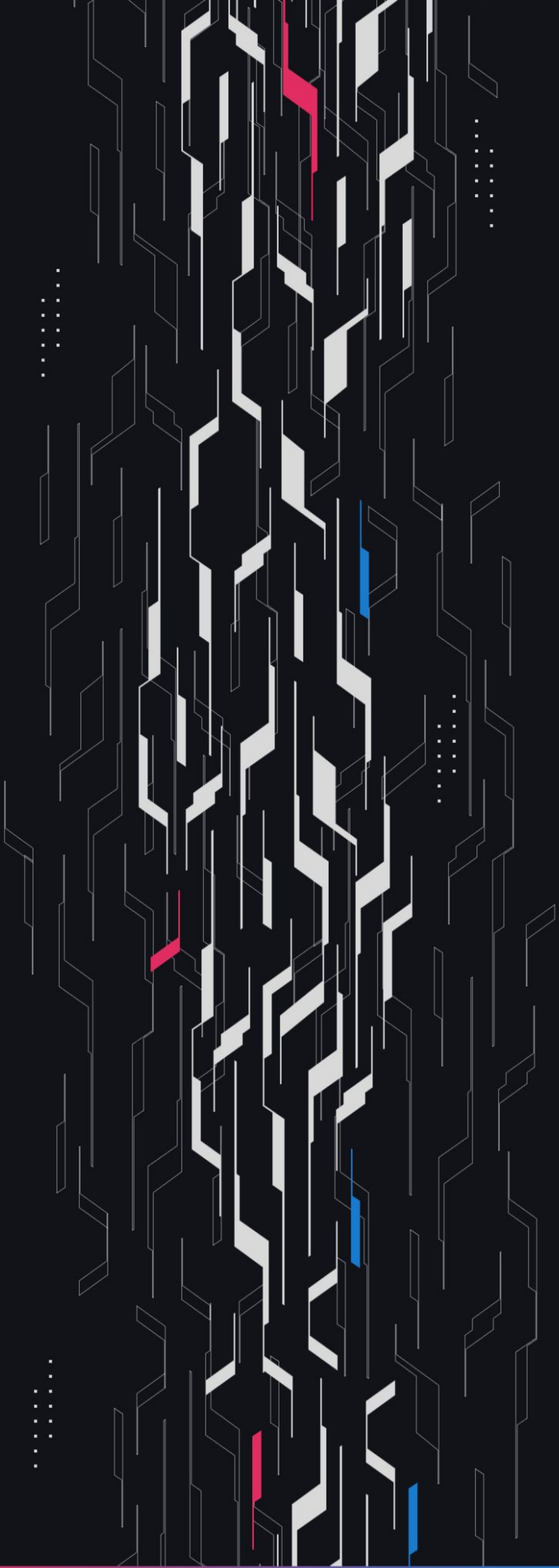
GUARDIAN

Smardex

USDN

Security Assessment

December 18th, 2024



Summary

Audit Firm Guardian

Prepared By Roberto Reigada, Andreas Zottl, Vladimir Zotov,
Owen Thurm, Kiki, Dogus Kose, Wafflemakr

Client Firm Smardex

Final Report Date December 18, 2024

Audit Summary

Smardex engaged Guardian to review the security of its review of their decentralized synthetic dollar. From the 30th of September to the 4th of November, a team of 7 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Issues Detected Throughout the engagement 4 High/Critical issues were uncovered and promptly remediated by the Smardex team. Several issues impacted the fundamental behavior of the protocol, following their remediation Guardian believes the protocol to uphold the functionality described for the USDN protocol.

For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

🔗 Blockchain network: **Ethereum**

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

📊 Code coverage & PoC test suite: https://github.com/GuardianAudits/usdn-fuzzing

Table of Contents

Project Overview

Project Summary

Project Name	USDN
Language	Solidity
Codebase	https://github.com/SmarDex-Ecosystem/usdn-contracts
Commit(s)	(initial) 7462fc4d1b886129a4091dfc91ff5304d1d3a970 (final) 2b8daa04c502cfea2907c333d9577ab866236c9a

Audit Summary

Delivery Date	December 18, 2024
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	2	0	0	0	0	2
● High	2	0	0	0	0	2
● Medium	34	0	0	15	0	19
● Low	79	0	0	43	1	35

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	● Critical	● High	● Medium
Likelihood: *Medium*	● High	● Medium	● Low
Likelihood: *Low*	● Medium	● Low	● Low

Impact

High	Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.
Medium	A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.
Low	Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High	The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.
Medium	An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.
Low	Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts. Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Invariants Assessed

During Guardian's review of USDN, fuzz-testing with Echidna was performed on the protocol's main functionalities. Given the dynamic interactions and the potential for unforeseen edge cases in the protocol, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 10,000,000+ runs with a prepared Echidna fuzzing suite.

ID	Description	Passed	Remediation	Run Count
GLOB-01	A positions tick should never be above the _highestPopulatedTick	✅	✅	10M+
GLOB-02	The current divisor should never equal the MIN_DIVISOR."	✅	✅	10M+
GLOB-03	FundingPerDay should never equal 0	✅	✅	10M+
GLOB-04	Each pending action should have an associated securityDeposit value.	✅	✅	10M+
GLOB-05	Trading expo should never go to 0.	✅	✅	10M+
GLOB-06	Position should never have a leverage smaller that 1.	✅	✅	10M+
GLOB-07	The internal total balance the contract deals with should not be bigger than the real balanceOf.	✅	✅	10M+
ERR-01	Non-whitelisted error should never appear in a call	❌	✅	10M+
DEPI-01	Sender's ETH balance decreased by security deposit	✅	✅	10M+

Invariants Assessed

ID	Description	Passed	Remediation	Run Count
DEPI-02	Sender's wstETH balance decreased by deposited amount	✅	✅	10M+
DEPI-03	Sender's SDEX balance decreased	✅	✅	10M+
DEPI-04	Protocol's ETH balance increased by security deposit	✅	✅	10M+
DEPI-05	Protocol's wstETH balance increased by deposit minus pending actions	✅	✅	10M+
DEPV-01	Recipient's USDN shares increased after validation	✅	✅	10M+
DEPV-02	Caller’s USDN shares unchanged	✅	✅	10M+
DEPV-03	Validator’s USDN shares unchanged	✅	✅	10M+
DEPV-04	Validator's ETH balance increased by security deposit after validation	✅	✅	10M+
DEPV-05	USDN token total supply changed by pending tokens after validation	✅	✅	10M+
DEPV-06	Caller’s wstETH balance unchanged	✅	✅	10M+
DEPV-07	Protocol's wstETH balance decreased by pending actions after validation	✅	✅	10M+

Invariants Assessed

ID	Description	Passed	Remediation	Run Count
DEPV-08	Validator's wstETH balance unchanged	✅	✅	10M+
DEPV-09	Caller’s wstETH balance unchanged	✅	✅	10M+
WITHI-01	Sender's ETH balance decreased by security deposit	✅	✅	10M+
WITHI-02	Sender's USDN shares decreased by withdrawn amount	✅	✅	10M+
WITHI-03	Protocol's ETH balance increased by security deposit minus last action's deposit	✅	✅	10M+
WITHI-04	Protocol's USDN shares increased by withdrawn amount plus pending actions	✅	✅	10M+
WITHV-01	Sender's ETH balance increased by action's security deposit value	✅	✅	10M+
WITHV-02	If successful, sender's wstETH balance increased or remained the same	✅	✅	10M+
WITHV-03	If successful, protocol's ETH balance decreased by action's security deposit value	✅	✅	10M+
WITHV-04	If successful, protocol's USDN shares decreased	✅	✅	10M+
WITHV-05	If successful, protocol's wstETH balance decreased by at least pending actions	✅	✅	10M+

Invariants Assessed

ID	Description	Passed	Remediation	Run Count
POSOPNI-01	Protocol's ETH balance increased by security deposit minus last action	✅	✅	10M+
POSOPNI-02	Sender's ETH balance decreased by security deposit minus last action	✅	✅	10M+
POSOPNI-03	Protocol's wstETH balance increased by deposit amount minus pending actions	✅	✅	10M+
POSOPNI-04	Sender's wstETH balance decreased by deposit amount	✅	✅	10M+
POSCLOSI-01	If successful, sender's ETH balance decreased by security deposit	✅	✅	10M+
POSCLOSI-02	If successful, validator's pending action is set to ValidateClosePosition	✅	✅	10M+
POSCLOSI-03	If successful, protocol's ETH balance increased by security deposit	✅	✅	10M+
POSCLOSI-04	Sender's wstETH balance unchanged	✅	✅	10M+
POSCLOSI-05	Protocol's wstETH balance unchanged	✅	✅	10M+
POSOPNV-01	If successful, validator's ETH balance increased by security deposits	✅	✅	10M+
POSOPNV-02	If successful, protocol's ETH balance decreased by security deposits	✅	✅	10M+

Invariants Assessed

ID	Description	Passed	Remediation	Run Count
POSOPNV-05	Protocol's wstETH balance decreased by pending actions	✅	✅	10M+
POSOPNV-06	Sender's wstETH balance unchanged	✅	✅	10M+
POSCLOSV-01	If successful, sender's ETH balance increased by security deposit	✅	✅	10M+
POSCLOSV-02	If successful, protocol's ETH balance decreased by security deposit	✅	✅	10M+
POSCLOSV-03	If successful, protocol's wstETH balance decreased by more than pending actions	✅	✅	10M+
POSCLOSV-04	If successful, protocol's wstETH balance decreased by less than close amount + pending actions	✅	✅	10M+
POSCLOSV-05	If successful, recipient's wstETH balance increased by less than close amount	✅	✅	10M+
POSCLOSV-06	If successful, recipient's wstETH balance increased	❌	✅	10M+
POSCLOSV-07	If successful and sender != validator, validator's ETH balance unchanged	✅	✅	10M+
POSCLOSV-08	If successful and sender != recipient, sender's wstETH balance unchanged	✅	✅	10M+
POSCLOSV-09	If successful and recipient != validator, recipient's ETH balance unchanged	✅	✅	10M+

Invariants Assessed

ID	Description	Passed	Remediation	Run Count
POSCLOSV-10	If successful and recipient != validator, validator's wstETH balance unchanged	✅	✅	10M+
PENDACTV-01	Correct number of actions validated	✅	✅	10M+
PENDACTV-02	Sender's ETH balance increased by security deposit	✅	✅	10M+
PENDACTV-03	Protocol's ETH balance decreased by security deposit	✅	✅	10M+

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	Rebalancer Position Closed Twice	Logical Error	● Critical	Resolved
C-02	Validate Withdraw Uses Current Total Shares	Logical Error	● Critical	Resolved
H-01	Users Do Not Pay Funding During Initiation	Logical Error	● High	Resolved
H-02	Liquidation Rewards Sent To The Rebalancer Will Be Lost	Unexpected Behavior	● High	Resolved
M-01	Imbalance Does Not Count Funding	Logical Error	● Medium	Acknowledged
M-02	Positions Opened Above Max Leverage	MEV	● Medium	Acknowledged
M-03	Bad Debt Value Extraction	Documentation	● Medium	Resolved
M-04	pendingBalanceVault Underflow	Rounding	● Medium	Resolved
M-05	Initiators Avoid A Portion Of Funding	Logical Error	● Medium	Acknowledged
M-06	_triggerRebalance Does Not Account For Liquidation Rewards	Unexpected Behavior	● Medium	Acknowledged
M-07	Slippage Check Occurs Before Adjustment	Validation	● Medium	Resolved
M-08	removeBlockedPendingAction Extractable Value	MEV	● Medium	Resolved
M-09	Tick Liquidation Penalty Cannot Be Reset	Unexpected Behavior	● Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-10	Incorrect pendingBalanceVault Correction	Logical Error	● Medium	Resolved
M-11	_calcPositionSizeBonus Errant Token Amount	Logical Error	● Medium	Resolved
M-12	Inaccurate Imbalance Check In _checkImbalanceLimitOpen	Validation	● Medium	Resolved
M-13	Exposure During Closure Unaccounted	Logical Error	● Medium	Acknowledged
M-14	Innacurate isLiquidationPending Check In ValidateOpen	Validation	● Medium	Resolved
M-15	Fee Change Hurts Traders	Logical Error	● Medium	Acknowledged
M-16	Min Price Validation Excludes Fee	Validation	● Medium	Resolved
M-17	Risk Free Trades With Rebalancer	Gaming	● Medium	Resolved
M-18	No Rebalancer Trigger On Individual Liquidation	Unexpected Behavior	● Medium	Acknowledged
M-19	Init Close Of A Liquidatable Pos Permitted	Validation	● Medium	Resolved
M-20	Increased Liquidation Rewards	Gaming	● Medium	Acknowledged
M-21	Extracting From Rebalancer Bonus	Gaming	● Medium	Resolved
M-22	Depositors Banned From Rebalancer If Liquidated	DoS	● Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-23	Blocked Queue Due To Non-Validatable Action	DoS	● Medium	Resolved
M-24	Bad Debt Not Handled In Validate Withdraw	DoS	● Medium	Resolved
M-25	Closed Amt Not Seized If Its Value Is < 0	Validation	● Medium	Resolved
M-26	User Cant Close Rebalanced Position	Validation	● Medium	Resolved
M-27	Liquidatable Positions Can Be Opened	Validation	● Medium	Acknowledged
M-28	Validate Deposit Ignores Funding	Logical Error	● Medium	Resolved
M-29	Sandwich Liquidations	MEV	● Medium	Acknowledged
M-30	Validate Withdrawal Ignores Funding	Logical Error	● Medium	Acknowledged
M-31	Remove Pending Position Price Can Be Stale	Logical Error	● Medium	Acknowledged
M-32	Neutral Price Used In Init Functions	Validation	● Medium	Acknowledged
M-33	Funding Rate Affected By Updates	Logical Error	● Medium	Acknowledged
M-34	Old Vault Validations Swing USDN Price	Gaming	● Medium	Acknowledged
L-01	Fee Amounts Use Round Down Division	Rounding	● Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-02	Incorrect Action Id Used	Logical Error	● Low	Resolved
L-03	Risk Of Immediately Liquidatable Positions	DoS	● Low	Acknowledged
L-04	Rebalancer Depositors Might Not Receive Any Bonus	Validation	● Low	Acknowledged
L-05	Lacking Configuration Validations	Validation	● Low	Resolved
L-06	Innacurate wstETH Price Calculation	Unexpected Behavior	● Low	Acknowledged
L-07	Stuck Action Functions Lack ReentrancyGuard	Unexpected Behavior	● Low	Resolved
L-08	TickMath Rounding	Documentation	● Low	Acknowledged
L-09	_getLatestStoredPythPrice Lacks Validation	Validation	● Low	Resolved
L-10	Storage Compatibility	Upgradeability	● Low	Resolved
L-11	Unexpected Share Approval Reverts	DoS	● Low	Resolved
L-12	Read Only Reentrancy Risk	Reentrancy	● Low	Resolved
L-13	Tick Penalty Changes Leverage	Documentation	● Low	Resolved
L-14	Missing Require Check In setMinLongPosition	Validation	● Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-15	validateOpenPosition Might Not Be Completed	Validation	● Low	Resolved
L-16	USDN Depositors May Pay Less Fees On Withdrawal	Unexpected Behavior	● Low	Acknowledged
L-17	Typo	Typo	● Low	Resolved
L-18	Insufficient Liquidation Incentives	Incentives	● Low	Acknowledged
L-19	Liquidations Untriggered By Recent Price	Unexpected Behavior	● Low	Acknowledged
L-20	Inaccurate NatSpec	Documentation	● Low	Resolved
L-21	Rebalancer Always Utilizes All Available Liquidity	Documentation	● Low	Acknowledged
L-22	Rebalancer NonReentrant Modifiers	Reentrancy	● Low	Resolved
L-23	Missing Deadline Documentation	Documentation	● Low	Resolved
L-24	Invalid Long Exposure Perturbs Accounting	Documentation	● Low	Acknowledged
L-25	Position Fee Not Refunded On Removal	Unexpected Behavior	● Low	Acknowledged
L-26	Gaming Vault Minted Shares	Gaming	● Low	Acknowledged
L-27	0 Shares Wrapping	Validation	● Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-28	Hardcoded Maximum Pyth Fee	Validation	● Low	Acknowledged
L-29	Rebalancer Tick Increment May Reduce Exposure	Unexpected Behavior	● Low	Acknowledged
L-30	Positive Value Liquidations May Still Cause Bad Debt	Unexpected Behavior	● Low	Acknowledged
L-31	USDN Price Can Be Easily Manipulated	Gaming	● Low	Resolved
L-32	Lacking securityDepositValue Validation	Validation	● Low	Resolved
L-33	Depositors Pay Reduced Fees	Logical Error	● Low	Resolved
L-34	Potential Precision Loss In _positionValue Calculation	Unexpected Behavior	● Low	Acknowledged
L-35	TotalSupply Exceeded By User Balances	Rounding	● Low	Resolved
L-36	getHighestPopulatedTick Might Return An Incorrect Tick	Unexpected Behavior	● Low	Resolved
L-37	No Slippage On Validation	Validation	● Low	Acknowledged
L-38	_triggerRebalance Does Not Account For The Current Action Effect	Unexpected Behavior	● Low	Acknowledged
L-39	DOS State Due To s._minLongPosition Restriction	Validation	● Low	Acknowledged
L-40	_usdnRebaseInterval Unassigned On Initialization	Unexpected Behavior	● Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-41	Full Operational Halt During Protocol Pause	Unexpected Behavior	● Low	Acknowledged
L-42	Update Events Emitted On No Update	Events	● Low	Acknowledged
L-43	Deposits Experience Invalid Funding	Logical Error	● Low	Acknowledged
L-44	Insufficient removeBlockedPendingAction Wait	Validation	● Low	Resolved
L-45	LastUpdateTimestamp Errantly Initialized	Logical Error	● Low	Acknowledged
L-46	Potential Reverts From New Actionable Actions	DoS	● Low	Resolved
L-47	Incorrect Event Emitted For Close Actions	Events	● Low	Acknowledged
L-48	Burned SDex Unforgiven On Removal	Unexpected Behavior	● Low	Acknowledged
L-49	Imbalance Variables Updates	Validation	● Low	Resolved
L-50	Preview Functions Not Accurate	Unexpected Behavior	● Low	Resolved
L-51	Liquidations May Occur Earlier Than Expected	Logical Error	● Low	Acknowledged
L-52	Protocol Avoids Correct TVL Growth	Warning	● Low	Acknowledged
L-53	Imbalance Breached While Validating Open	Logical Error	● Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-54	Validators With No Receive Function	Documentation	● Low	Resolved
L-55	USDN May Temporarily Lose Peg	Warning	● Low	Resolved
L-56	Funding Retroactively Affected By Admin Update	Logical Error	● Low	Acknowledged
L-57	Sparse Queue After Pyth Validation Ends	Unexpected Behavior	● Low	Partially Resolved
L-58	Instantaneous Actionable Actions	Unexpected Behavior	● Low	Acknowledged
L-59	Missing Validation For Admin Functions	Validation	● Low	Resolved
L-60	Incorrect Return Value For Failed Actions	Composability	● Low	Resolved
L-61	Leverage DoS With lastPrice Update	DoS	● Low	Acknowledged
L-62	Small Rebalancer Positions Are Allowed	Validation	● Low	Acknowledged
L-63	Some ERC20 Tokens Are Incompatible	Documentation	● Low	Resolved
L-64	Liquidatable Positions Can Be Transfered	Documentation	● Low	Acknowledged
L-65	Sandwich Validate Close Pos	MEV	● Low	Acknowledged
L-66	Rebalancer Only Triggered On Liquidations	Logical Error	● Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-67	Rebalancing Position Impacts Traders	Documentation	● Low	Acknowledged
L-68	Sandwich Deposit/Withdraw	MEV	● Low	Acknowledged
L-69	DoS By Occupying The Validator	Griefing	● Low	Acknowledged
L-70	Bad Debt Not Handled In Validate Open Pos	DoS	● Low	Resolved
L-71	Ether Griefing Attack	Logical Error	● Low	Resolved
L-72	Sandwich Remove Pending Open Pos	MEV	● Low	Acknowledged
L-73	Fee Can Be Avoided By Depositing Dust Amts	Validation	● Low	Acknowledged
L-74	Traders Can Choose Prices In Edge Cases	Validation	● Low	Acknowledged
L-75	SDEX Burn Could Be Bypassed	Validation	● Low	Resolved
L-76	Imbalance Checks Wrong In Edge Cases	Validation	● Low	Resolved
L-77	Stepwise Jump In sdexToBurn Calc	Logical Error	● Low	Acknowledged
L-78	Rebalancer tradingExpoToFill Not Entirely Filled	Unexpected Behavior	● Low	Acknowledged
L-79	Funding Is Charged During Paused States	Logical Error	● Low	Resolved

