SMART CONTRACT AUDIT REPORT

SparkDEX - Perp

Version: 0.2

Protofire
October, 2024

Table Of Contents

PROTOFIRE

The SparkDEX - Perp audit report

The audit report was prepared for the **SparkDEX team** and includes following Github repository and files:

https://github.com/SparkDEX/perp-smart-contracts/ audited commit c8371937c7a4f1e9de677bc360dbf985310995c6:

Update 1
The SparkDEX team has updated their code. Commit with updated code: 354743877324e02ba97a538aac7c9c4c871cf4ea.
The remediation commit for the Update 1 is: 1996bf7d40cc8e4ada3d69d42d7362d7b905055d.

The code repository does not contain any tests.

Procedure

The audit includes the following procedures:

- code analysis by the Slither static analyzer, followed by manual analysis and selection of problems
- manual code analysis, including logic check
- checking the business logic for compliance with the documentation (or white paper)

In addition, during the audit, we also provide recommendations on optimizing smart contracts and using best practices.

PROTOFIRE

Summary of audit findings

Severity	Count
HIGH	0
MEDIUM	0
LOW	4
INFORMATIONAL	6
TOTAL	**10**

Severity classification

High severity issues

High severity issues can lead to a direct or indirect loss of funds by both users and owners, a serious violation of the logic of the contract, including through the intervention of third parties. Such issues require immediate correction.

Medium severity issues

Medium severity issues may cause contract functionality to fail or behave incorrectly. Also, medium severity issues can cause financial damage. Such issues require increased attention.

Low severity issues

Low severity issues do not have a major security impact. It is recommended that such issues be taken into account.

Informational severity issues

These issues provide general guidelines for writing code as well as best practices.

Smart contract Executor [SDP-01]

The Executor contract is a core component of a decentralized trading platform that handles order execution and position liquidation. It works with external keepers to process market, limit, and trailing stop orders, ensuring price accuracy through Pyth price updates. For protection, only whitelisted keepers can initiate these actions. The contract validates orders and liquidates positions based on margin thresholds and reliable price data.

No issues have been identified.

PROTOFIRE

Smart contract FundingTracker [SDP-02]

The FundingTracker contract is responsible for calculating and updating funding rates for different markets and collateral assets in a decentralized trading protocol. Funding rates are determined based on the imbalance between long and short positions in each market. The contract ensures accurate calculations by using time intervals and updates only when the funding interval has elapsed. It integrates with other core protocol contracts (PositionManager and Executor) to manage positions and facilitate funding updates. The FundingTracker is essential for maintaining a fair compensation mechanism between long and short traders on the platform.

No issues have been identified.

Smart contract OrderBook [SDP-03]

The OrderBook contract is the core component of a decentralized exchange, responsible for managing the placement and cancellation of orders. It supports market, limit, stop, and trailing stop orders, allowing users to trade assets with various strategies. The contract validates all orders to ensure market integrity and charges a small execution fee to pay keepers who process orders.

ID: **SDP03-01**	Severity: **Low**	Status: **Fixed**

Function lacks validation

The `setOrderExecutionFee()` function (managed by governance) allows setting the orderExecutionFee without any validations. This can lead to potential issues if the fee is set to an excessively high value (as well as front-run).

Although this function is controlled by the `Timelock.sol` contract, the timelock mechanism is not enforced, meaning governance can change the `orderExecutionFee` immediately without delays.

Consider adding validation to prevent setting high values.

PROTOFIRE

Smart contract PositionManager [SDP-04]

The PositionManager contract is a core component of a decentralized exchange, handling the creation, modification, and closure of trading positions. It allows users to open long or short positions on various markets, add or remove margin to control leverage, and tracks fees generated from order executions. The contract ensures accurate calculation of profit and loss (PnL) and distributes fees to keepers (order executors), a liquidity pool, and the protocol's treasury. It maintains detailed records of open interest (OI) for risk management purposes.

No issues have been identified.

