createIncreasePosition()`

Description

In Perpetual.sol contract `transferFrom()` is used to transfer fees in `openPosition()` and `createIncreasePosition()`. Though there are no direct consequences, sometimes transfers may fail so it is better to use `safeTransferFrom()` instead.

Recommendation

Use `safeTransferFrom()` instead of `transferFrom()`

Low Severity Issues

GMX exchange router address might change

Resolved

Path

Perpetual.sol

Description

GMX's exchange router is only initialized in constructor and there is no way to change it's address. According to gmx integration notes

If using contracts such as the ExchangeRouter, Oracle or Reader do note that their addresses will change as new logic is added

The contract will be useless if the address of exchange router changes.

Recommendation

Ensure that there is a setter function to change the contract address.

Increase position does not return position key

Resolved

Path

Perpetual.sol

Description

The contract tries to call `createIncreasePosition` via two functions. However the returned position key is no being cached. As a result, user needs to manually find their position id.

Recommendation

Return the cached positionId.

Informational Issues

Misleading Return Value in toggle()

Resolved

Path
Perpetual.sol

Description
Perpetual.sol contains two function , `openPosition` and `createIncreasePosition` , both calling `positionRouter.createIncreasePosition`
The only difference between the two is , one can specify callback address and referral while other can't.

We can safely conclude that both of the functions are inherently doing the same thing so one of them does not serve any additional purpose.

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 PerpX. We performed our audit according to the procedure described above. Issues of High, Medium and Low severity were found.PrepX team resolved all the issues mentioned

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+ Years of Expertise	**1M+** Lines of Code Audited
50+ Chains Supported	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

July 2025

For

perpX

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com