C-01 | Rebalancer Position Closed Twice

Category	Severity	Location	Status
Logical Error	● Critical	Rebalancer.sol: 473	Resolved

Description PoC

When initiating a close order for the rebalancer position with the initiateClosePosition function it is possible that the initiation of the close triggers the rebalancer itself.

This results in many issues, including but not limited to:
- Double counting the balance which is being closed
- Removing the closed amount from the rebalancer's previous tick twice
- Overwriting the state updates which were made in the Rebalancer.updatePosition function

Recommendation

Revert the Rebalancer.initiateClosePosition function when a rebalancer is triggered by the _usdnProtocol.initiateClosePosition function call.

Resolution

Smardex Team: The issue was resolved in PR#627.

C-02 | Validate Withdraw Uses Current Total Shares

Category	Severity	Location	Status
Logical Error	● Critical	UsdnProtocolVaultLibrary.sol: 1057	Resolved

Description PoC

The _validateWithdrawalWithAction function does not use the totalShares amount saved at init in the pending action. Instead, it calculates the amount of assets a user should receive by using the current totalShares amount, together with the available balanceVault from init.

Therefore reducing the totalShares amount between init and validation of a withdrawal will give the user more assets than the user should receive and vice versa.

This can be exploited in the following way:
• Attacker deposits funds
• Attacker initiates multiple withdrawals of part of his shares for example two withdrawals of 50% of his shares
• The first withdrawal is validated and reduces the totalShares amount
• The second withdrawal is validated and uses the balanceVault from init but the reduced totalShares amount to calculate how many assets the attacker receives and therefore sends the attacker more assets than he should receive
• Attacker repeats the process to drain the protocol

All of this is independent of price changes allowing the attack to be performed swiftly and consistently.

Recommendation

Use the totalShares amount saved at init in the pending action instead of the current one.

Resolution

Smardex Team: The issue was resolved in PR#638.

H-01 | Users Do Not Pay Funding During Initiation

Category	Severity	Location	Status
Logical Error	● High	Global	Resolved

Description PoC

Upon initiation, users positions are effectively created and they have an effect on the skew in exposure between the vault and traders, thus the funding is accounting for this initiated position. However the position does not pay the funding during the period from [initiateOpen, validateOpen] as the new position exposure is computed with the new liquidation price, accounting for the updates made to the liquidation multiplier.

The vault and long balances are corrected with the _validateOpenPositionUpdateBalances function upon validation so that the funding which was not actually charged to the user is not included in the aggregate balance accounting, therefore this avoids a fundamental accounting error. However there are two issues which stem from this behavior.

Firstly, the user who initiated their position does not have to pay their funding fees while they have still manipulated the skew. Thus potentially making other traders pay funding while not having to pay funding themselves. A malicious actor could initiate a large long position and wait to validate it as long as they can before immediately initiating and validating a close action to affect the funding skew as much as possible while paying as little as possible themselves.

And secondly, the trader is immune from the funding fees which take place in the timeframe between [startPriceTimestamp, lastPriceTimestamp]. Technically the trader should experience these funding fees as their position is not opened at the lastPrice, but at the startPrice which can be from an earlier timestamp.

Recommendation

Consider charging the position the funding fees during the period: [initiateOpen, validateOpen]. This way traders are held accountable for the affect they have on the skew, and they will pay the funding fees they rightfully should from the period [startPriceTimestamp, lastPriceTimestamp].

The downside of this approach is that the trader is charged/credited with funding in the range [initiateOpen, startPriceOnValidation.timestamp] which they technically shouldn't be since their position is not officially opened and exposed to this price action. However this range should be minimal given that the oracle middleware aims to use the earliest price in the range from [initiateOpen + 24 seconds, validateOpen] to execute validations at.

This could be implemented by using a liquidation price based upon the liquidationMultiplier recorded at initiation for the new position exposure calculation. Additionally, the new value of the position should be based upon a liquidation price with the latest liquidationMultiplier. In the _validateOpenPositionWithAction function when computing the expoAfter for the normal validation case, where leverage does not exceed the max:

```
        uint128 liqPriceWithoutPenaltyWithoutFunding = Utils._getEffectivePriceForTick(Utils.calcTickWithoutPenalty
(data.action.tick, data.liquidationPenalty), + data.action.liqMultiplier +); // calculate the new total expo uint128
expoBefore = data.pos.totalExpo; • uint128 expoAfter = Utils._calcPositionTotalExpo(data.pos.amount, data.startPrice,
data.liqPriceWithoutPenalty); + uint128 expoAfter = Utils._calcPositionTotalExpo(data.pos.amount, data.startPrice,
liqPriceWithoutPenaltyWithoutFunding);
```

Resolution

Smardex Team: The issue was resolved in PR#628.

H-02 | Liquidation Rewards Sent To The Rebalancer Will Be Lost

Category	Severity	Location	Status
Unexpected Behavior	● High	Rebalancer.sol: 473	Resolved

Description

The Rebalancer contract implements the function initiateClosePosition which is responsible for closing a portion of the current Rebalancer position within the UsdnProtocol, based on a user's deposit.

This function makes an external call to _usdnProtocol.initiateClosePosition. During this external call, where the Rebalancer contract acts as the caller, one or more liquidity ticks may be liquidated. If liquidation occurs, the Rebalancer contract will receive wstETH tokens as a reward.

However, these wstETH tokens are not forwarded to the original caller of rebalancer.initiateClosePosition and instead remain trapped within the Rebalancer contract, where they will be permanently inaccessible.

Recommendation

Consider tracking the wstETH balance of the Rebalancer contract before and after the _usdnProtocol.initiateClosePosition call. If the wstETH balance was increased after the call, send the difference of wstETH tokens to the rebalancer.initiateClosePosition caller.

Resolution

Smardex Team: The issue was resolved in PR#633.

M-01 | Imbalance Does Not Count Funding

Category	Severity	Location	Status
Logical Error	● Medium	Global	Acknowledged

Description

While initiating a request to open a position the tradingExpo used to compute the new positions predicted exposure is based on a longTradingExpoWithFunding which includes the funding up until the current block.timestamp.

However the imbalance validations that follow, and the imbalance validations throughout the codebase use the current vault and long balances which ignore the funding accrued in the timeframe from the lastUpdateTime until the current block.timestamp.

Therefore there is an immediate contradiction in the imbalance validation during the initiation of an update request. The position exposure includes the latest funding, while the aggregate balances do not.

Furthermore, the imbalance validations for all actions cannot be accurately validated against the latest funding changes which have yet to be stored.

Recommendation

Consider accounting for the funding at the latest timestamp in the imbalance validation functions throughout the codebase. Furthermore, consider if the latest unrecorded funding should be taken into account when triggering the rebalancer.

Resolution

Smardex Team: Acknowledged.

M-02 | Positions Opened Above Max Leverage

Category	Severity	Location	Status
MEV	● Medium	UsdnProtocolActionsLibrary.sol: 260	Acknowledged

Description

When validating the creation of a position if the new leverage is over the `maxLeverage` then the position is adjusted to be at or within the `maxLeverage` bound. First, the `currentLiqPenalty` and the `liquidationPrice` corresponding to `maxLeverage` is used to select the liquidation tick for the position.

Then if the `liquidationPenalty` is different on the selected tick, that new penalty is applied to the `data.liqPriceWithoutPenalty` which is ultimately used to determine the leverage of the position.

However when the liquidation tick with penalty has a lower `liquidationPenalty` than the `currentLiqPenalty` in storage than the resulting `liqPriceWithoutPenalty` can be higher than the original `liqPriceWithoutPenalty` which was based on the `maxLeverage`.

`maxLeverage = startPrice / (startPrice - liqPriceWithoutPenalty0)`
Then, since `liqPriceWithoutPenalty0 < liqPriceWithoutPenalty1`:
`newLeverage = startPrice / (startPrice - liqPriceWithoutPenalty1) > maxLeverage`

This is unexpected for the protocol as no positions should have a leverage greater than the `maxLeverage`. Additionally, it is worth noting that the same can occur for the rebalancer position when determining the `liqPriceWithoutPenalty` in the `_calcRebalancerPositionTick` function.

Recommendation

Consider if it is acceptable to have positions which exceed the `maxLeverage`. The most straightforward solution would be to configure the `maxLeverage` accordingly to account for the fact that some positions may go slightly over it depending on the `liquidationPenalty` updates. Be sure to also keep this in mind when updating the `currentLiqPenalty` value in storage.

Resolution

Smardex Team: Acknowledged.

M-03 | Bad Debt Value Extraction

Category	Severity	Location	Status
Documentation	● Medium	UsdnProtocolLongLibrary.sol: 924	Resolved

Description PoC

When positions with bad debt are liquidated on a tick where the liquidation price without penalty is above the current price, then a correction is made to the balance of the vault.

The losses which are in excess of the position's collateral were errantly counted as going towards the vault. Upon liquidation these errant trader losses are clawed back from the vault.

Depositors may observe that this correction is going to occur if a liquidation round occurs and intentionally frontrun this to initiate a withdrawal.

The initiation of a withdrawal will protect the depositor from this clawing back of inaccurate trader losses because the `balanceVault` and `balanceLong` are cached on the withdrawal object.

Furthermore this clawing back of bad debt can cause a stepwise decrease in the price of USDN which may also be arbitrageable by shorting USDN.

Recommendation

Be aware of this risk of late liquidations and carefully consider it when configuring liquidation rewards and liquidation penalties.

Resolution

Smardex Team: The issue was resolved in PR#722.

M-04 | pendingBalanceVault Underflow

Category	Severity	Location	Status
Rounding	● Medium	Global	Resolved

Description

The `pendingBalanceVault` is deducted by the anticipated withdrawal amount when a withdrawal action is initiated.

However due to extreme price action in some edge cases the actual balance of the vault may fall below the pending balance, in which case this would cause an underflow revert in all areas where the pending vault balance is added to the vault balance.

This would effectively block the queue and shut down the protocol.

Recommendation

It may not be deemed too complex to handle this edge case in all locations for this iteration of the protocol. However be aware of this edge case and consider handling it.

Resolution

Smardex Team: The issue was resolved in PR#711.

M-05 | Initiators Avoid A Portion Of Funding

Category	Severity	Location	Status
Logical Error	● Medium	UsdnProtocolLongLibrary.sol: 279	Acknowledged

Description PoC

When initiating a position the user's exposure is computed accounting for the current funding rate accrued to the latest block.timestamp.

However upon initiation users can affect the funding directly after the initiation is complete in the following time range [initiationLastUpdateTimestamp, initiationBlockTimestamp] as long as a subsequent action is performed with another price before or near the initiationLastUpdateTimestamp.

This way users who initiate do not have to incur the cost of the existing funding rate over the range [initiationLastUpdateTimestamp, initiationBlockTimestamp], meanwhile their positions are affecting the skew and thus should incur the full funding cost.

These positions will incur the difference in the funding rate that they cause, but not the base funding rate that was pre-existing before they initiated their position.

If positions are not held accountable for the total funding rate in the range [initiateLatestPriceTimestamp, initiateBlockTimestamp], then they can force other long positions to pay for increased funding, by way of increasing the long exposure, while not paying for it themselves.

This may allow for extractable value for a USDN depositor or cause other positions to be liquidateable by this manipulation.

Recommendation

Consider only using the long exposure without syncing the funding to the latest timestamp, e.g. UsdnProtocolCoreLibrary.longTradingExpoWithFunding(s, lastPrice, uint128(s._lastUpdateTimestamp)). Or simply, s._totalExpo - s._longBalance.

Resolution

Smardex Team: Acknowledged.

M-06 | _triggerRebalance Does Not Account For Liquidation Rewards

Category	Severity	Location	Status
Unexpected Behavior	● Medium	UsdnProtocolLongLibrary.sol: 532	Acknowledged

Description

The _triggerRebalance function is responsible for initiating a rebalance when the imbalance on the long side becomes too significant, aiming to restore the protocol to a balanced state. It calculates the tick of the rebalancer position to open using the _calcRebalancerPositionTick function. However, this function relies on an outdated value of cache.vaultBalance, as the actual s._balanceVault state variable is reduced in the _sendRewardsToLiquidator function.

By ignoring the s._balanceVault decrease caused by the liquidation rewards, it is possible that after the rewards are paid out and removed from the s._balanceVault the protocol enters again an unbalanced state especially if the s._balanceVault's liquidity is low. Although the maximum liquidation rewards are currently capped at 0.5 Ether, and the protocol is expected to hold significantly more in its vault, this value could still be updated by a privileged account to a higher one.

Finally, the _usdnRebase function is also affected by this as it uses the s._balanceVault to calculate the USDN price. This calculation will not be accurate as the s._balanceVault has not been updated/decreased by the time _usdnRebase function is called.

Recommendation

Consider revising the calculation of liquidation rewards by dividing it into two components:

1. Fixed component: A fixed portion of the liquidation reward based on the number of ticks liquidated. This component should be slightly optimistic, providing a small excess to account for potential future events such as a triggerRebase or usdnRebase.
2. Variable component: A variable portion of the liquidation reward, calculated based on the remaining collateral after liquidation.

This way, the exact liquidation reward will be known beforehand and the _triggerRebalance and _usdnRebase functions can account for the exact s._balanceVault decrease caused by the liquidation rewards distribution.

Resolution

Smardex Team: Acknowledged.

M-07 | Slippage Check Occurs Before Adjustment

Category	Severity	Location	Status
Validation	● Medium	UsdnProtocolLongLibrary.sol: 273	Resolved

Description

In the _prepareInitiateOpenPositionData function the params.userMaxPrice is validated against the lastPrice, however the user's execution price will include a fee which is not included in the lastPrice.

On the following line the data_.adjustedPrice is assigned which is adjusted by this fee and more closely estimates the execution price that the user will experience.

Ideally the fee is included in the slippage validation since it will affect the user's ultimate execution price, and allows users to protect themselves if the _positionFeeBps were to be unexpectedly changed.

Recommendation

Consider validating the params.userMaxPrice against the adjustedPrice which includes the position fee.

Resolution

Smardex Team: The issue was resolved in PR#613.

M-08 | removeBlockedPendingAction Extractable Value

Category	Severity	Location	Status
MEV	● Medium	UsdnProtocolCoreLibrary.sol: 426	Resolved

Description

In the _removeBlockedPendingAction function when a ValidateClosePosition pending action is removed the entire closeBoundedPositionValue amount is added to the vault.

This means the trader loses their entire position value in the event that their close position action is stuck, even if they were nowhere near close to liquidation. This could cause significant loss for a trader which has a large position.

Furthermore, this creates a potentially large arbitrage opportunity. If a removal of a ValidateClosePosition pending action for a large position close is sitting in the mempool it would be potentially significantly profitable for an actor to front-run this transaction and initiate a deposit into the vault right before the removal takes place.

Because vault deposits only vary based on price action between initiation and validation the malicious actor would realize their corresponding value of the immediate vault balance increase from the removal of the pending close position action.