Smart contract ReferralStorage [SDP-05]

The ReferralStorage contract is a core component of a decentralized exchange's referral program. It enables users to earn rewards by referring others and allows referrers to set customized fee discounts or rebates for their referees. The system uses a tier-based approach where different tiers offer varying levels of rebates and discounts. The contract stores referral codes, links them to referrers and referees, and manages the distribution of referral rewards.

ID: **SDP05-01**	Severity: **Low**	Status: **Acknowledged**

A front-run possibility

The function `registerCode()` can be front-run by any user. For example, userA wants to be the owner of a code and sends a transaction to be the owner of this code, but userB front-runs userA and becomes the owner of this code instead of the userA.

ID: **SDP05-02**	Severity: **Low**	Status: **Acknowledged**

Loss of rewards for referrers

The `_setTraderReferralCode()` function allows setting the trader referral code for an account without any restrictions. This can lead to the referral code being changed frequently, potentially causing the referrer to not receive the appropriate rewards.

ID: **SDP05-03**	Severity: **Info**	Status: **Fixed**

Unindexed events

Several events are declared without indexed parameters. Indexing event parameters can significantly improve the efficiency of event filtering and querying in the Ethereum blockchain. Without indexed parameters, it becomes more challenging to search for specific events, especially in large datasets.

Smart contract Store [SDP-06]

The Store contract is a core component of a decentralized exchange, acting as the central repository for assets that back trader positions. It persistently stores data on supported assets and markets, defining parameters like fees and leverage limits. Users can deposit assets to provide liquidity, and these funds are used to settle trader profits and losses. The Store includes a buffer mechanism that temporarily absorbs trader losses to protect liquidity providers. Additionally, the contract manages protocol-generated fees and user deposits/withdrawals.

ID: **SDP06-01**	Severity: **Low**	Status: **Fixed**

Function lacks validation

The `setOrderExecutionFee()` function (managed by governance) allows setting the orderExecutionFee without any validations. This can lead to potential issues if the fee is set to an excessively high value (as well as front-run).

Although this function is controlled by the `Timelock.sol` contract, the timelock mechanism is not enforced, meaning governance can change the `orderExecutionFee` immediately without delays.

Consider adding validation to prevent setting high values.

ID: **SDP06-02**	Severity: **Info**	Status: **Acknowledged**

Keeper backend work

Since the work of the keepers is not within the scope of the audit, we recommend ensuring that the call to `_setGlobalUPLs()` is executed for all assets for which orders will be processed in the `_cancelOrder()` loop.

Smart contract AddressStorage [SDP-07]

The AddressStorage contract serves as a centralized repository for storing addresses used within a larger decentralized application. It provides a secure and organized way to manage these addresses, with access control restricted to the governance mechanism. The contract offers functions for setting and retrieving addresses associated with specific keys. This allows for flexibility and easy updates as the application evolves.

No issues have been identified.

PROTOFIRE

Smart contract BatchSender [SDP-08]

The BatchSender contract facilitates efficient batch transfers of ERC20 tokens. It's primarily designed for scenarios like distributing rewards or airdrops to multiple recipients in a single transaction. The contract allows keepers to execute batch transfers.

No issues have been identified.

PROTOFIRE

Smart contract FTSOv2 [SDP-09]

The FTSOv2 contract integrates with the Flare Network to retrieve and manage price feeds from the Flare Time Series Oracle (FTSO). The contract includes functionalities to set price feed stale periods, and fetch the latest FTSO prices for given token symbols, while enforcing stale period constraints to maintain data accuracy.

ID: **SDP09-01**	Severity: **Info**	Status: **Fixed**

Stale price from the oracle

The `votingEpochDurationSeconds` is set using the `stateData.votingEpochDurationSeconds` value. If this value is set to 0 in the constructor, it will cause the price feed to always be stale.

PROTOFIRE

Smart contract Governable [SDP-10]

The Governable contract provides a basic access control mechanism for a decentralized application. It designates a specific address as the "governance" entity, which has exclusive control over sensitive functions within the system.

No issues have been identified.

Smart contract ReferralReader [SDP-11]

The ReferralReader contract simplifies the process of retrieving referral information in bulk. It interacts with an external ReferralStorage contract, which stores the relationship between referral codes and their owners.

No issues have been identified.