Recommendation

Instead of giving the entire position value to the vault upon removal of a close pending action, consider sending the position value to the specified to address. The protocol can then manually decide how much should be refunded to the trader versus donated to the vault via a new donate function.

Resolution

Smardex Team: The issue was resolved in PR#672.

M-09 | Tick Liquidation Penalty Cannot Be Reset

Category	Severity	Location	Status
Unexpected Behavior	● Medium	UsdnProtocolLongLibrary.sol: 332	Resolved

Description

In the _removeAmountFromPosition function the liquidationPenalty is not reset on the tick data after all positions have been removed.

Additionally, during the initiate open flow, the existing tick's liquidation penalty as the result of getTickLiquidationPenalty(s, data_.posId.tick) is used to re-assign the new liquidation penalty.

Thus even after all positions have been cleared from the tick, the latest configured liquidation penalty cannot be assigned to this tick.

Recommendation

Reset the tick's liquidationPenalty to zero after all positions have been removed from the tick in the _removeAmountFromPosition function.

Resolution

Smardex Team: The issue was resolved in PR#615.

M-10 | Incorrect pendingBalanceVault Correction

Category	Severity	Location	Status
Logical Error	● Medium	UsdnProtocolCoreLibrary.sol: 374	Resolved

Description

In the _removeBlockedPendingAction function the _pendingBalanceVault is corrected by computing the withdrawal amount without accounting for fees.

However upon the initiation of the withdrawal the _pendingBalanceVault was decremented by the withdrawal amount less fees.

As a result when a withdrawal action is removed with the _removeBlockedPendingAction function using cleanup as true the _pendingBalanceVault experiences an invalid net increase by the withdrawal fee amount.

Recommendation

Correct the _pendingBalanceVault by the withdrawal amount accounting for fees in the _removeBlockedPendingAction function.

Resolution

Smardex Team: The issue was resolved in PR#657.

M-11 | _calcPositionSizeBonus Errant Token Amount

Category	Severity	Location	Status
Logical Error	● Medium	LiquidationRewardsManager.sol: 103	Resolved

Description

The position size bonus for liquidator remuneration is computed with the `_calcPositionSizeBonus` function.

The bonus is based upon a difference in the current price and the liquidated ticks price and the documentation for the `_calcPositionSizeBonus` indicates that the resulting value is denominated in native ether.

However when the asset is wstEth and the `WstEthOracleMiddleware` is used the price will be for wstEth and thus the `_calcPositionSizeBonus` function will return a wstEth amount.

However this returned value is treated as native ether and converted to wstEth redundantly on line 108: `wstETHRewards_ = _wstEth.getWstETHByStETH(totalRewardETH);`

Recommendation

Instead of converting the position size bonus amount to wstEth with the `getWstETHByStETH(totalRewardETH)` call, add it to the resulting wstEth value. If this approach is taken be sure to cap the `maxReward` to the resulting wstEth amount and assign this number as a wstEth value.

Resolution

Smardex Team: The issue was resolved in PR#632.

M-12 | Inaccurate Imbalance Check In _checkImbalanceLimitOpen

Category	Severity	Location	Status
Validation	● Medium	UsdnProtocolLongLibrary.sol: 314	Resolved

Description

The function `_checkImbalanceLimitOpen` inaccurately calculates the imbalance when initiating the opening of a new position because it does not account for the position fees deducted. This miscalculation could allow the system to enter a state where it is unbalanced beyond what should be possible after the action is validated.

This is because `_checkImbalanceLimitOpen` calculates the `imbalanceBps` by assuming that the balance of the vault is equal to just `s._balanceVault + s._pendingBalanceVault` and that the balance of the long side is equal to `s._balanceLong + openCollatValue` where `openCollatValue` is equal to `params.amount`.

However, `openCollatValue` does not reflect the true position value and should be instead equal to `data_.positionValue` (`params.amount` minus the position fee paid). On the other hand, `currentVaultExpo` should also include the position fee which can be calculated as `params.amount - data_.positionValue`. Do notice that this update is performed after `_checkImbalanceLimitOpen` check.

Recommendation

Consider updating the imbalance check to include the fees.

Resolution

Smardex Team: The issue was resolved in PR#616.

M-13 | Exposure During Closure Unaccounted

Category	Severity	Location	Status
Logical Error	● Medium	Global	Acknowledged

Description

During the initiation of a close action, the position, or the portion of a position being removed, is effectively removed from the protocol as it is deducted from it’s corresponding tick and the cumulative balance and exposure accounting. However the USDN vault still serves as a counterparty for any PnL realized by the position in the period: [initiateClose, validateClose] and the user is still exposed to this price action as well. This may present an issue as the system will technically be more exposed to the long side than the imbalance tracking accounts for. As a result the corresponding imbalance mechanisms may not be used when they ought to.

Extracting value from a vault withdrawal (assuming 0 protocol fees):
1. Bob initiates close position. Bob provided empty Pyth data so Pyth.getPriceUnsafe() is called retrieving the price of Ether 59 minutes ago (price manually updated on-chain by Bob 59 minutes ago). The price retrieved is 1750 even though the current Ether price is 59 minutes later/now 1925. 14 minutes passes and Bob has not validated yet.
2. Alice initiates a withdrawal providing the actual current Pyth price of (1925 Ether price). We are assuming a 10% increment of Ether price in 1 hour 14 minutes.
3. Bob validates its position closure, and balanceVault is decreased based on the size of the long position closed and on the 10% price increase.
4. Alice validates withdrawal, but she still gets the assets respective to the vaultBalance at the time of the initiation. As the vaultBalance was higher, Alice UNFAIRLY gets more assets than she deserves. As at minute 60 + 14 the position value of Bob was already determined (it was already determined at minute 60 + 24 seconds) but Bob decided to delay his validation and because of this Alice received a higher amount of assets.

```
assetsReceivedWithExploit .................................. 1821845730818816586
assetsReceivedWithoutExploit ............................... 1813360204002626022
Increment of ............................................... 46794/10000000 (0.47%)
Value of the position closed compared to total long value ... 2%
Price change ............................................... 10% increment of Ether price
```

Extracting value from a vault deposit (assuming 0 protocol fees):
1. Bob initiates close position. Bob provided empty Pyth data so Pyth.getPriceUnsafe() is called retrieving the price of Ether 59 minutes ago (price manually updated onchain by Bob 59 minutes ago). The price retrieved is 1750 even though the current Ether price is 59 minutes later/now 1575. 14 minutes passes and Bob has not validated yet.
2. Alice initiates a deposit providing the actual current Pyth price of (1575 Ether price). We are assuming a 10% decrement of Ether price in 1 hour 14 minutes.
3. Bob validates its position closure, and balanceVault is increased based on the size of the long position closed and on the 10% price decrease.
4. Alice validates her deposit, but she still gets the shares respective to the vaultBalance at the time of the initiation. As the vaultBalance was lower, Alice unfairly gets more shares than she deserves. As at minute 60 + 14 the position value of Bob was already determined (it was already determined at minute 60 + 24 seconds) but Bob decided to delay his validation and because of this Alice received a higher amount of shares.

```
sharesReceivedWithExploit ..................................
36213654980821231944568756540902094568O2
sharesReceivedWithoutExploit ...............................
36047534243868681756283408010336483147777
Increment of ............................................... 45872/10000000 (0.46%)
Value of the position closed compared to total long value ... 2%
Price change ............................................... 10% decrement of Ether price
```

Recommendation

Consider introducing specific accounting in the imbalance calculations for the exposure of all position amounts which have been effectively closed after initiation, but have yet to be validated and have their exposure officially removed. If this approach is taken, then one inconsistency should be addressed. When an individual liquidation occurs during a close action validation, the position is valued at the block in which the action was initiated, however this contradicts the logic that follows for a normal close, where the vault is exposed to the current value of the position.

These two scenarios should be standardized in terms of exposure. If the first suggestion is implemented, the liquidations should be adjusted to also have exposure to the latest asset price. Alternatively, the exposure over the period [initiateClose, validateClose] could be removed entirely, and the exact position value to be realized could be computed during the initiation.

Resolution

Smardex Team: Acknowledged.

M-14 | Innacurate isLiquidationPending Check In ValidateOpen

Category	Severity	Location	Status
Validation	● Medium	UsdnProtocolActionsLongLibrary.sol: 612	Resolved

Description

The _prepareValidateOpenPositionData function implements the following check:

```
data_.liqPriceWithoutPenalty = Utils.getEffectivePriceForTick (s, Utils.calcTickWithoutPenalty(data_.action.tick,
data_.liquidationPenalty)); data_.lastPrice = s._lastPrice;

if (data_.lastPrice < data_.liqPriceWithoutPenalty) {// the position must be liquidated data_.isLiquidationPending =
true; return (data_, false);}
```

This check compares the s._lastPrice to the liquidation price without the penalty. If the s._lastPrice is lower, the function marks the position as liquidatable and halts further validation. However, this check is not accurate because the liquidation price should factor in the liquidation penalty.

By only comparing against the liquidation price without the penalty, when there is a lastPrice between the liqPriceWithoutPenalty and the actual liquidation price (including the penalty), position is not being flagged for liquidation, even though it should be.

Recommendation

To fix this, the liquidation check should compare the s._lastPrice against the liquidation price with the penalty. The updated check should look like this:

```
uint256 liqPrice = Utils.getEffectivePriceForTick(s, data_.posId.tick); if (data_.lastPrice < liqPrice) {// the
position must be liquidated data_.isLiquidationPending = true; return (data_, false);}
```

Resolution

Smardex Team: The issue was resolved in PR#631.

M-15 | Fee Change Hurts Traders

Category	Severity	Location	Status
Logical Error	● Medium	UsdnProtocolActionsLongLibrary.sol: 572	Acknowledged

Description

In the _prepareValidateOpenPositionData function the startPrice is affected by the current fee in storage, however this fee may be different than the one that was applied upon initiation and validated against the userMaxPrice.

This is in contrast to the deposit and withdrawal flow, where the vaultFeeBps are cached onto the deposit/withdrawal object and used during validation.

Recommendation

Consider caching the _positionFeeBps value upon initiation in the open position object to be applied on validation.

Resolution

Smardex Team: Acknowledged.

M-16 | Min Price Validation Excludes Fee

Category	Severity	Location	Status
Validation	● Medium	UsdnProtocolActionsUtilsLibrary.sol: 170	Resolved

Description

In the _prepareClosePositionData function the userMinPrice is validated against the lastPrice, however the price applied to the value of the position and thus the amount received is reduced by the position _positionFeeBps.

Thus the resulting price experienced by the user upon closing their position can often be less than the minimum desired without any unpredictable price action occurring between the initiation and validation of a close.

The position fees are easy to predict ahead of time, and thus should be included in the userMinPrice validation.

Recommendation

Consider reducing the lastPrice by the _positionFeeBps when validating the userMinPrice in the _prepareClosePositionData function.

Resolution

Smardex Team: The issue was resolved in PR#630.

M-17 | Risk Free Trades With Rebalancer

Category	Severity	Location	Status
Gaming	● Medium	Rebalancer.sol	Resolved

Description PoC

The Rebalancer is allowed to open positions at the current lastPrice immediately, without going through the initiation and validation process. This opens up the opportunity for risk free trades for the users who deposit into the rebalancer.

There there are several mechanisms in the Rebalancer to prevent the gameability of a risk free trade, but it is still possible to extract risk free value from the Rebalancer mechanism. The goal of an actor in this exploit is to get their deposit included in the rebalancer position to take advantage of an outdated lastPrice upon the triggering of the rebalancer.

A malicious actor can see when a large position or a large set of positions are close to being liquidated and initiate a deposit into the rebalancer. If the rebalancer is able to be triggered with a lastPrice that is less than the current market price within the [initiateRebalancerDeposit, initiateRebalancerDeposit + 24 seconds] window, then the user can immediately game the stale pricing by:

- Validating their rebalancer deposit
- triggering the rebalancer via a liquidation call
- Exiting the rebalancer with the initiateClosePosition function

If the correct conditions are not met within the timeframe, the actor can simply choose to not validate their deposit and wait until the cooldown period has ended to collect their funds. The actor can open consecutive initiate deposits with multiple addresses to ensure that they are able to take advantage of a rebalancer triggering in a given timeframe.

Recommendation

The extractable value from this grows with the size being liquidated and the imbalance created, however it is unlikely to occur with a great magnitude consistently. Therefore it may be fine to acknowledge this extractable value. Otherwise consider taking further measures to reduce the feasibility of this value extraction.

Such as introducing a fee upon validation or cancellation of a rebalancer deposit, or allowing other users to validate an arbitrary user's deposit so they do not have guaranteed optionality over the execution of their deposit.

Resolution

Smardex Team: The issue was resolved in PR#677.

M-18 | No Rebalancer Trigger On Individual Liquidation

Category	Severity	Location	Status
Unexpected Behavior	● Medium	UsdnProtocolActionsLongLIbrary.sol: 806	Acknowledged

Description

When closing a position the position has already been removed from the tick cumulatives as well as the exposure and balance of the aggregate long tracking.

When the close action is validated the vault balance is then incremented by the entire position value to account for this position being liquidated.

However since this liquidation logic is separated from the tick cumulative liquidation the rebalancer position will not be triggered when the individual position liquidation would be the one to set the net imbalance over the `_closeExpoImbalanceLimitBps`.

As a result the liquidation of a single large position upon the validation of it's close action could adversely affect the balance of exposures without a counter-action from the rebalancer position.

Recommendation

Consider if this behavior and corresponding risk of imbalance without counter-action from the rebalancer position is acceptable.

If it is not, consider triggering the rebalancer after the individual position liquidation occurs in the `_validateClosePositionWithAction` function.

Resolution

Smardex Team: Acknowledged.

M-19 | Init Close Of A Liquidatable Pos Permitted

Category	Severity	Location	Status
Validation	● Medium	UsdnProtocolActionsUtilsLibrary.sol: 174-179	Resolved

Description

When a recent price is given to any init or validate function the _applyPnlAndFundingAndLiquidate flow is executed and if too many ticks are liquidatable, some are left over and the flow breaks early as liquidations are pending.

In this case, a user can initiate a close of a position that sits on such a liquidatable tick that was not liquidated yet by providing a price that is less recent than the current lastPrice and therefore skipping the _applyPnlAndFundingAndLiquidate flow.

As the init close pos function only checks if the version of the positions tick changed but not if the lastPrice is above its liquidation price the call will pass.

Recommendation

Consider not allowing any actions to be initiated or validated in the event that any liquidations are pending at the lastPrice, even if the currentPrice timestamp is not recent.

Resolution

Smardex Team: The issue was resolved in PR#722.

M-20 | Increased Liquidation Rewards

Category	Severity	Location	Status
Gaming	● Medium	LiquidationRewardsManager.sol: 104	Acknowledged

Description

Positions will be liquidated once the price drops below the current tick price. Liquidating multiple ticks with high expo, or with a big liquidation bonus (due to where the current price is at, compared to the tick liquidation price), will grant a higher liquidation reward. However, the total ETH reward calculated should not exceed 0.5 ETH.

Therefore, it can be more profitable for a user to liquidate ticks one by one, through the `initiate` and `validate` functions (functions that will try to perform one iteration of liquidation if there are liquidatable ticks) ,instead of using the `liquidate` function that contains a capped liquidation reward.

Recommendation

Cap the maximum rewards that can be gained from liquidations that are happening via initiate or validate functions if there are still pending liquidations.

These amounts should be capped such that the maximum reward that can be gained from individual liquidation should match with the max reward amount of multiple liquidations using `liquidate`.

It might be also good to consider limiting rewards that can be gained from liquidations in a single block. This can also incentivize the usage of `liquidate`.

Resolution

Smardex Team: Acknowledged.

M-21 | Extracting From Rebalancer Bonus

Category	Severity	Location	Status
Gaming	● Medium	Rebalancer.sol	Resolved

Description PoC

Rebalance mechanism is crucial for holding the protocol in balance, hence the users are incentivized to deposit funds to rebalancer (no fee for position opening, 80% of the remaining collateral from the liquidations will be distributed etc.)

However this mechanism can be gamed such that a user can sandwich a liquidation call to extract value from rebalancer bonus without a need for locking value in rebalancer more than a couple minutes.

Here are the steps to perform the attack:

1- Deposit into the rebalancer near liquidation and into the vault to increase imbalance.
2- Frontrun any user's liquidation attempt and perform the following in one transaction:
2.1- Validate rebalancer deposit
2.2- Liquidate (Which will create a rebalancer position)
2.3- Initiate a withdrawal from vault so that it will be possible to withdraw from rebalancer.
2.4- Initiate rebalancer close.
3- Wait 24 seconds (delay) and validate the withdrawal from vault and rebalancer close.

Result: Profit from bonus in rebalancer + pnl from trades in vault.

Recommendation

Prevent position closing from the rebalancer for some time after deposits so that there won't be a risk-free profit opportunity with the specified attack anymore.

Resolution

Smardex Team: The issue was resolved in PR#677.

M-22 | Depositors Banned From Rebalancer If Liquidated

Category	Severity	Location	Status
DoS	● Medium	Rebalancer.sol: 245	Resolved

Description PoC

A rebalancer position is created when liquidations create an imbalance in the protocol. Although the rebalancer position liquidation price is low (it's configured to have max 3x leverage), there could be cases when this position might get liquidated.

If there are no new depositors or the imbalance is not big enough, this liquidation will not open a new rebalancer position. Users that were participating in the rebalancer position that got liquidated will now be temporarily banned from depositing into the rebalancer again.

This is due to the fact that the user's `entryPositionVersion` is greater than the `_lastLiquidatedVersion`.

Recommendation

If the rebalancer is liquidated, notify the rebalancer contract by executing `updatePosition` before any early return.

Resolution

Smardex Team: The issue was resolved in PR#642.

M-23 | Blocked Queue Due To Non-Validatable Action

Category	Severity	Location	Status
DoS	● Medium	UsdnProtocolActionsLongLibrary.sol: 619	Resolved

Description PoC

Users should be able to validate open positions as long as the price is above their liquidation price. In case of sudden price drops, user's validation price could be below their liquidation price, causing a revert on the leverage calculation.

As the validation price is a fixed Pyth update price during a certain time interval, it should always use the same exact update data. Once this blocked action becomes `actionable`, any user creating a new action should also pass a `previousActionsData` param, to validate the next actionable action in queue.

Therefore, the blocked action will now DoS new protocol actions for 5 minutes, from `lowLatencyValidatorDeadline` to `lowLatencyDelay`. The admins will need to wait `lowLatencyValidatorDeadline + 1 hours` in order to unblock this action and the user will lose the security deposit.

Recommendation

Early return if the validation price is below the liquidation price, to signal a liquidatable state: `data_.isLiquidationPending = true;`

Resolution

Smardex Team: The issue was resolved in PR#639.

M-24 | Bad Debt Not Handled In Validate Withdraw

Category	Severity	Location	Status
DoS	● Medium	UsdnProtocolVaultLibrary.sol: 1063	Resolved

Description

Validate Withdraw decreases the balance of the vault by the calculated `assetToTransferAfterFees` amt and does not cap it at zero. Therefore if this amt is bigger than the `_balanceVault` (for example as bad debt was taken between init and validate) an underflow occurs which leads to a long term DoS as this is a validation function.

Recommendation

Handle bad debt in the validate withdraw function. For example, this can be done by capping `assetToTransferAfterFees` to `s._balanceVault` to avoid an underflow.

Resolution

Smardex Team: The issue was resolved in PR#674.

M-25 | Closed Amt Not Seized If Its Value Is < 0

Category	Severity	Location	Status
Validation	● Medium	UsdnProtocolActionsLongLibrary.sol: 796-828	Resolved

Description

The _validateClosePositionWithAction function checks if the position is liquidatable and if so seizes it's closeBoundedPositionValue and gives it the vault.

This check happens with the neutral price and later on, it calculates the value of the position with the price rounded down by the pyth interval and reduced by the position fee.

Therefore the value of the position could be < 0, and in that case the closeBoundedPositionValue is not seized and nothing happens as the rest of the function is only executed if (data.positionValue > 0). In this case, the closeBoundedPositionValue is stuck in the system.

Recommendation

Seize the closeBoundedPositionValue if the position value is <= 0.

Resolution

Smardex Team: The issue was resolved in PR#679.

M-26 | User Can't Close Rebalanced Position

Category	Severity	Location	Status
Validation	● Medium	UsdnProtocolActionsUtilsLibrary.sol	Resolved

Description

The protocol has specific logic to ensure that any user can fully close their position if the position is apart of the rebalancer. The logic is intended to ignore the min long position amount if the user is coming from the rebalancer and the close amount is the full amount from that user.

However, the check fails to take into account any change in the users position value. So when the rebalancer is in profit the close amount will be greater than the position amount used in the check and visa versa for when the rebalancer is at a loss.

Because of this users will not be able to close their position if any pnl is experienced and there is only a small amount of assets in the rebalancer. Given that the rebalancer is a long position that can face liquidation having only a small amount in it is a real possibility.

Recommendation

When closing a position consider checking if the user is making a full close and if so, let the user bypass this check.

Resolution

Smardex Team: The issue was resolved in https://github.com/SmarDex-Ecosystem/usdn-contracts/pull/640/files#diff-fc903c7d38e9e68316fb4e9d65de2ea0994d47bcfe2a6716b01ce0d29018ab01L408-L418 and https://github.com/SmarDex-Ecosystem/usdn-contracts/pull/640/files#diff-cdb6008d48b1cd54009dc3ab20e997eda7e97f4969eddd5af8fad7553031027cR738-R748.

M-27 | Liquidatable Positions Can Be Opened

Category	Severity	Location	Status
Validation	● Medium	UsdnProtocolActionsLongLibrary.sol: 233-241	Acknowledged

Description

In the validate open position flow if the maximum leverage is exceeded with the new `startPrice` the position and it's tick is recalculated. This new tick could be liquidatable in edge cases if the price between init and validate changed drastically but the position is still created.

The primary instance where this becomes a problem is when the liquidation tick actually increases when modified. In these situations a once healthy position can become liquidateable resulting in prior liquidation safeguards not being triggered and allowing liquidatable positions to be opened.

Recommendation

After modifying the positions leverage check if the position is liquidatable and if so, liquidate the position.

Resolution

Smardex Team: Acknowledged.

M-28 | Validate Deposit Ignores Funding

Category	Severity	Location	Status
Logical Error	● Medium	UsdnProtocolVaultLibrary.sol: 737	Resolved

Description

The `_validateDepositWithAction` function uses either the available vault balance at init for the share price calculation or applies the PnL of the current price on it, whichever is bigger, but does not take funding into account.

Therefore, if the longs paid funding fees to the vault between initiation and validation of a deposit the user receives more assets than the user should and the other way around.

Recommendation

Consider incorporating the latest vault funding into how many assets the users receive. Otherwise, if this is acceptable to the protocol, be sure to document the behavior so that users are aware.

Resolution

Smardex Team: Resolved.

M-29 | Sandwich Liquidations

Category	Severity	Location	Status
MEV	● Medium	Global	Acknowledged

Description

When liquidations happen a stepwise jump in the value of shares happens. Either the liquidated value will increase the share value or the liquidation reward and/or bad debt will decrease it.

As the init call of deposit/withdraw saves the current balances and calculates the share/asset amt received based on that, users are able to sandwich liquidations to make profit or avoid losses.
Deposit for profit:

• LP sees that a liquidation call will increase the vaults balance
• LP front runs the transaction and initiates a deposit to mint shares based on the old balance
• The liquidation call goes through
• The LP validates the deposit and is in instant profit without price changes
Withdraw to avoid loss:
• LP sees that a liquidation call will decrease the vaults balance (bad debt)
• LP front runs the transaction and initiates a withdraw to burn shares based on the old balance
• The liquidation call goes through
• The LP validates the withdraw and avoided paying for the bad debt and socialized more loss to the other LPs by doing so

Recommendation

This finding serves only to document this behavior. Be aware of these potentially unexpected behaviors and how they could be manipulated.

Resolution

Smardex Team: Acknowledged.

M-30 | Validate Withdrawal Ignores Funding

Category	Severity	Location	Status
Logical Error	● Medium	UsdnProtocolVaultLibrary.sol: 993	Acknowledged

Description

The _validateWithdrawalWithAction function uses either the available vault balance at init for the share price calculation or applies the PnL of the current price on it, whichever is less, but does not take funding into account.

Therefore if the vault paid funding fees to the longs between initiation and validation of a withdrawal the user receives more assets than the user should.

In certain edge cases this can also lead to a underflow DoS of the _validateWithdrawalWithAction function as this calculated withdrawal amount is later decreased from the _balanceVault where funding was applied to.

Recommendation

Consider incorporating the latest vault funding into how many assets the users receives. Otherwise if this is acceptable to the protocol, be sure to document the behavior so that users are aware.

Resolution

Smardex Team: Acknowledged.

M-31 | Remove Pending Position Price Can Be Stale

Category	Severity	Location	Status
Logical Error	● Medium	_removeBlockedPendingAction	Acknowledged

Description

The _removeBlockedPendingAction uses the _lastPrice to calculate the value of the stuck open long position. As this function does not update the price before performing this action it can potentially be stale.

Recommendation

Fetch the current price at the beginning of the _removeBlockedPendingAction function.

Resolution

Smardex Team: Acknowledged.

M-32 | Neutral Price Used In Init Functions

Category	Severity	Location	Status
Validation	● Medium	Global	Acknowledged

Description

All init functions use the lastPrice (the latest neutral price) for calculations, while the validation functions use prices that were adjusted by the pyth interval up or down to round against the user.

Therefore all the checks and temporary state updates at init are most likely wrong at validation time.
Here are a few examples:
• Slippage checks
• Imbalance checks
• The temporary position between init and validate open position
• SDEX calculations
• When a stuck open position action is removed by the admin the user receives the position value based on a unadjusted start price

These examples will lead to users entering a position at a price they explicitly did not agree too, Protocol reaching an imbalanced state and Incorrect amount of SDEX being burned.

Recommendation

Use the adjusted price in the init functions instead of the neutral price if the calculation uses the adjusted price in the validation function.

Resolution

Smardex Team: Acknowledged.

M-33 | Funding Rate Affected By Updates

Category	Severity	Location	Status
Logical Error	● Medium	UsdnProtocolCoreLibrary.sol: 728, 737	Acknowledged

Description

In the _fundingPerDay function the resulting _fundingPerDay value is a function of the current imbalance added to the existing EMA value. This resulting _fundingPerDay value is then factored into the EMA.

The additive nature of the current skew to the current EMA for the resulting _fundingPerDay value means that the more updates occur in a given timeframe the higher the funding will be.

In the attached PoC shows that in a week period the funding is 34% greater if there is an update every day versus if there are only updates at the beginning and end of the period.

This is unexpected as the amount of updates should not affect the funding amount paid or the funding rate and instead this should be based purely on the skew experienced and the time of imbalance.

Recommendation

Consider refactoring the funding calculations such that the fundingPerDay portion that is based upon the current imbalance as represented by numerator^2 * _fundingSF / denominator^2 is simply factored into the EMA as the latest data point instead of adding it to the EMA for the resulting _fundingPerDay value.

The latest EMA computed this way including the most recent skew calculation can be used to compute the resulting funding value.

Resolution

Smardex Team: Acknowledged.

M-34 | Old Vault Validations Swing USDN Price

Category	Severity	Location	Status
Gaming	● Medium	Global	Acknowledged

Description PoC

The behavior of the deposit flow is such that a user's shares are not minted and their deposited amount is not added to the vault balance until validation time. This however causes several issues related to funding and the price of USDN. The minting of shares is an activity that changes the price of shares when the ratio used to mint is stale.

For example:

• validationPrice is $80 from 30 minutes ago (initiation price is the same)
• Current market price is $100
• Vault balance is 100 wstEth at the current market price
• Vault balance is 120 wstEth at the validationPrice
• Deposit is for 10 wstEth
• There are 10,000 total vault shares
• Divisor is 1
• USDN price is 100 wstEth * $100 / 10,000 shares = $1
• If the deposit validation were to occur at the current market price the user would receive 10 * 10,000 / 100 = 1,000 shares
• USDN price at the current market price would be: 110 wstEth * $100 / 11,000 shares = $1
• However the deposit validation takes place at the validation price, which is the most recent price after the initiation + 24 seconds.
• Therefore the user receives shares: 10 * 10,000 / 80 = 1,250 shares
• USDN price at the current market price is now: 110 wstEth * $100 / 11,250 shares = $0.9777

Thus the price of USDN experiences a stepwise change which can be non-trivial, a similar issue exists with withdrawals as well. Furthermore, there are other less obvious issues regarding funding with the current deposit accounting. Firstly, depositors are forced to be held accountable for the funding that occurs in the timeframe [initiationTimestamp, validationTimestamp] even though their deposited amounts are not added to the vaultBalance until validationTimestamp and thus have not affected the skew for this period.

Secondly, depositors are forced to be held accountable for the difference in funding which occurs over the [initiationLastPriceTimestamp, initiationTimestamp] period versus the predicted amount of funding which would occur in that timeframe which is computed here: https://github.com/GuardianAudits/usdn-1/blob/7462fc4d1b886129a4091dfc91ff5304d1d3a970/src/UsdnProtocol/libraries/UsdnProtocolVaultLibrary.sol#L576.

Recommendation

Use the same approach as for the open position actions for the deposit actions. Mint the shares up front and let the deposited amount directly be added to the vaultBalance, but do not give the shares to the user yet. Upon validation issue a correction to the shares received by the user based upon the price difference between initiation and validation.

This will result in a stepwise jump in share price similar to the one experienced when using the _validateOpenPositionUpdateBalances function during the open position flow, however this stepwise jump will be far more insignificant than the one experienced currently due to old deposit validations.

The same approach should be taken for vault withdrawals, similar to the closing of positions. Another solution would be to simply require all prices to be much more recent than the current configurations which would reduce the potential worse case magnitude of the stepwise jump described in this finding.

Resolution

Smardex Team: Acknowledged.

L-01 | Fee Amounts Use Round Down Division

Category	Severity	Location	Status
Rounding	● Low	Global	Acknowledged

Description

Throughout the codebase rounding occurs in favor of the user when instead these operations should round in favor of the protocol. For instance, the fee charged to the user is rounded down in the `_prepareInitiateDepositData` and `_validateDepositWithAction` functions.

Recommendation

Throughout the codebase and particularly in those areas mentioned, be sure to round in favor of the protocol instead of the user.

Resolution

Smardex Team: Acknowledged.

L-02 | Incorrect Action Id Used

Category	Severity	Location	Status
Logical Error	● Low	UsdnProtocolActionsUtilsLibrary.sol: 155	Resolved

Description

In the `_prepareClosePositionData` function when fetching the oracle price the actionId is calculated based on the `params.owner` instead of the validator. This is invalid as the action id is intended to be based on the validator and the owner may not be the validator of the close position action.

Recommendation

Use the `params.validator` for the action id in the `_prepareClosePositionData` function.

Resolution

Smardex Team: The issue was resolved in PR#files#diff-fc903c7d38e9e68316fb4e9d65de2ea0994d47bcfe2a6716b01ce0d29018ab01L155.

L-03 | Risk Of Immediately Liquidatable Positions

Category	Severity	Location	Status
DoS	● Low	UsdnProtocolLongLibrary.sol: 300	Acknowledged

Description

During action initiations if pyth data is not provided, the price used for the initiation can be up to the `_timeElapsedLimit` old. This value is not configured in the `initializeStorage` function, however in tests it is indicated that it may be on the order of an hour.

This means it is likely that prices used on initiation may be inaccurate up to the tolerance of the on chain oracle. This should be carefully considered when configuring the `_safetyMarginBps` as there is an increased risk of immediately liquidatable positions when the lastPrice is stale up to the `_timeElapsedLimit`.

Recommendation

The current configuration of 2% for the `_safetyMarginBps` is relatively safe as the deviation threshold for the ETH/USD Chainlink on-chain feed is 0.50%. However this risk should be carefully considered when configuring the `_safetyMarginBps` or using another underlying asset.

Resolution

Smardex Team: Acknowledged.

L-04 | Rebalancer Depositors Might Not Receive Any Bonus

Category	Severity	Location	Status
Validation	● Low	UsdnProtocolLongLibrary.sol: 588	Acknowledged

Description

In the current implementation, the only additional incentive for rebalancer depositors is the potential bonus. This bonus is calculated as a percentage of `s._rebalancerBonusBps` (set at 80%) from the `remainingCollateral` after a liquidation:

```
     calculate the bonus now and update the cache to make sure removing it from the
vault doesn't push the // imbalance above the threshold uint128 bonus;

     if (remainingCollateral > 0) {bonus = (uint256(remainingCollateral) *
s._rebalancerBonusBps / Constants.BPS_DIVISOR).toUint128(); cache.vaultBalance -=
bonus;}
```

If the `remainingCollateral` is below zero (resulting in bad debt), rebalancer depositors will not receive any bonus. Additionally, the bonus is split among all depositors.

Therefore, if the total deposit amount in the rebalancer is large, the bonus received by each depositor will be negligible in comparison to their deposit.

Recommendation

To enhance depositor incentives, consider retaining a small percentage of each liquidation that does not result in bad debt and use this to increase a cumulative bonus state variable.

Once the rebalancer is triggered, the accumulated bonus from this state variable can be distributed to the depositors. After the bonus distribution, reset the bonus state variable to zero for the next cycle.

Resolution

Smardex Team: Acknowledged.

L-05 | Lacking Configuration Validations

Category	Severity	Location	Status
Validation	● Low	LiquidationRewardsManagerModule.sol: 117	Resolved

Description

In the `setRewardsParameters` function there are validations such that the gas usage values cannot be assigned above a set maximum. However there are no maximum bounds for the `baseFeeOffset`, `gasMultiplierBps`, `positionBonusMultiplierBps`, `fixedReward`, and `maxReward` values.

Recommendation

Consider if any of these values should be validated against a maximum and implement these validations as necessary.

Resolution

Smardex Team: The issue was resolved in PR#658.

L-06 | Innacurate wstETH Price Calculation

Category	Severity	Location	Status
Unexpected Behavior	● Low	WstEthOracleMiddleware.sol: 54	Acknowledged

Description

The WstEthOracleMiddleware contract implements the function parseAndValidatePrice which fetches the ethPrice from Pyth/Chainlink and then calculates the wstETH price by multiplying it by the wstEth.stEthPerToken. However, the ethPrice could reflect a previous timestamp, while _wstEth.stEthPerToken reflects the amount of stETH per wstETH at the current block or timestamp. Typically, this discrepancy does not lead to significant issues because wstEth.stEthPerToken updates only once per day, but if a rebase occurs between the ethPrice.timestamp and the current block.timestamp, the wstETH price calculated would be inaccurate. In the Lido protocol, stETH rebases occur daily through reports from the Lido Accounting Oracle.

The oracle submits data to the Ethereum network each day at approximately 12 PM UTC. These updates adjust stETH balances based on validator rewards, causing daily balance rebases. The stETH token accrues rewards automatically, which are distributed across all holders in the form of daily balance adjustments. The rebase works by increasing the stEthPerToken ratio (the amount of stETH that represents 1 ETH) as staking rewards accumulate. The stEthPerToken ratio is adjusted based on the total pooled ETH and the total number of stETH tokens.

The oracle calculates this based on data retrieved from the Ethereum consensus layer. If validators earn rewards, this ratio increases, reflecting the growth in staked ETH value. The formula looks like this: stEthPerToken = totalPooledEther / totalstETHShares. In case of validator penalties or slashing events, the daily rebase could be negative, reducing the value of stETH proportionally. This is managed through the same oracle mechanism, which reports any losses to the Ethereum network. The Lido protocol could enter in Bunker mode if significant penalties occur, in which case special handling rules apply to mitigate further losses and protect the network.