PROTOFIRE

Smart contract Timelock [SDP-12]

The Timelock contract introduces a time delay for administrative actions within a related decentralized application. It allows authorized administrators to signal proposed changes, which must wait a predefined buffer period before execution. This mechanism adds a layer of security and prevents hasty or unauthorized changes to the system.

ID: **SDP12-01**	Severity: **Info**	Status: **Acknowledged**

Not all functions have timelock

In the contract only some functions are under the timelock. Almost all functions can be called without any signal/buffer.

Only these functions have timelock:

1. `setGov()`
2. `setAddress()`
3. `setAsset()`
4. `setMarket()`

Also there is a function `setMarketWithoutSignal()` that performs the same action, but without a timelock.

Consider using timelock pattern for all important functions. And add documentation for the rest.

Smart contract TradingValidator [SDP-13]

The TradingValidator contract implements risk controls, primarily focusing on limiting open interest (OI) and potential losses within liquidity pools. It allows governance to set maximum OI thresholds for specific markets and assets, preventing positions from being opened when these limits are reached. The contract also tracks an amortized profit and loss metric for liquidity pools, enforcing profit limits to protect against excessive drawdowns.

ID: **SDP13-01**	Severity: **Info**	Status: **Acknowledged**

Zero address check missed for the _addressStorage

The constructor does not include a check to ensure that the `_addressStorage` address is not the zero address (`0x0000000000000000000000000000000000000000`).

PROTOFIRE

Smart contract ProxyAdmin [SDP-14]

The `ProxyAdmin` contract is an administrative contract that leverages role-based access control to manage function calls and token withdrawals. It allows only users with the `ADMIN_ROLE` to execute calls to other contracts with value and to withdraw tokens held by the contract.

ID: **SDP14-01**	Severity: **Info**	Status: **Fixed**

Zero address check missed for the admin

The constructor does not include a check to ensure that the admin address is not the zero address (0x0000000000000000000000000000000000000000). Assigning the zero address as the admin can lead to a loss of control over the contract. The zero address is not a valid owner and cannot perform administrative functions, which can render the contract unmanageable.

PROTOFIRE

The list of additional risks that should be considered by the development team

- The Solidity pragma version 0.8.22 is used. This version introduces the PUSH0 opcode, which may not be supported by all EVM-compatible chains. The PUSH0 opcode was introduced in Solidity version 0.8.20.
- In the OrderBook.sol and Store.sol files, the whitelistedFundingAccount can make transactions on behalf of users. This introduces a centralization risk, as a compromise of the private key associated with a whitelisted account can put all users with approved funds at risk. The ability for whitelisted accounts to make transactions on behalf of users centralizes control, making the system more vulnerable to a single point of failure.
- In the PositionManager.sol file, the function uses keccak256 to generate a hash based on the _user, _asset, and _market parameters. While the risk is extremely small, there is a theoretical possibility of a hash collision, where two different sets of inputs produce the same hash.
- In the Store.sol file, if the amount required from the pool (diffToPayFromPool) is greater than the current poolBalance, it indicates that the user's profit and loss (PnL) exceeds the available pool balance, preventing the user from making any withdrawals.
- In OrderBook.sol, the OpenZeppelin SignatureChecker utility is used to verify signatures by checking if the caller's code size is 0, which typically indicates an externally owned account (EOA) rather than a smart contract. However, this check may be unreliable when used within a contract's constructor, as the code size of a contract under

construction is also 0, making it difficult to distinguish accurately between an EOA and a smart contract at that stage.

- The project contains multiple instances where contracts are being referenced and utilized that are marked as "out of scope." Utilizing out-of-scope contracts can introduce unknown security vulnerabilities, as these contracts may not have been thoroughly audited or reviewed.

Conclusion

During the audit 4 low and 6 info severity issues were found.

However, the provided repository does not contain tests for contracts. Adding tests for project contracts is strongly recommended.

Please note that auditing is not a complete replacement for unit tests and integration tests.

PROTOFIRE

Disclaimer

Note that smart contract audit provided by Protofire is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, Protofire recommends proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Smart contract audit provided by Protofire shall not be used as investment or financial advice.