In the last 2 submitted reports from the Lido Accounting Oracle:

• stEthPerToken went from 1181096503086228134 to 1181205118477305019 increase of 0.0092%
• stEthPerToken went from 1181205118477305019 to 1181316084034843739 increase of 0.0094%

On the other hand, the Pyth price is always rounded against the user by using a percentage of its confidence interval (_confRatioBps), which is initially set to the 40%. If we take a look at the latest Pyth data submitted on-chain for the ETH/USD price feed we get:

• Price: 2380.87555000
• Confidence interval: 2.16445000
• Exponent: -8
• Publish time: 1728568800

Calculating the 40% of the confidence interval (2.16445000) results in 0.86578000. This confidence interval represents approximately the 0.0364% of the price, which is larger than the observed stEthPerToken increase of around 0.01%. Based on our analysis, considering the current stETH APY and the small daily increases in the stEthPerToken, the risk of price manipulation, such as performing operations right before the daily rebase, is minimal. However, if staking rewards increase significantly or if rewards are distributed less frequently (e.g., monthly), leading to larger stEthPerToken increases, the risk of manipulation could become substantial and expose the system to potential price manipulation exploits.

Recommendation

Consider using the wstETH/USD Pyth Price Feed (0x6df640f3b8963d8f8358f791f352b8364513f6ab1cca5ed3f1f7b5448980e784) directly to accurately retrieve the price of wstETH at any given timestamp.

Resolution

Smardex Team: Acknowledged.

L-07 | Stuck Action Functions Lack ReentrancyGuard

Category	Severity	Location	Status
Unexpected Behavior	● Low	UsdnProtocolFallback.sol	Resolved

Description

In the UsdnProtocolFallback contract the removeBlockedPendingAction functions do not have an initializedAndNonReentrant modifier. This allows potentially unexpected state updates to occur in the rare case where a trusted party re-enters into the system to call these functions during an action validation or initiation.

Recommendation

Consider adding the initializedAndNonReentrant modifier to the removeBlockedPendingAction functions.

Resolution

Smardex Team: The issue was resolved in PR#694.

L-08 | TickMath Rounding

Category	Severity	Location	Status
Documentation	● Low	TickMath.sol	Acknowledged

Description

The getTickAtPrice function is subject to several rounding errors which cause prices which should correlate to one tick to correlate to a lower tick. This is because the lnWad function rounds down and this precision loss is not corrected for not accounted for in the getPriceAtTick function. Thus there are cases where the result of getPriceAtTick does not agree with the result of getTickAtPrice.

For example, getTickAtPrice(getPriceAtTick(4)) == 3. This invalidates a core invariant of any TickMath library and leads to potentially unexpected cases. In the getTickAtPrice function when the ln value is positive, round down division is used. However in the case where 1 wei has been rounded off of the ln result this results in a full tick loss in precision. This is clear with the example of tick 4:

• LN_BASE = 99_995_000_333_308
• getPriceAtTick(4) = 1000400060004000098
• lnWad(1000400060004000098) = 399980001333231
• 399980001333231 / 99_995_000_333_308 = 3.99999999999998999500008332883**…**
• 399980001333232 / 99_995_000_333_308 = 4

In fact in the positive ln case the ln value will only ever be 1 wei less than it ought to be when the imprecision occurs. Therefore it can be corrected by checking if imprecision has occurred and adding 1 wei if it has. The negative ln case experiences the same imprecision, except the lnWad function rounds the magnitude down, not the signed value.

However this cannot be resolved by simply deducting by 1 wei since the LN_BASE value truncates precision. The truncation of the LN_BASE value ultimately yields imprecision which increases the result of the division by reducing the denominator. Because of this imprecision upwards on the magnitude of the division result, the roundUp division often exacerbates the precision loss. To resolve both of these imprecisions, round down division can be used for the negative case.

With both of these adjustments in place, the getTickAtPrice(getPriceAtTick(tick)) == tick invariant holds. This rounding error rounds in the benefit of the protocol, treating the current price as if it were on a lower tick and thus performing liquidations earlier than they technically should occur. In this case the rounding error may be acceptable to the protocol and preferred over an alternative implementation.

Recommendation

Be aware of this rounding imprecision, since it rounds in favor of the protocol the imprecision may be acceptable and this finding can serve to document the behavior.

Resolution

Smardex Team: Acknowledged.

L-09 | _getLatestStoredPythPrice Lacks Validation

Category	Severity	Location	Status
Validation	● Low	PythOracle.sol: 165	Resolved

Description

In the `_getFormattedPythPrice` function there is a validation that prevents the use of any pyth feed which reports a positive expo. This avoids a significant underflow issue which would occur in the `_formatPythPrice` function where the expo is negated before being cast to a uint.

However in the case where the low latency price is not used and the latest pyth stored price is queried with the `_getLatestStoredPythPrice` function the `_formatPythPrice` function is called without first performing this validation on the pyth result.

Recommendation

Consider adding the expo validation to the `_getLatestStoredPythPrice`.

Resolution

Smardex Team: The issue was resolved in PR#652.

L-10 | Storage Compatibility

Category	Severity	Location	Status
Upgradeability	● Low	Global	Resolved

Description

The USDN codebase uses a UUPS upgradeability pattern, but not all abstract contracts use namespaced storage. This can create issues with future upgrades if the storage layout were to change as documented here: https://eips.ethereum.org/EIPS/eip-7201.

For example the `InitializableReentrancyGuard` and `UsdnProtocolStorage` contracts include storage variables which use the default assigned storage slot.

Recommendation

Consider using the namespaced storage pattern for these storage values.

Resolution

Smardex Team: The issue was resolved in PR#666.

L-11 | Unexpected Share Approval Reverts

Category	Severity	Location	Status
DoS	● Low	Usdn.sol	Resolved

Description

The approved amount for the Usdn token is in tokens instead of shares. Therefore when attempting to transfer or burn shares from an address there may be unexpected reverts in the event of a rebase.

Consider the following scenario:
• User A is approved to spend 10 USDN tokens of User B
• Divisor is initially 1e18
• User A sends a transaction to the mempool to transfer 10 shares from User B with the `transferSharesFrom` function
• A transaction triggering a rebase is recorded before User A's transaction and the divisor becomes 0.9e18
• User A's transaction is now attempting to transfer 10 / 0.9 = 11.11... tokens from User B, which exceeds the allowance of 10 tokens
• As a result the transaction reverts

Recommendation

Clearly document this behavior so that integrators and users are aware of this risk.

Resolution

Smardex Team: The issue was resolved in PR#684.

L-12 | Read Only Reentrancy Risk

Category	Severity	Location	Status
Reentrancy	● Low	UsdnProtocolVaultLibrary.sol: 690	Resolved

Description

In the `_initiateDeposit` function the `transferCallback` is invoked before the `s._pendingBalanceVault` is updated. This gives the caller control over the transaction execution while not all state updates have occurred for the deposit.

This could potentially be used to exploit integrating systems which would rely on the imbalance checks which incorporate the `_pendingBalanceVault`.

Recommendation

Ensure all state updates have occurred for the deposit before making the external callback. This way untrusted actors can only take control of the transaction execution when the system is in a valid state.

Resolution

Smardex Team: The issue was resolved in PR#702.

L-13 | Tick Penalty Changes Leverage

Category	Severity	Location	Status
Documentation	● Low	UsdnProtocolLongLibrary.sol: 304	Resolved

Description

The desiredLiqPrice provided by a user upon initiating the opening of a position corresponds to their liquidation tick price and not their effective liquidation price without penalty.

The liquidation price without penalty is used to compute the position's exposure and thus leverage, therefore if the liquidation penalty changes unexpectedly before a user's transaction is recorded then the user's leverage could change significantly.

There is currently a user supplied userMaxLeverage validation, however the position could have unexpectedly low leverage which would not trigger this validation.

Recommendation

Consider documenting this risk to users. Otherwise consider introducing a userMinLeverage validation.

Resolution

Smardex Team: The issue was resolved in PR#710.

L-14 | Missing Require Check In setMinLongPosition

Category	Severity	Location	Status
Validation	● Low	UsdnProtocolFallback.sol: 716	Acknowledged

Description

The USDN protocol implements a setter function to update the s._minLongPosition value:

```
     function setMinLongPosition (uint256 newMinLongPosition) external
onlyRole(SET_PROTOCOL_PARAMS_ROLE) {s._minLongPosition = newMinLongPosition;
emit IUsdnProtocolEvents.MinLongPositionUpdated(newMinLongPosition);
IBaseRebalancer rebalancer = s._rebalancer;

     if (address(rebalancer) = address(0) && rebalancer.getMinAssetDeposit()
< newMinLongPosition) {rebalancer.setMinAssetDeposit(newMinLongPosition);}}
```

This value sets the minimum required amount of wstETH for a user to open a long position. However, the function lacks a validation check to ensure that the newMinLongPosition parameter is greater than the current maxReward defined in the LiquidationRewardsManager contract.

If s._minLongPosition is set to a very low value (e.g., 0.001 wstETH), it could be exploited by a malicious user which could open multiple small positions, wait for them to become liquidatable, and then profit by liquidating them. The liquidation rewards would exceed the value of each individual position.

Recommendation

Ensure that maxReward is never higher than the current s._minLongPosition.

Resolution

Smardex Team: Acknowledged.

L-15 | validateOpenPosition Might Not Be Completed

Category	Severity	Location	Status
Validation	● Low	UsdnProtocolActionsLongLibrary.sol: 612	Resolved

Description

In the validateOpenPosition function, the _prepareValidateOpenPositionData internal function is called in order to update the protocol balances, liquidate positions if the given price is recent and necessary and validate the open position action.

If the Pyth price provided is not recent (older publish time than the current publish time of the protocol lastPrice) liquidations will not be performed and the function will try to validate the open position action. This will occur often as the validation time given should belong to initiateTimestamp + 24. During the validation, the following check is performed:

```
    if (data_.lastPrice < data_.liqPriceWithoutPenalty) {the position must be
liquidated data_.isLiquidationPending = true; return (data_, false);}
```

If the protocol lastPrice is lower than data_.liqPriceWithoutPenalty the _prepareValidateOpenPositionData function will return the tuple (data_, false) where data_.isLiquidationPending will be set to true.

Therefore the _validateOpenPositionWithAction would return the tuple (false, false) for (bool isValidated_, bool liquidated_) meaning that the position was not validated or liquidated. Basically the position was not validated because it was liquidatable but it was not liquidated because the Pyth price provided was not recent.

At this point the action would remain in the queue and an extra transaction to liquidate the position would be required.

Recommendation

In this case, consider automatically executing the liquidation of the position using the current lastPrice.

Resolution

Smardex Team: The issue was resolved in PR#722.

L-16 | USDN Depositors May Pay Less Fees On Withdrawal

Category	Severity	Location	Status
Unexpected Behavior	● Low	Global	Acknowledged

Description

Since the fees are distributed back to the vault on withdrawals a depositor can pay less fees by splitting their withdrawal up into two separate withdrawals, so that the second withdrawal gets a piece of the first withdrawal fee.

Recommendation

This behavior is likely not worth the complexity to fully address it. Simply be aware of this behavior.

Resolution

Smardex Team: Acknowledged.

L-17 | Typo

Category	Severity	Location	Status
Typo	● Low	UsdnProtocolCoreLibrary.sol: 257	Resolved

Description

In the `_applyPnlAndFunding` function, it is mentioned that "in case of positive funding, the vault balance must be decremented ...". However this is incorrect as in the case of positive funding the long balance must be decremented.

Recommendation

Correct the comment to state that, "in case of positive funding, the *long* balance must be decremented"

Resolution

Smardex Team: The issue was resolved in PR#645.

L-18 | Insufficient Liquidation Incentives

Category	Severity	Location	Status
Incentives	● Low	LiquidationRewardsManager.sol: 169	Acknowledged

Description

In the _calcGasPrice function the resulting gas price is capped to the lower of the block.basefee + baseFeeOffset and the tx.gasprice.

However in the case where the tx.gasprice is higher than the block.basefee + baseFeeOffset, then it is possible that the resulting liquidation rewards are an insufficient incentive to call the liquidate function.

This can occur in the case where the tx.gasprice is significantly larger than the block.basefee + baseFeeOffset and when the tick being liquidated is at the current price tick, and thus no bonus reward is given.

In this case it is likely that the fixedReward would be exceeded in terms of gas costs for the liquidation call.

Recommendation

Consider whether the insufficient incentive in this case is acceptable to the protocol. If it isn't, consider allowing the gas price used in the reward calculation to be the current tx.gasprice.

This should not be a large issue when the tx.gasprice is high as there is a configurable maxReward to limit the size of the liquidation reward to a reasonable price. Otherwise be sure to configure the baseFeeOffset with this behavior in mind.

Resolution

Smardex Team: Acknowledged.

L-19 | Liquidations Untriggered By Recent Price

Category	Severity	Location	Status
Unexpected Behavior	● Low	UsdnProtocolCoreLibrary.sol: 229	Acknowledged

Description

In the _applyPnlAndFunding function the function early returns if the price is from the same timestamp as the current lastPrice:

```
        if (timestamp <= lastUpdateTimestamp) {return Types.ApplyPnlAndFundingData({isPriceRecent:
timestamp == lastUpdateTimestamp, tempLongBalance: s._balanceLong.toInt256(), tempVaultBalance:
s._balanceVault.toInt256(), lastPrice: s._lastPrice});}
```

The price is determined to be recent if timestamp == lastUpdateTimestamp, and liquidations will be performed if the price is recent.

However the liquidations will be performed with the lastPrice instead of the currentPrice thus when lastPrice != currentPrice liquidations may not occur when they should have.

For example the existing lastPrice may come from the last chainlink update at t=100 with a price of $97. Meanwhile the new currentPrice may be a Pyth update which also comes from t=100 and has a price of $96.50.

The pyth update may more accurately represent the market price and there may be liquidations that need to occur at $96.50 but can not at $97.

Recommendation

Consider using the lower of the lastPrice and currentPrice in this scenario to trigger liquidations as conservatively as possible in the protocols favor.

Resolution

Smardex Team: Acknowledged.

L-20 | Inaccurate NatSpec

Category	Severity	Location	Status
Documentation	● Low	UsdnProtocolActionsUtilsLibrary.sol: 426	Resolved

Description

In the documentation for the _assetToRemove function in the UsdnProtocolActionsUtilsLibrary file it is mentioned that the posExpo parameter represents the total exposure of the position. However this is often not the case as the _assetToRemove function is used for partial position decreases.

Recommendation

Consider updating the documentation to specify that the posExpo parameter represents the amount of exposure that is to be closed from the position rather than the entire exposure of the position.

Resolution

Smardex Team: The issue was resolved in PR#618.

L-21 | Rebalancer Always Utilizes All Available Liquidity

Category	Severity	Location	Status
Documentation	● Low	UsdnProtocolLongLibrary.sol: 1036	Acknowledged

Description

The _triggerRebalance function currently uses all the available liquidity in the rebalancer, along with the rebalancer's position value, to open a new long position. It calculates the rebalancer position's tick through the _calcRebalancerPositionTick function, ensuring that the leverage of the opened position is at least equal to the protocol's minimum leverage. However, the minimum leverage, REBALANCER_MIN_LEVERAGE, is hardcoded to a value slightly above 1:

```
      uint256 internal constant REBALANCER_MIN_LEVERAGE = 10 ** LEVERAGE_DECIMALS + 1; //
x1.000000000000000000001
      Due to this extremely low minimum leverage, the following if code block is never entered,
leading to scenarios where the rebalancer can open a new position with leverage as low as
1.000000000000000000001:
      check that the trading expo filled by the position would not be below the min leverage
data.lowestUsableTradingExpo = positionAmount * Constants.REBALANCER_MIN_LEVERAGE / 10 **
Constants.LEVERAGE_DECIMALS - positionAmount;
      if (data.lowestUsableTradingExpo > tradingExpoToFill) {tradingExpoToFill =
data.lowestUsableTradingExpo;}
```

As a result, if the rebalancer has high liquidity, a new position with minimal leverage (just above 1) is created. In such case, rebalancer depositors will experience minimal profits because:
• The bonus is split among all rebalancer depositors, diluting the reward.
• The potential profit and loss (PnL) from the position is negligible, as the leverage is very close to 1, offering little profit opportunity from price movements.

Moreover, after the rebalancer is triggered and all available liquidity is used to open a new position, the protocol may soon find itself in an imbalanced state, requiring another rebalance. This creates an additional challenge as the rebalancer can only be triggered by a liquidation event, which may be infrequent in certain scenarios.

Recommendation

Consider adjusting the minimum leverage threshold to a more meaningful value, ensuring that new rebalancer positions are opened with higher leverage, offering more substantial PnL opportunities for depositors. On the other hand, a major refactoring of the Rebalancer contract might be necessary.

Rather than using all available liquidity at once, the rebalancer should utilize liquidity in a more efficient, FIFO (First In, First Out) manner. This would prioritize users who deposited earlier, allowing their assets to be deployed first in the rebalancer. By doing so, the rebalancer could selectively draw liquidity as needed.

Resolution

Smardex Team: Acknowledged.

L-22 | Rebalancer NonReentrant Modifiers

Category	Severity	Location	Status
Reentrancy	● Low	Rebalancer.sol	Resolved

Description

In the Rebalancer contract, currently only the `initiateClosePosition` function has a `nonReentrant` modifier. A `nonReentrant` modifier will be added to the `updatePosition` function to prevent a Critical issue which allows for the double closing of the Rebalancer position.

However out of an abundance of caution it would be safest to add the `nonReentrant` modifier to the other user facing Rebalancer functions.

Recommendation

Consider adding a `nonReentrant` modifier to all user facing functions in the `Rebalancer` contract out of an abundance of caution.

Resolution

Smardex Team: The issue was resolved in PR#725.

L-23 | Missing Deadline Documentation

Category	Severity	Location	Status
Documentation	● Low	UsdnProtocolVaultLibrary.sol: 1090	Resolved

Description

In the documentation for the _isActionable function in the UsdnProtocolVaultLibrary library the lowLatencyDeadline and onChainDeadline parameters are undocumented in the function NatSpec.

Recommendation

Document the lowLatencyDeadline and onChainDeadline parameters in the NatSpec for the _isActionable function.

Resolution

Smardex Team: The issue was resolved in PR#646.

L-24 | Invalid Long Exposure Perturbs Accounting

Category	Severity	Location	Status
Documentation	● Low	Global	Acknowledged

Description PoC

During the initiation of an open position action, the position is effectively recorded as being opened from the protocol's perspective. The protocol accounts for this additional exposure on the liquidation tick and for the long aggregate accounting. However the trader is not held accountable for the PnL that occurs after initiation and before validation.

Therefore this exposure does not actually exist for the protocol, meanwhile the protocol thinks that it does. This is ultimately corrected in terms of the PnL calculations with the `_validateOpenPositionUpdateBalances` function, however before this correction is made the aggregate balances of the vault vs. longs is misattributed due to the "ghost" pnl of the initiated but not yet validated position.

Firstly, this allows USDN shareholders to exit the vault before a trader's "ghost" losses have been corrected. Or enter the vault before a trader's "ghost" gains have been corrected. The risk of gaming through this vector is limited due to the two-step nature of vault deposits and withdrawals, however it gives an incentive for users to control the ordering of validations which occur to extract as much value out of the temporary mis-accounting.

Secondly, this perturbs the imbalance measuring since the system will count profit or losses from the temporary positions as if they have real exposure. Most notably, the rebalancer may be deemed unnecessary after a large liquidation, when in fact it should be used when accounting for the fact that "ghost" PnL should be corrected.

This behavior is however necessary to avoid giving trader’s a risk free opportunity for profit with pre-knowledge of their execution price and the inaccuracies should be small given that the earliest price in the range from initiation to validation is used.

Recommendation

This finding serves only to document this behavior. Be aware of these potentially unexpected behaviors and how they could be manipulated in unlikely scenarios.

Resolution

Smardex Team: Acknowledged.

L-25 | Position Fee Not Refunded On Removal

Category	Severity	Location	Status
Unexpected Behavior	● Low	UsdnProtocolCoreLibrary.sol: 376	Acknowledged

Description

In the `_removeBlockedPendingAction` function when removing a `ValidateOpenPosition` pending action the position fee which was charged to the user based on their entry price and given to the vault is not refunded to the user.

Recommendation

Consider if this is the expected behavior. If it is not then consider refunding the amount of position fee paid to the `to` address in the `_removeBlockedPendingAction` function.

Resolution

Smardex Team: Acknowledged.

L-26 | Gaming Vault Minted Shares

Category	Severity	Location	Status
Gaming	● Low	UsdnProtocolVaultLibrary.sol: 777	Acknowledged

Description PoC

Users can deposit assets into the vault, using initiateDeposit and executing the deposit with validateDeposit. However, the initiate action does not enforce a specific oracle to be used.

In the case of Chainlink, it will be allowed as long it's not older than _timeElapsedLimit, if s._lastPrice has not been updated recently, and the published time is not older than the on chain Pyth price (which is not updated constantly in Ethereum).

Users can then choose a specific old Chainlink update that will result in more USDN shares being minted, as long as the validation price is lower than the initiation price. In the following scenarios, we can observe that the balanceVault will be increased, but the value will be lower than using a lower initiation price:

initiate price > validate price
- initiatePrice: 1995 (Chainlink)
- balanceVault at initiation: 98.67
- validatePrice: 1990.79
- balanceVault adjusted: 98.88
- USDC shares minted: 19,894.48

initiate price < validate price
- initiatePrice: 1979.9 (Chainlink)
- balanceVault at initiation: 99.42
- validatePrice: 1990.79
- balanceVault not adjusted: 99.42
- USDC shares minted: 19,786.67

Recommendation

Consider this scenario when assigning the max stale threshold for Chainlink using _timeElapsedLimit.

Resolution

Smardex Team: Acknowledged.

L-27 | 0 Shares Wrapping

Category	Severity	Location	Status
Validation	● Low	Wusdn.sol: 148	Resolved

Description

In the _wrapShares function the resulting wrappedAmount_ may round down to zero when the usdnShares amount is less than the SHARES_RATIO. This will result in a 0 amount of wrapped Usdn being minted to the user and a 0 amount of Usdn shares being transferred from the user.

Recommendation

Consider validating that the wrappedAmount_ is greater than 0 in the _wrapShares function to avoid unexpected wrap calls.

Resolution

Smardex Team: The issue was resolved in PR#660.

L-28 | Hardcoded Maximum Pyth Fee

Category	Severity	Location	Status
Validation	● Low	PythOracle.sol: 71	Acknowledged

Description

As per the Pyth Network Documentation:
"The Pyth Network protocol has been designed to allow for the optional enablement of data fees in order to update the state of an on-chain price feeds. The ongoing existence of and size of the fee will be determined by governance on a per-blockchain basis; until governance is live, the fee will be 1 of the smallest denomination of the blockchain's native token (e.g., 1 wei on Ethereum)."

The PythOracle contract implements this check:

```
uint256 pythFee = _pyth.getUpdateFee(pricesUpdateData);
if (pythFee > 0.01 ether) {revert OracleMiddlewarePythFeeSafeguard(pythFee);}
```

If the Pyth fee is set in the future to a value higher than 0.01 ether by the Pyth governance the USDN protocol would be totally blocked as all the interactions with the Pyth price feed would revert.

Recommendation

The 0.01 ether should not be a hard-coded constant. Consider implementing a privileged function to update the max. Pyth fee supported.

Resolution

Smardex Team: Acknowledged.

L-29 | Rebalancer Tick Increment May Reduce Exposure

Category	Severity	Location	Status
Unexpected Behavior	● Low	UsdnProtocolLongLibrary.sol: 1077	Acknowledged

Description

In the _calcRebalancerPositionTick function when computing the rebalancer's liquidation tick there is a case to handle liquidation ticks that do not meet the longImbalanceTargetBps. In this case a single tick spacing is added to the liquidation tick to increase the leverage and thus the exposure of the rebalancer.

This is in hopes to fill the missing exposure so that the resulting imbalance will be within the longImbalanceTargetBps. However the resulting liqPriceWithoutPenalty may in fact be lower than the original liqPriceWithoutPenalty in the event that the liquidation penalty on the new liquidation tick is more than one tick spacing larger than the liquidation penalty on the old tick.

In this case the resulting exposure after the adjustment by 1 tick spacing will be lower than before the adjustment. This further moves the long exposure away from reaching the longImbalanceTargetBps rather than towards it.

Recommendation

This case will only appear in production if the liquidation penalty is assigned to a value that is greater than one full tick spacing above or below any liquidation penalty that was assigned in the past and is currently active on a tick.

No code change may be necessary if this fact is considered carefully while configuring updates to the liquidation penalty.

Otherwise if these updates would occur, it would be best to take the rebalancer position that corresponds to the highest exposure between the original determined rebalancer position and the position which has been adjusted up by one tick spacing.

Resolution

Smardex Team: Acknowledged.

L-30 | Positive Value Liquidations May Still Cause Bad Debt

Category	Severity	Location	Status
Unexpected Behavior	● Low	UsdnProtocolLongLibrary.sol: 532	Acknowledged

Description

When a tick is liquidated, the `_sendRewardsToLiquidator` function is triggered. This function calculates the liquidation rewards using the `getLiquidationRewards` function, removes the rewards from `s._balanceVault` and transfers them to the liquidator.

However, the liquidation rewards are not closely tied to the remaining value of the liquidated tick. Even if a tick is liquidated late (with a negative remaining value, indicating bad debt), a fixed amount of wstETH is still sent to the caller and deducted from `s._balanceVault`, which can increase/worsen the bad debt.

Additionally, the closer the `currentPrice` is to the liquidation price, the higher the position size bonus will be. This can result in a situation where, for instance, a tick with a positive remaining position value of only 1 is liquidated, but the rewards are high enough that a liquidation executed on time still leads to bad debt as these rewards are taken from the vault.

Recommendation

Consider making liquidation rewards partially dependent on the remaining value of the liquidated tick. This would prevent over-rewarding liquidations that occur late or when the remaining value is low.

Resolution

Smardex Team: Acknowledged.

L-31 | USDN Price Can Be Easily Manipulated

Category	Severity	Location	Status
Gaming	● Low	UsdnProtocolLongLibrary.sol: 484	Resolved

Description

Through the use of a “Proxy” contract it is possible to increase the vault and long exposure atomically. This “Proxy” contract can deploy other contracts that deposit and open long positions in a balanced way. We have performed multiple tests around this, and this is how the USDN price was affected.

In the tests below, the following approach was taken:

• _usdnRebaseInterval set to 1.
• Deploy a “Proxy” contract.
• “Proxy” contract initiates multiple deposits/long position opens in a balanced way and using the same lastPrice.
• “Proxy” contract validates, 24 seconds later, all the initiated actions in the same order but with a price increase/decrease (percentage). For example, 85 percentage, means that if the initiation price used was 1000, the validation price used was 850.
• Final USDN Price represents the USDN price after all these validations were performed.
• The initial USDN Price in these tests was 1.00$.

Test #1:
• Initial Vault/Long exposure: Around 98e18.
• Total wstETH added through deposits/long positions: 100e18.
• Leverage of positions opened: 2x.

Test #2:
• Initial Vault/Long exposure: Around 98e18.
• Total wstETH added through deposits/long positions: 100e18.
• Leverage of positions opened: 4x.

Test #3:
• Initial Vault/Long exposure: Around 98e18.
• Total wstETH added through deposits/long positions: 500e18.
• Leverage of positions opened: 2x.

Test #4:
• Initial Vault/Long exposure: Around 98e18.
• Total wstETH added through deposits/long positions: 500e18.
• Leverage of positions opened: 4x.

In each of the tests it was observed that given a user with enough liquidity it is possible to manipulate the USDN price, possibly exposing it to speculation and opportunistic behavior. This "artificial" price alteration could create a favorable condition for anyone attempting to profit through short positions on USDN in external markets. In such an environment, a speculator could trigger a price movement, open a leveraged short position and benefit directly from the price drop that they themselves initiated.

Recommendation

_usdnRebaseInterval should be kept unset so that USDN rebases can occur constantly without delay, ensuring that the decrement of the divisor is as low as possible to prevent any potential manipulation. Another possible suggestion is restricting the amount of exposure that could be pending validation at the same time. This could also help mitigate the impact of the L-04 issue.

Resolution

Smardex Team: The issue was resolved in PR#653.

L-32 | Lacking securityDepositValue Validation

Category	Severity	Location	Status
Validation	● Low	UsdnProtocolFallback.sol: 659	Resolved

Description

The setSecurityDepositValue function performs no validation on the new securityDepositValue. Therefore the securityDepositValue may be assigned to zero or an extraordinarily high amount.

Recommendation

Consider introducing validations for the setSecurityDepositValue function similarly to other setter functions to prevent invalid configurations.

Resolution

Smardex Team: The issue was resolved in PR#658.

L-33 | Depositors Pay Reduced Fees

Category	Severity	Location	Status
Logical Error	● Low	UsdnProtocolVaultLibrary.sol: 798	Resolved

Description

During the validation of a deposit, the deposit fees are not added to the balanceVault stack variable before computing the amount of mintedTokens, thus the depositor subsequently owns a portion of their own fees. This is clear if you imagine the scenario for a depositor with 10x the magnitude of the current vault balance:

```
* fee percent = 1%
* balanceVault = 10
* usdnTotalShares = 10
* deposit.amount = 100
* amountAfterFees = 99
* mintedTokens = amountAfterFees * balanceVault / usdnTotalShares
* mintedTokens = 99 * 10 / 10 = 99
* usdnTotalShares = 109
* balanceVault = 110
* balance of bob's shares = 99 * 110 / 109 = 99.9
* Thus instead of paying a 1% fee, bob pays a 0.1% fee
```

Though the normal circumstance will show a fee reduction much less than this.

Recommendation

Consider if this is the expected behavior or not. If it is not, consider adding the fee value to the balanceVault before computing the mintedTokens amount so that depositors pay the full fee amount to the other vault depositors.

If this approach is taken then care should be taken to update the usdnSharesToMintEstimated value in the _prepareInitiateDepositData function.

Resolution

Smardex Team: The issue was resolved in PR#635.

L-34 | Potential Precision Loss In _positionValue Calculation

Category	Severity	Location	Status
Unexpected Behavior	● Low	UsdnProtocolUtilsLibrary.sol: 438	Acknowledged

Description

In the current implementation, the _positionValue function calculates the value of a position based on the current price, liquidation price without penalty and the position's total exposure.

However, due to Solidity's integer division, there is a risk of significant precision loss when the currentPrice is very close to liqPriceWithoutPenalty or when the closed exposure is very small relative to the currentPrice:

```
      function _positionValue(uint128 currentPrice, uint128 liqPriceWithoutPenalty, uint128
positionTotalExpo) internal pure returns (int256 value_)

      {if (currentPrice < liqPriceWithoutPenalty) {value_ = -FixedPointMathLib
.fullMulDiv(positionTotalExpo, liqPriceWithoutPenalty - currentPrice, currentPrice)
.toInt256();} else {value_ = FixedPointMathLib.fullMulDiv(positionTotalExpo, currentPrice -
liqPriceWithoutPenalty, currentPrice).toInt256();}}
```

When the currentPrice is very close to the liqPriceWithoutPenalty, or when the exposure is extremely small, the result of the division can be rounded down. This can lead to the user not receiving the correct payout, especially during a long position closure, causing a loss of funds.

Recommendation

Consider enforcing a minimum amountToClose in the initiateClosePosition function to prevent situations where the assets received are either zero or significantly reduced due to rounding.

This would ensure that users avoid losses caused by closing very small positions where precision loss or rounding can result in negligible payouts.

Resolution

Smardex Team: Acknowledged.

L-35 | TotalSupply Exceeded By User Balances

Category	Severity	Location	Status
Rounding	● Low	Usdn.sol	Resolved

Description

The totalSupply of the Usdn token uses the _convertToTokens function with Rounding.Closest, however due to precision loss, this may disagree with the summation of each individual user's balance.

For example, consider the following scenario:

• _totalShares are 10e36 + 10 wei
• divisor is 1e18
• totalSupply is computed as 10e36 + 10 wei/1e18 = 10e18 with closest rounding
• User A holds 1000000000000000000500000000000000010 shares
• User A balance is 1e36 + 0.5e18 + 10 wei / 1e18 = 1e18 + 1 wei (1000000000000000001)
• User B holds 8999999999999999999500000000000000000 shares
• User B balance is 9000000000000000000
• The summation of User A and User B Balances is 10e18 + 1, but the totalSupply is reported as 10e18.

There is no immediate large impact for the USDN system, however this is worth noting especially for consumers of the USDN token balances amounts.

Recommendation

Be sure to document this inconsistency for integrators and consumers of the USDN balances.

Resolution

Smardex Team: The issue was resolved in PR#728.

L-36 | getHighestPopulatedTick Might Return An Incorrect Tick

Category	Severity	Location	Status
Unexpected Behavior	● Low	UsdnProtocolFallback.sol: 421	Resolved

Description

The function getHighestPopulatedTick is used to retrieve the highest tick with an open position:

```
function getHighestPopulatedTick() external view returns (int24) {return s._highestPopulatedTick;}
```

However, when the last position of a tick is fully closed the _removeAmountFromPosition function does not update the s._highestPopulatedTick with the new highest populated tick.

Therefore, in cases where the last position in the highest populated tick is closed, the tick remains marked as the highest populated despite having no open positions.

This issue does not impact liquidations or cause any significant effects, aside from temporarily displaying an incorrect value for s._highestPopulatedTick. The incorrect tick will persist until either a liquidation occurs or a new position is opened at a higher tick.

Recommendation

Consider updating the s._highestPopulatedTick when the last position of a tick is closed.

Resolution

Smardex Team: The issue was resolved in PR#705.

L-37 | No Slippage On Validation

Category	Severity	Location	Status
Validation	● Low	UsdnProtocolActionsLongLibrary.sol: 209	Acknowledged

Description

When validating the opening of a position the real entry price of a position is determined. The actual entry price experienced by a position on validation cannot be predicated as it is subject to price changes between the initiation and validation time frame.

A user is currently able to specify a `maxPrice` which the position can be opened at, however this is only enforced on initiation unlike the `maxLeverage`. Thus a trader can have their position opened past their `maxPrice` which may be unexpected.

However currently this is necessary to avoid risk free trade opportunities in the timeframe between the validation price and the validation timestamp.

Recommendation

Be sure to document this behavior to users so that they are aware of this when assigning the `maxPrice` value.

Resolution

Smardex Team: Acknowledged.

L-38 | _triggerRebalance Does Not Account For The Current Action Effect

Category	Severity	Location	Status
Unexpected Behavior	● Low	UsdnProtocolLongLibrary.sol: 532	Acknowledged

Description

The `_triggerRebalance` function is responsible for initiating a rebalance when the protocol experiences a significant imbalance on the long side. It calculates the rebalancer's new position tick using the `_calcRebalancerPositionTick` function, aiming to bring the protocol back to a balanced state.

However, this function does not account for the effects of ongoing actions, such as an initiation or validation, which may impact the current state of the protocol. When an action like position initiation or validation is in progress, the actual state of the protocol will change, but `_triggerRebalance` operates based on the previous state, failing to consider these changes.

This could result in inaccurate rebalancing, as the rebalancer may open a position that does not reflect the real-time protocol conditions.

Recommendation

Even though this could require some major refactoring, consider executing the initiation or validation actions before calling `_triggerRebalance`. By doing so, the rebalancer can account for any changes these actions introduce, ensuring that the rebalancing is based on the updated protocol state.

This approach would prevent the rebalancer from operating on outdated information and help maintain a more accurate balance in the protocol.

Resolution

Smardex Team: Acknowledged.

L-39 | DOS State Due To s._minLongPosition Restriction

Category	Severity	Location	Status
Validation	● Low	Global	Acknowledged

Description

The protocol implements several safeguards to maintain balance between the long side and the vault side:

• _checkImbalanceLimitDeposit: Ensures that the protocol does not allow deposits that would create an imbalance beyond the vault side's deposit limit.
• _checkImbalanceLimitWithdrawal: Ensures that withdrawals do not cause an imbalance exceeding the long side's withdrawal limit.
• _checkImbalanceLimitOpen: Prevents the opening of new long positions that would cause an imbalance beyond the open limit on the long side.
• _checkImbalanceLimitClose: Prevents closing positions if it would create an imbalance beyond the close limit on the vault side.

Additionally, the protocol enforces the s._minLongPosition variable, which sets the minimum allowable size for long positions. As users interact with the protocol, a highly balanced state could emerge. However, in such a scenario, users might face difficulties in opening new positions with the minimum leverage allowed by s._minLongPosition.

This would occur because the position size would not satisfy the _checkImbalanceLimitOpen restriction. Furthermore, users may also be unable to close their open positions as:

• Attempting to fully close a position would fail because it would violate the _checkImbalanceLimitWithdrawal.
• Attempting to partially close a position would revert because the remaining position would fall below the s._minLongPosition.

This situation would block users on the long side, leaving them exposed to liquidation risks. To resolve this issue, a privileged account would need to invoke the setMinLongPosition function to temporarily lower the s._minLongPosition threshold or adjust the openExpoImbalanceLimitBps and closeExpoImbalanceLimitBps to expand the imbalance limits.

Recommendation

Ensure that the exposure on both sides (vault and long) is sufficient to accommodate the opening of new positions with the minimum allowed position size (s._minLongPosition) and the lowest leverage. This is particularly important during the initialization phase of the protocol, when the initialize() function is called.

Resolution

Smardex Team: Acknowledged.

L-40 | _usdnRebaseInterval Unassigned On Initialization

Category	Severity	Location	Status
Unexpected Behavior	● Low	UsdnProtocolImpl.sol: 33	Resolved

Description

In the initializeStorage function the _usdnRebaseInterval value is not assigned with an initial value. This may be intended if the initial value for _usdnRebaseInterval is intended to be zero. However if the initial value should not be zero then the initialization is missing.

Recommendation

If the initial _usdnRebaseInterval value should not be zero then initialize it to the appropriate value.

Resolution

Smardex Team: The issue was resolved in PR#653.

L-41 | Full Operational Halt During Protocol Pause

Category	Severity	Location	Status
Unexpected Behavior	● Low	Global	Acknowledged

Description

When the protocol enters a paused state all operations are suspended. This includes:

• Initiating any type of new action.
• Validating any ongoing actions.
• Performing liquidations.

As a result, users are unable to respond to market price fluctuations since they cannot close their positions.

Furthermore, because liquidations cannot be executed while the protocol is paused, it is highly likely that upon unpausing the protocol, there will be multiple liquidations involving bad debt, as once the protocol resumes, positions that should have been liquidated earlier could be severely underwater.

Recommendation

It is recommended to either:

1. Allow liquidations during the paused state: By enabling liquidations while other operations remain paused, the system could still manage risk and prevent bad debt from accumulating.
2. Implement a granular pausing mechanism: Rather than pausing the entire protocol, consider splitting the pause functionality to selectively suspend non-critical functions. For example, initiation of new actions and validations could be paused, but liquidations and certain other risk management functions could continue to operate, ensuring the system remains secure while allowing for flexibility.

Resolution

Smardex Team: Acknowledged.

L-42 | Update Events Emitted On No Update

Category	Severity	Location	Status
Events	● Low	UsdnProtocolFallback.sol	Acknowledged

Description

In the Fallback contract there are many setter functions which emit events for the assignment of new values for important addresses and configurations.

However in these setter functions there may be no update to the stored values when the new value is the same as the existing value, meanwhile the update event is still emitted. This may cause confusion for consumers of these events as no update has actually occurred.

Recommendation

Consider validating that the new value is not the same as the old value in the setter functions to avoid emitting update events when no update has occurred.

Resolution

Smardex Team: Acknowledged.

L-43 | Deposits Experience Invalid Funding

Category	Severity	Location	Status
Logical Error	● Low	Global	Acknowledged

Description

Vault deposits do not affect the skew of USDN vault exposure vs. trader exposure because the _pendingBalanceVault value is not included in the funding calculation.

However deposits are still affected from the funding which occurs in the timeframe from [initiate., validate] since the original balanceVault is used to compute the assets they receive.
For example:

- Vault balance upon initiate deposit: 10
- Funding occurs which adds 5 to the vault bal
- Vault bal used in validate deposit is the cached 10
- Shares = assets * shares / totalAssets = 10 * 10 / 10 = 10 shares

Current value of the 10 shares = 10 * 25 / 20 = 12.5, the depositor got the funding from [initiate, validate] but the depositor didn’t affect the skew.

As a result a large deposit could siphon funding from the depositors who rightfully should have received it. Additionally, if this large deposit was otherwise included in the vaultBalance then the vault would have been paid less funding or may have even had to pay funding.

Recommendation

Consider using a vault balance which is affected by the funding between initiate and validate to compute the shares received by the depositor.

Resolution

Smardex Team: Acknowledged.

L-44 | Insufficient removeBlockedPendingAction Wait

Category	Severity	Location	Status
Validation	● Low	UsdnProtocolCoreLibrary.sol: 358	Resolved

Description

The _removeBlockedPendingAction function may not be invoked within s._lowLatencyValidatorDeadline + 1 hours of the initiation of an action. This is to give users ample time to validate the action before it is removed from the queue.

The current _onChainValidatorDeadline is set to 65 minutes, meaning that once the low latency delay has passed arbitrary users will not have the chance to validate if it is removed using the _removeBlockedPendingAction function at the s._lowLatencyValidatorDeadline + 1 hours time.

Recommendation

Consider if this is acceptable to the protocol. If it is not, consider configuring the _removeBlockedPendingAction function delay to be a function of the low latency delay plus some wait time to allow arbitrary users to validate the action with an on chain oracle before it can be removed.

Resolution

Smardex Team: The issue was resolved in PR#719.

L-45 | LastUpdateTimestamp Errantly Initialized

Category	Severity	Location	Status
Logical Error	● Low	UsdnProtocolCoreLibrary.sol: 56	Acknowledged

Description

In the initialize function the _lastUpdateTimestamp value in storage is assigned to the block.timestamp, however in all other assignments the _lastUpdateTimestamp is assigned with the timestamp corresponding to the provided price.

As a result there may be unexpected edge cases directly after initialization where subsequent actions use a price which is more recent than the one used on initialization but the lastPrice is not updated because it has been assigned to the timestamp of initialization which is more recent.

For example:

- Initialization occurs at t = 100
- The initialization uses a price from t = 50
- A subsequent open order is initiated with a price from t = 60
- The open order is validated with a price from t = 90

In this example the price used to validate the open order is technically the latest price, but will not be treated as such. This can cause unexpected behavior for the PnL correction which occurs during the validation of an open order. As well as prevent liquidations from occurring as the price is not deemed recent.

Recommendation

Consider initializing the _lastUpdateTimestamp to the timestamp of the price being used upon initialization.

Resolution

Smardex Team: Acknowledged.

L-46 | Potential Reverts From New Actionable Actions

Category	Severity	Location	Status
DoS	● Low	UsdnProtocolVaultLibrary.sol: 435	Resolved

Description

Whenever a user initiates or validates an action, they must also validate the next actionable action in the queue, if there is one. A corresponding `previousActionData` must be sent as a parameter.

However, there may be no actionable actions when the transaction is submitted, but an action can become actionable when the transaction is executed. If the user sends empty `previousActionData`, the entire transaction will revert.

Recommendation

Consider allowing empty previous action data to be empty and avoid triggering `_executePendingActionOrRevert` in this case.

This way users can avoid potentially unexpected reverts when the queue is empty and may become populated depending on the block in which the transaction is recorded.

Resolution

Smardex Team: The issue was resolved in PR#701.

L-47 | Incorrect Event Emitted For Close Actions

Category	Severity	Location	Status
Events	● Low	UsdnProtocolActionsLongLibrary.sol: 802	Acknowledged

Description

There is an edge case where the validation price of open position is below the liquidation price, but the tick was never liquidated. In case of partial closing positions, the protocol will incorrectly emit `LiquidatedPosition` event, as the user can still create a new close position action.

Recommendation

Only emit the event if the validate action is a complete close position amount. Otherwise consider indicating that this was a partial position liquidation.

Resolution

Smardex Team: Acknowledged.

L-48 | Burned SDex Unforgiven On Removal

Category	Severity	Location	Status
Unexpected Behavior	● Low	UsdnProtocolCoreLibrary.sol: 363	Acknowledged

Description

In the _removeBlockedPendingAction function the deposited asset amount is returned to the to address, however the burned sDex amount is not.

Recommendation

Be aware of this loss of burned sDex for the depositor and consider making them whole if this situation with the _removeBlockedPendingAction function arises.

If this should be resolved manually on a case-by-case basis, then consider storing the amount of sDex burned on the deposit action and minting this amount back to the to address in the _removeBlockedPendingAction if cleanup is true.

Resolution

Smardex Team: Acknowledged.

L-49 | Imbalance Variables Updates

Category	Severity	Location	Status
Validation	● Low	USDNProtocolFallback.sol: 665	Resolved

Description

Variables related to imbalance are crucial to protect delta-neutrality of USDN and also prevent different attack vectors. Hence it is utmost important to be careful when changing these.

While function setExpoImbalanceLimits have some checks in order to limit these values, there are still some values that are possible but shouldn't be possible.

For example, the RebalancerCloseExpoImbalanceLimitBps should never be bigger than longImbalanceTargetBps. But this is not checked in the setter function. If this ever happens, then it would be possible to leave the rebalancer position immediately after rebalancing happened.

Recommendation

Consider adding validation for this specific case and general upper and lower limits as necessary to avoid any risk of potential misconfiguration.

Resolution

Smardex Team: The issue was resolved in PR#703.

L-50 | Preview Functions Not Accurate

Category	Severity	Location	Status
Unexpected Behavior	● Low	UsdnProtocolFallback.sol: 42	Resolved

Description

Both previewDeposit and previewWithdraw use the exact price passed for calculations, but confidence interval is not applied. Additionally, if the real validate action triggers a rebase, the tokens minted will be completely different.

Therefore, these preview functions will not accurately calculate real amounts. If using these functions to determine the sharesOutMin or amountOutMin, the transaction might revert.

Recommendation

If this is intended behavior, document it to the users, so they are aware that these functions are only for rough estimates.

Resolution

Smardex Team: The issue was resolved in PR#700.

L-51 | Liquidations May Occur Earlier Than Expected

Category	Severity	Location	Status
Logical Error	● Low	TickMath.sol: 94	Acknowledged

Description

Users might expect they will be liquidatable only when the price is exactly reaches to the liquidation price. But because of rounding down of prices, positions can be liquidated earlier than expected.

Recommendation

Inform users about this behavior and if possible, share the exact price at which the liquidation will occur.

Resolution

Smardex Team: Acknowledged.

L-52 | Protocol Avoids Correct TVL Growth

Category	Severity	Location	Status
Warning	● Low	Global	Acknowledged

Description

According to current initialization parameters, the protocol will start in a balanced state and with a 5% maximum imbalance limit which if reached will prevent any actions until the imbalance is corrected.

With the current initialization parameters, a position with 2 wstETH and around 3.7x leverage will be enough to put the protocol to the limit which after that traders will need to wait for new vault deposits to be able to open new long positions.

This process will continue for a long time which will create a stair-stepping case to scale TVL and will slow down the scaling phase.

Recommendation

Consider these parameters for initial setup with this in mind, especially the maximum imbalance limits which can be bigger initially to make scaling faster.

Resolution

Smardex Team: Acknowledged.

L-53 | Imbalance Breached While Validating Open

Category	Severity	Location	Status
Logical Error	● Low	USDNProtocolLongLibrary: 971	Resolved

Description

The leverage of a position is recalculated in validation according to the new price. Which will change the exposure of the position and will change the imbalance amount in the protocol.

While maximum imbalance checks are performed during initiate actions, it is not checked during validation actions. This can naturally lead to imbalance breaches.

Recommendation

Be aware of this risk and document it for integrators and users.

Resolution

Smardex Team: The issue was resolved in PR#712.

L-54 | Validators With No Receive Function

Category	Severity	Location	Status
Documentation	● Low	Global	Resolved

Description

If a user uses an address that can't receive ether as a validator, the validator won't be able to validate the action.

This itself is not a problem for the protocol considering these actions will become actionable by anyone after some time. But it will lead to loss of funds for the user who is unaware of this fact.

Recommendation

Inform users about this behavior properly and let them be sure that the address used for validator will be able to receive ether.

Resolution

Smardex Team: The issue was resolved in PR#704.

L-55 | USDN May Temporarily Lose Peg

Category	Severity	Location	Status
Warning	● Low	USDNProtocolLongLibrary.sol: 472	Resolved

Description

The variable `_usdnRebaseInterval` is not used in initial deployment. If this variable non-zero, then it will be possible for USDN price to increase unexpectedly temporarily.

This can happen because although there is a liquidation, aforementioned variable can prevent rebases which will inflate the value of USDN.

Users will be able to arbitrage the USDN price during this time, as the next rebase will push price down to target again.

Recommendation

Consider always triggering the rebase if the price is recent.

Resolution

Smardex Team: The issue was resolved in PR#653.

L-56 | Funding Retroactively Affected By Admin Update

Category	Severity	Location	Status
Logical Error	● Low	UsdnProtocolFallback.sol: 598	Acknowledged

Description

The SET_PROTOCOL_PARAMS_ROLE is allowed to make critical storage variable changes using admin functions. Specifically, setFundingSF, setEMAPeriod, and setProtocolFeeBps involve updates that will impact the outcome of PnL and Funding calculations.

Therefore, failing to execute applyPnlAndFunding before the admin update will cause the following values to accrue based on the new state variables, when they should instead accrue based on the old values until the changes occur:

- _fundingPerDay is calculated based on s._fundingSF
- _calculateFee uses s._protocolFeeBps to calculate protocol fee on funding value.
- _updateEMA relies on s._EMAPeriod for the EMA calculation.

Recommendation

Before making critical state changes, consider triggering an update using UsdnProtocolCoreLibrary._applyPnlAndFunding.

Resolution

Smardex Team: Acknowledged.

L-57 | Sparse Queue After Pyth Validation Ends

Category	Severity	Location	Status
Unexpected Behavior	● Low	UsdnProtocolVault.sol: 58	Partially Resolved

Description

The getActionablePendingActions returns the actionable pending actions, up to a max of 20. In case we have a pending action that needs Chainlink validation, this action will stay in the front of the queue for some decent amount of time.

Therefore, the pending action can make the queue grow, as long as the first and last action in the queue are not actionable. There could be cases where a queue of 20+ actions are pending (even if most actions in the middle are empty or cleared).

If an action in the queue becomes actionable, and the position in the queue is at an index greater than 20, the getActionablePendingActions will not catch this action and may even return an empty array.

Additionally, the internal _executePendingAction will also fail to execute any actionable action, as the max amount of items read from the queue is 20 (MAX_ACTIONABLE_PENDING_ACTIONS).

Recommendation

Consider increasing the max actionable pending actions value, or refactoring the function so it can ignore empty actions in the queue so it's easier to find the actionable actions.

Resolution

Smardex Team: The issue was resolved in PR#701.

L-58 | Instantaneous Actionable Actions

Category	Severity	Location	Status
Unexpected Behavior	● Low	UsdnProtocolFallback.sol: 522	Acknowledged

Description

Validator deadline timestamps can be updated using `setValidatorDeadlines` admin function.
The new value for `_lowLatencyValidatorDeadline` state variable can range between 60 seconds and the `lowLatencyDelay` (20 minutes).

Reducing this deadline may cause some pending actions to become actionable instantly. Any user can then validate the pending action and claim the security deposit.

Recommendation

Document this behavior to the users, making sure they are aware of deadline updates and potentially lose their security deposits if they don't validate their actions on time.

Resolution

Smardex Team: Acknowledged.

L-59 | Missing Validation For Admin Functions

Category	Severity	Location	Status
Validation	● Low	Global	Resolved

Description

Variables that can change with setter functions are not validated properly. Using a value that would create problems for these variables can be catastrophic and the effect of it can be instant.

Hence please consider following checks in setter functions:

1- `setOracleMiddleware`: check if the `lowLatencyDelay` in the new middleware is within bounds compared to deadlines.
2- `setRebalancer`: Compare `minLongPosition` to minDepositAssets in new contract.
3- `setValidatorDeadlines`: These should give more room to [`LowlatencyValidatorDeadline`, `lowLatencyDelay`], as it can be 1 second apart and lead to problems.
4- `setMaxLeverage`: Having a max cap of 100x is too much to be handled in Ethereum, which leads to increased risk for bad debt.

Recommendation

Put further validations in the setter functions mentioned above.

Resolution

Smardex Team: The issue was resolved in PR#716.

L-60 | Incorrect Return Value For Failed Actions

Category	Severity	Location	Status
Composability	● Low	UsdnProtocolActionsLongLibrary.sol: 664	Resolved

Description

Initiate actions have a return value `success` that should determine if the action was initiated or not. However, `initiateClose` will return true if the position is liquidated, which is unexpected. External actors may have issues integrating USDN protocol, creating unexpected behaviors.

Recommendation

Consider returning false if the action was not initiated.

Resolution

Smardex Team: The issue was resolved in PR#692.

L-61 | Leverage DoS With lastPrice Update

Category	Severity	Location	Status
DoS	● Low	UsdnProtocolLongLibrary.sol: 953	Acknowledged

Description

Leverage for the position will be calculated according to the lastPrice variable in the protocol. When a user provides a non recent price from block timestamp t, if there is a price stored in the protocol after timestamp t, leverage will be calculated according to that specific lastPrice.

Hence the user who provides a desired liquidation price with their submitted price in mind, can see that their leverage changed because the price used for the position will be lastPrice.

However, if the user tries to open a position with maximum leverage, it is possible that this action will revert when using lastPrice as leverage will exceed maximum leverage limit.

Recommendation

Document this behavior to the users so that they can act accordingly and prevent themselves from this small DoS edge case.

Resolution

Smardex Team: Acknowledged.

L-62 | Small Rebalancer Positions Are Allowed

Category	Severity	Location	Status
Validation	● Low	UsdnProtocolLongLibrary.sol: 630	Acknowledged

Description

In the _triggerRebalancer function of the UsdnProtocolLongLibrary, the current implementation allows for extremely small positions:

This check permits positions larger than 1/10000th of the _minLongPosition. Such small positions can lead to situations where the position's collateral is less than the rewards a user could receive for liquidating it. This discrepancy creates a potential for bad debt that the vault would have to absorb.

The issue arises from the following factors:

1. The liquidation reward are designed to be profitable by getting larger as gas costs increase.
2. The liquidation logic does have a max reward cap, but this is based on the typical min collateral amount for non-rebalancing users which is much larger than what the rebalancer requires.

Due to these factors small positions made by the rebalancer can lead to small amounts of bad debt accrued over time.

Recommendation

Be aware of the risk of a small amount of bad debt in these rare cases. If this is not acceptable then consider introducing a case where if the remaining collateral is greater than 1/10000th of the _minLongPosition but less than the _minLongPosition then the funds are given back to the Rebalancer instead of seeded into a new position or given to the vault.

Resolution

Smardex Team: Acknowledged.

L-63 | Some ERC20 Tokens Are Incompatible

Category	Severity	Location	Status
Documentation	● Low	Global	Resolved

Description

Throughout the codebase `_asset` amounts are pushed to arbitrary addresses in validation actions.

For wstEth there is no way for these transfers to revert unless they run out of gas, however for tokens which have a blacklist, callback or other unique behaviors when transferring may revert and cause the queue to enter a stuck state.

Recommendation

Be aware that the USDN system is incompatible with assets that have a blacklist, callback or other unique functionality.

Resolution

Smardex Team: The issue was resolved in PR#648.

L-64 | Liquidatable Positions Can Be Transfered

Category	Severity	Location	Status
Documentation	● Low	UsdnProtocolActionsUtilsLibrary.sol	Acknowledged

Description

When the transferPositionOwnership function is called the protocol does not allow a transfer of a position which tick was liquidated.

But it does not check if the position is healthy before by calling _applyPnlAndFundingAndLiquidate. Because of this, it is possible for liquidatable positions to be transferred.

Recommendation

If this is not the expected behavior, consider checking if the position is liquidatable by calling _applyPnlAndFundingAndLiquidate before performing the checks in transferPositionOwnership. Otherwise this finding serves to document this behavior.

Resolution

Smardex Team: Acknowledged.

L-65 | Sandwich Validate Close Pos

Category	Severity	Location	Status
MEV	● Low	UsdnProtocolActionsLongLibrary.sol: 769	Acknowledged

Description

When close position action is validated a stepwise jump in the value of shares happens. The vault balance can increase or decrease based on the price changes between init and validate or if the price falls below the positions liquidation price the full value could be seized by the vault.

As the init call of deposit/withdraw saves the current balances and calculates the share/asset amount received based on that, users are able to sandwich validate close position calls to make profit or avoid losses.

Deposit for profit:

• LP sees that a validate close pos call will increase the vaults balance
• LP front runs the transaction and initiates a deposit to mint shares based on the old balance
• The validate close pos goes through
• The LP validates the deposit and is in instant profit without price changes

Withdraw to avoid loss:

• LP sees that a validate close pos will decrease the vaults balance
• LP front runs the transaction and initiates a withdraw to burn shares based on the old balance
• The validate close pos goes through
• The LP validates the withdraw and avoided paying for the loss and socialized more loss to the other LPs by doing so

Recommendation

This finding serves only to document this behavior. Be aware of these potentially unexpected behaviors and how they could be manipulated.

Resolution

Smardex Team: Acknowledged.

L-66 | Rebalancer Only Triggered On Liquidations

Category	Severity	Location	Status
Logical Error	● Low	UsdnProtocolLongLibrary.sol: 197	Acknowledged

Description

The rebalancer is only potentially trigger if liquidations happened. But because of funding it could be that there is a large enough imbalance to reach the threshold even without liquidations.

Recommendation

Consider rebalancing the pool even if there are no liquidations, when the imbalance is large enough.

Resolution

Smardex Team: Acknowledged.

L-67 | Rebalancing Position Impacts Traders

Category	Severity	Location	Status
Documentation	● Low	UsdnProtocolCoreLibrary.sol: 270	Acknowledged

Description

When rebalances occur, existing traders end up closing with less funds than if there was no rebalance at all. This will result in an inconsistent amount of funds the trader can withdraw.

This is because of truncation that can occur when the trader's leverage is low enough that the long balance exceeds 99% of the exposure. In cases like this, all traders will have some of their long balance truncated and sent to the vault.

This is possible due to the unbounded amount of assets that a user can deposit into the rebalancer, but for truncation to occur, an immense amount of capital is required.

Recommendation

Because the inconsistencies are less than $0.01 in every case besides truncation and for truncation to occur, it would require tens of millions of dollars to be deposited into the rebalancer at once, it is recommended that this be documented and that traders closing amounts and rebalancing activities monitored.

Resolution

Smardex Team: Acknowledged.

L-68 | Sandwich Deposit/Withdraw

Category	Severity	Location	Status
MEV	● Low	GLOBAL	Acknowledged

Description

When users initiates a deposit or withdrawal the current state of the system is saved in the pending action and the user mints shares on validation based on the state at init.

This can be abused by sandwiching a deposit or withdraw validation call if the share price changed significantly between init and validation of the action.

For example:

• A user initiates a deposit of 100 assets at a share price of 1 asset == 1 share
• The user validates the deposit after a while
• An attacker sees that the share price changed to 0.9 assets == 1 share (funding, liquidations with bad debt, ...) and front runs the validation call to initiate a deposit
• The first user's validation call goes through and the user receives 100 shares for depositing 100 assets and these shares are now worth 90 assets
• The validation call of the attacker goes through and the attacker gains more value then if they did not front-run at all as the other user's deposit is socialized among all LPs.

Recommendation

This finding serves only to document this behavior. Be aware of these potentially unexpected behaviors and how they could be manipulated.

Resolution

Smardex Team: Acknowledged.

L-69 | DoS By Occupying The Validator

Category	Severity	Location	Status
Griefing	● Low	UsdnProtocolCoreLibrary.sol: 334-33	Acknowledged

Description

When a user initiates an action and the given validator already has an pending action the call will revert. This can be misused by malicious actors as a DoS attack by front-running other user's transaction and occupying the given validator.

Recommendation

Allow users to disable that other users can set their address as the validator.

Resolution

Smardex Team: Acknowledged.

L-70 | Bad Debt Not Handled In Validate Open Pos

Category	Severity	Location	Status
DoS	● Low	_validateOpenPositionUpdateBalances.sol	Resolved

Description

The validate open position flow does not handle bad debt in the `_validateOpenPositionUpdateBalances` function (does not cap the values at 0). This can lead to an underflow and therefore to a long term DoS as this is a validation function.

Recommendation

Handle bad debt in the validate open position flow.

Resolution

Smardex Team: The issue was resolved in PR#708.

L-71 | Ether Griefing Attack

Category	Severity	Location	Status
Logical Error	● Low	UsdnProtocol/UsdnProtocolFallback.sol: 68-75	Resolved

Description

When a user inits an action and use a different account as validator and this validator has a stale pending open position action, the action is removed and the security deposit is sent to this validator.

This enables a griefing attack for a malicious validator if there is another stale pending open pos action in the system:

• User inits a new action and uses a malicious validator
• The validator has a stale pending open pos action and the security deposit is sent to the validator
• The validator receives the funds and reenters the system by calling `refundSecurityDeposit`
• The `refundSecurityDeposit` function transfer the security deposit value of another pending open pos action to another validator and decreases the balance of the contract by doing so
• The init call of the user goes on and as the balance of the contract decreased by the security deposit in the step before, the users will lose 0.5 ether in the `_refundExcessEther` flow when the current balance of the contract is compared with the one at the start of the init call

Recommendation

Add a reentrancy guard to the `refundSecurityDeposit` function.

Resolution

Smardex Team: The issue was resolved in PR#689.

L-72 | Sandwich Remove Pending Open Pos

Category	Severity	Location	Status
MEV	● Low	UsdnProtocolCoreLibrary.sol: 418	Acknowledged

Description

If a pending open position is removed with the `_removeBlockedPendingAction` function and the position accrued bad debt, the bad debt is applied to the vault.

LPs can see the transaction that would apply bad debt to the vault and front-run it to initiate a withdrawal before.

As the validate withdraw function uses the balance of the vault at init time the LP can avoid paying for the bad debt and socialize more debt to the other LPs by doing so.

Recommendation

This finding serves only to document this behavior. Be aware of these potentially unexpected behaviors and how they could be manipulated.

Resolution

Smardex Team: Acknowledged.

L-73 | Fee Can Be Avoided By Depositing Dust Amts

Category	Severity	Location	Status
Validation	● Low	UsdnProtocolVaultLibrary.sol: 571	Acknowledged

Description

There is no minimum deposit amount implemented in the system, therefore the fee can be avoided by depositing a tiny amount of ether so that the fee rounds down to zero.

This is unlikely to realistically be leveraged in production, however there may be additional unexpected behaviors which arise with small deposits.

Recommendation

Consider implementing a minimum deposit amount.

Resolution

Smardex Team: Acknowledged.

L-74 | Traders Can Choose Prices In Edge Cases

Category	Severity	Location	Status
Validation	● Low	OracleMiddleware.sol: 294	Acknowledged

Description

In the _getValidateActionPrice function, if the _lowLatencyDelay(20mins) passed since the init, the chainlink price at init + _lowLatencyDelay(20mins) is used instead of the price at init + _validationDelay(24secs).

Therefore users can wait and see if the price moves in their favor and depending on that decide if they execute it with the price at init + 24s or at init + 20m. The only mechanism that prevents this are MEV bots that should try to get the security deposit after init + 15m.

If for any reason this does not happen the user can choose between the two prices with a difference of 20 minutes.

Recommendation

Consider always using the validation price at init + validationDelay.

Resolution

Smardex Team: Acknowledged.

L-75 | SDEX Burn Could Be Bypassed

Category	Severity	Location	Status
Validation	● Low	UsdnProtocolVaultLibrary.sol: 679	Resolved

Description

The transferCallback function calls the msg.sender contract and checks if the SDEX amt of the DEAD_ADDRESS increased by the given amt after this call.

If this protocol is deployed twice for example to enable a second asset, the user could re-enter on this callback in the second protocol instance and perform two deposits of equal worth by burning the SDEX amt needed only once and this check will pass as the SDEX balance of the DEAD_ADDRESS increased by the needed amt (but only once instead of twice).

Recommendation

Be aware of this potential issue if the protocol will be deployed multiple times in the future and consider transferring the assets to different dead addresses in that case.

Resolution

Smardex Team: The issue was resolved in PR#790.

L-76 | Imbalance Checks Wrong In Edge Cases

Category	Severity	Location	Status
Validation	● Low	UsdnProtocolVaultLibrary.sol: 1156-1170, UsdnProtocolLongLibrary.sol: 977-984	Resolved

Description

The `_checkImbalanceLimitWithdrawal` function passes if the calculated `newVaultExpo` is < 0 as the `if (newVaultExpo == 0) { revert }` check will pass and the `imbalanceBps` value will be below 0 when it divides by the negative `newVaultExpo` value.

It can be in edge cases that the `newVaultExpo` is below 0:

• two LPs are in the system
• the first LP initiates a full withdrawal and the `_pendingBalanceVault` is decreased based on the current state in the system
• before the withdrawal is validated bad debt is applied to the vault and therefore the `_balanceVault` is decreased
• the second LP initiates a full withdrawal therefore two withdrawal values are decreased from the `_balanceVault` in the `newVaultExpo` calculation which are combined bigger than the `_balanceVault` value, as the first withdrawal value saved in `_pendingBalanceVault` was calculated at a time where the vault hold more value than it does now

This edge case allows to initiate a withdrawal for more funds than there are left in the vault, which will most likely lead to a long-term DoS as the validation call reverts. The same behaviour can be seen in the `_checkImbalanceLimitOpen` function.

Recommendation

Change the `if (newVaultExpo == 0) { revert }` check to `if (newVaultExpo <= 0) { revert }`.

Resolution

Smardex Team: The issue was resolved in PR#709.

L-77 | Stepwise Jump In sdexToBurn Calc

Category	Severity	Location	Status
Logical Error	● Low	UsdnProtocolVaultLibrary.sol: 591-598	Acknowledged

Description PoC

The sdexToBurn calculation uses the estimated USDN tokens based on the saved divisor instead of the current state. As the saved divisor is usually updated in intervals it can be stale at any time. This leads to a stepwise jump in the needed amount of SDEX to burn. The higher the divisor the less SDEX needs to be paid.

For example:

• block 1:
• Alice deposits 10 wstETH
• block 2:
• divisor is updated
• block 3:
• Bob deposits 10 wstETH

In this example, Bob needs to burn more SDEX than Alice without any price changes in wstETH or SDEX. Therefore users can save SDEX by depositing before a divisor update. Aside from burning too few fees, another issue can occur. If the previewDeposit function is used to calculate the needed sdexToBurn_ for a deposit and this amount is approved to the protocol an unexpected revert could happen if the divisor is updated before the deposit is executed.

For example by a liquidate call that ignores the divisor interval:

• User calls previewDeposit and approves the returned sdexToBurn_ amount to the protocol
• The User creates a deposit transaction
• A liquidator calls liquidate with more gas in the same block
• The liquidate call gets executed first and the deposit call reverts as not enough SDEX is approved.
This can lead to unexpected reverts from time to time.

Recommendation

Options that can be considered to resolve this are:
• Calculate the current divisor and use it to calculate the sdexToBurn_
• Ignore the divisor interval not only in liquidate but also in the deposit flow
• Calculate the sdexToBurn_ fee based on the value of the deposited asset instead of the USDN tokens to mint

Resolution

Smardex Team: Acknowledged.

L-78 | Rebalancer tradingExpoToFill Not Entirely Filled

Category	Severity	Location	Status
Unexpected Behavior	● Low	UsdnProtocolLongLibrary.sol: 1056, 1068	Acknowledged

Description

In the _calcRebalancerPositionTick function if the liquidationPenalty stored on the liquidation tick of the new rebalancer position does not match the currentLiqPenalty then the liqPriceWithoutPenalty is adjusted to use the correct liquidation penalty for that tick. This may result in a lower unpenalized liquidation price when the penalty stored on the tick is higher than the current penalty.

This results in a lower leverage and lower exposure for the resulting rebalancer position than was desired. Ultimately this means the entire tradingExpoToFill will not be met as the actual liquidation price has been reduced from the idealLiqPrice which corresponds to the tradingExpoToFill. In this case the following if case will trigger in order to fill the gap to the longImbalanceTargetBps by incrementing the liquidation tick by a tick spacing.

However in the case where the highestUsableTradingExpo was used then this case will not trigger as it requires that data.highestUsableTradingExpo != tradingExpoToFill. This assumes that the highestUsableTradingExpo has been entirely used by the new rebalancer position, but this may not be the case in the edge case where the liquidation penalty is higher than expected.

When this edge case occurs the if case should trigger when there is a significant dearth from the highestUsableTradingExpo. This way the longImbalanceTargetBps has a higher chance of being reached when the tradingExpoToFill was not met in these highestUsableTradingExpo cases.

Recommendation

Consider changing the data.highestUsableTradingExpo != tradingExpoToFill requirement for the tick spacing increment case to be based upon how much exposure is actually taken up by the new rebalancer position. The most exact validation would be to compare the highestUsableTradingExpo to the exposure of the position after it has been moved up by one tick to see if this increment can reasonably occur within the max leverage.

This would be implemented in pseudocode as:

```
        positionExposureIncrement =  0.0202 * currentPrice * liquidationPriceWithoutPenalty / ((currentPrice -
liquidationPriceWithoutPenalty) * (currentPrice - 1.0202 * liquidationPriceWithoutPenalty)) if
(data.highestUsableTradingExpo < posData_.totalExpo + posData_.totalExpo * positionExposureIncrement && ...){...}
```

It's worth noting that this would not account for a change in the liquidation penalty at the new tick. This exact validation is likely too complex to be worth implementing, but serves as an example to show ho this edge case might be addressed.

In practice a less accurate validation could be acceptable, or this issue may be simply acknowledged as an acceptable inaccuracy of the rebalancer.

Resolution

Smardex Team: Acknowledged.

L-79 | Funding Is Charged During Paused States

Category	Severity	Location	Status
Logical Error	● Low	UsdnProtocolFallback.sol: 794	Resolved

Description

Funding fees are charged in order to incentivize users to balance the protocol. However, when the system is in a paused state, these fees are still being charged. When the system is unpaused and a recent price is provided, funding will be charged to the entire paused elapsed time.

Recommendation

Freeze funding fees during paused states. It can be achieved via applying funding when pausing the protocol and updating the last timestamp when unpausing.

Resolution

Smardex Team: The issue was resolved in PR#678.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits