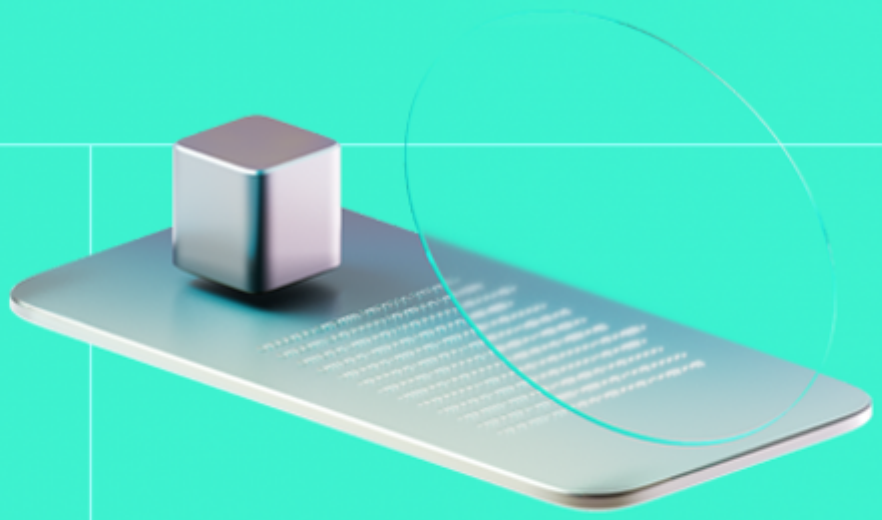
HACKEN

Smart Contract Code Review And Security Analysis Report

Customer: Dexalot

Date: 27/03/2024



We express our gratitude to the Dexalot team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

Dexalot is a revolutionary decentralized exchange that features an on-chain Central Limit Order Book (CLOB), providing a trading experience similar to traditional centralized exchanges. Operating on its own dedicated Avalanche subnet and expanding cross-chain, Dexalot offers enhanced trading efficiency and accessibility across multiple blockchain ecosystems.

Platform: EVM

Language: Solidity

Tags: DEX; Bridge; Upgradable; Centralization

Timeline: 15/02/2024 - 27/03/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/Dexalot/contracts
Commit	2492aa2

Audit Summary

10/10	10/10	96.67%	10/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 9.9/10

The system users should acknowledge all the risks summed up in the risks section of the report

4	1	0	3
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	1
Low	3

Vulnerability	Status
F-2024-1144 - Chainlink's latestRoundData might return stale or incorrect results	Mitigated
F-2024-1156 - Unrestricted Modification of Amplification Coefficient (A) in InventoryManager Affecting Withdrawal Fees	Mitigated
F-2024-1289 - Expanded Access Control Risks in depositToken Functions Allowing Unauthorized Token Transfers	Mitigated
F-2024-1159 - Insufficient Fee Withdrawal Flexibility in Multi-Chain Environment	Fixed

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Dexalot
Audited By	Ivan Bondar
Approved By	Grzegorz Trawinski
Website	https://dexalot.com/
Changelog	07/03/2024 - Preliminary Report; 27/03/2024 - Final Report

Table of Contents

System Overview

Dexalot is a pioneering decentralized exchange designed to emulate the user experience of traditional centralized exchanges (CEX) while upholding the principles of decentralization and transparency. It operates on the Avalanche C-Chain and its own Avalanche subnet, Dexalot, and plans to extend its presence to Ethereum, Arbitrum, and Gunzilla chains. This multi-chain approach positions Dexalot as a versatile and accessible platform for a broad user base.

Key Features of Dexalot:

- Avalanche Subnet Deployment: Dexalot leverages a dedicated Avalanche subnet, optimizing for user-friendliness, reduced transaction fees, and quick transaction finality. This approach ensures a seamless trading experience akin to CEXs.
- Expansion to Multiple Chains: In addition to Avalanche, Dexalot is set to be available on Ethereum, Arbitrum, and Gunzilla chains, broadening its reach and interoperability in the DeFi ecosystem.
- On-Chain Central Limit Order Book (CLOB): The platform's hallmark feature is its on-chain CLOB, which mirrors the operational dynamics of CEXs. Traders can place limit orders, with the flexibility of immediate execution or addition as new maker orders in the order book.
- ERC20 and Native Currency Trading: Dexalot supports a diverse range of trading pairs, including ERC20 tokens and native chain currencies like AVAX, facilitating a comprehensive trading environment.

The files in the scope:

- **Exchange.sol** - An abstract administrative contract, inherited by ExchangeMain and ExchangeSub. It manages access levels using OpenZeppelin's AccessControl roles, with roles like DEFAULT_ADMIN_ROLE and AUCTION_ADMIN_ROLE.
- **ExchangeMain.sol** - The mainnet version of the Dexalot Exchange, serving as the DEFAULT_ADMIN for the PortfolioMain contract.
- **ExchangeSub.sol** - The subnet version of the Dexalot Exchange. It carries all functions for AUCTION_ADMIN, and is the DEFAULT_ADMIN for both PortfolioSub and TradePairs contracts.
- **Portfolio.sol** - An abstract contract shared by PortfolioMain and PortfolioSub. It manages cross-chain asset handling, bridging assets between the mainnet and subnet.
- **PortfolioMain.sol** - Operates on the mainnet, prevalidating PortfolioSub checks and managing deposits to the subnet.
- **PortfolioSub.sol** - Handles deposits from the mainnet, withdrawal requests, and token transfers between traders on the subnet. It interacts with the GasStation for gas token management and supports a feature called AutoFill to ease gas token handling.
- **PortfolioSubHelper.sol** - enhances the Dexalot platform's functionality by managing special accounts that can have different trading rates and by ensuring smooth transition and compatibility with multi-chain trading post-upgrade
- **TradePairs.sol** - Manages trade pairs data structures and functions on Dexalot. Each trade pair is denoted as BASEASSET/QUOTEASSET, like AVAX/USDT. The ExchangeSub contract must have DEFAULT_ADMIN_ROLE on TradePairs, and TradePairs should have EXECUTOR_ROLE on OrderBooks.
- **PortfolioBridgeMain.sol** - Serves as a bridge aggregator and message relayer for the mainnet. It uses multiple bridges, with LayerZero as the default provider. The contract does not hold user funds but manages bridge fees. It also serves as a symbol mapper for multi-chain symbol handling.
- **PortfolioBridgeSub.sol** - Similar to PortfolioBridgeMain, this contract aggregates and relays messages for the subnet. It implements volume caps and delayed transfers. The contract maps symbols received into subnet symbols and vice versa for multichain implementations.
- **LzApp.sol** - An abstract Layer Zero contract that forms the backbone of cross-chain communication functionalities in the Dexalot ecosystem. This contract is crucial for implementing Dexalot's cross-chain features, as it is extended by the PortfolioBridgeMain contract for specific implementation within Dexalot.
- **MainnetRFQ** - Utilizes prices from the Dexalot subnet to offer token swaps on the C-Chain. Users interact with the RFQ API for swaps, and the contract handles potential slippage and quote expiry adjustments.
- **DelayedTransfers.sol** - Manages delayed transfers for withdrawals in the PortfolioBridgeSub contract. It checks volume and threshold limits for withdrawals and implements volume caps per epoch for each token.
- **InventoryManager.sol** - Manages the inventory of tokens on the subnet and calculates withdrawal fees. The inventory, stored by subnet symbol and symbolId, is updated by the PortfolioBridgeSub contract. Withdrawal fees are calculated using the InvariantMathLibrary, which employs a stableswap invariant formula based on the quantity requested, the current inventory in the requested chain, and the total inventory across all chains.

- **InvariantMathLibrary.sol** - A library for calculating withdrawal fees of the same asset across different chains using a stableswap invariant combining constant product and sum formulas. It is used in conjunction with the InventoryManager contract and employs Newton's method for approximations.
- **UtilsLibrary.sol** - Provides common utility functions used across Dexalot's smart contracts, offering simple, pure functions for use in other contracts.
- **IDelayedTransfers.sol**: Defines the structure for delayed transfers functionalities, particularly in the context of withdrawal limits and volume caps.
- **IERC1271.sol**: A standard interface for contracts to interact with digital signatures, allowing them to validate signatures in a generic manner.
- **ILayerZeroUserApplicationConfig.sol**: Part of LayerZero's cross-chain communication protocol, this interface defines configurations for user applications to interact with LayerZero.
- **ILayerZeroEndpoint.sol**: Another interface in the LayerZero ecosystem, focusing on the endpoint functionalities for enabling cross-chain interactions.
- **ILayerZeroReceiver.sol**: Specifies the interface for receiving cross-chain messages within the LayerZero framework, crucial for decentralized communication between different blockchains.
- **IMainnetRFQ.sol**: Interface for the MainnetRFQ on the mainnet, enabling token swaps based on prices from the Dexalot subnet.
- **IPortfolio.sol**: Defines the general structure for portfolio management in Dexalot, encompassing token holdings and trading functionalities.
- **IPortfolioBridgeSub.sol**: Specific to the subnet, this interface details functions for bridging and message relaying in the Dexalot subnet environment.
- **IPortfolioMain.sol**: The mainnet counterpart for portfolio management in Dexalot, outlining the functionalities specific to the Avalanche mainnet.
- **IPortfolioSub.sol**: Interface for managing portfolios specifically within the Dexalot subnet, focusing on the unique aspects of subnet operations.
- **IBannedAccounts.sol**: Specifies functions for managing banned accounts, likely used to maintain security and integrity within the Dexalot ecosystem.
- **IPortfolioBridge.sol**: General interface for bridging functionalities within Dexalot, allowing for efficient and secure cross-chain operations.
- **IPortfolioSubHelper.sol**: Aids in managing portfolios on the subnet, providing additional support and utility functions.
- **IPortfolioMinter.sol**: Focuses on the minting aspects within the portfolio, likely relating to the creation of new tokens or assets in the Dexalot environment.
- **IGasStation.sol**: An interface designed to manage gas-related operations, potentially for transactions and smart contract executions on Dexalot.
- **ITradePairs.sol**: Outlines the structure for trade pair functionalities in Dexalot, central to trading operations and market management.
- **IInventoryManager.so**: Details the interface for managing inventory, particularly in the context of withdrawal fees and asset management across chains.

Privileged roles

- **ExchangeMain.sol:**
  - **DEFAULT_ADMIN_ROLE:**
    - **Admin Management:** Add or remove DEFAULT_ADMIN_ROLE to/from admins.
    - **Auction Management:** Add or remove AUCTION_ADMIN_ROLE to/from admins.
    - **Exchange Management:** Pause trading or the entire exchange for upgrades.
    - **Portfolio Management:** Set the Portfolio contract address and pause/unpause it.
    - **Price Feed Management:** Set the address for the Price Feed contract.
    - **Token Management:** Add tokens to the PortfolioMain contract, including details like symbol, token address, chain ID, decimals, fee, and virtual status.
  - **AUCTION_ADMIN_ROLE:**
    - **Token Configuration:** Add tokens with specific parameters.
    - **Contract Management:** Add or remove trusted contracts to/from the PortfolioMain contract.
- **ExchangeSub.sol:**
  - **DEFAULT_ADMIN_ROLE:**
    - **Admin Management:** Add or remove DEFAULT_ADMIN_ROLE to/from admins.
    - **Portfolio Management:** Set Portfolio contract address and pause/unpause it.
    - **Exchange Management:** Pause trading or the entire exchange for upgrades.
    - **Contract Address Management:** Set OrderBooks and TradePairs contract addresses.

        - **Trading Configuration:** Pause trading, pause individual trade pairs, and set trading parameters like min/max trade amounts and auction modes.
    - **AUCTION_ADMIN_ROLE:**
        - **Auction Management:** Add or remove AUCTION_ADMIN_ROLE to/from admins.
        - **Token Management:** Add tokens to the PortfolioSub contract, including details like symbol, token address, chain ID, decimals, fee, and virtual status.
        - **Trade Pair Management:** Add new trading pairs and update auction modes, rates, auction prices, and min/max trade amounts for trade pairs.
        - **Order Matching:** Match auction orders for a specific trade pair.
        - **Randomized Auction Closing Sequence**: Emits the result of a coin flip based on the latest AVAX/USD price. This function contributes to the randomness in auction closures, ensuring fairness and preventing potential gaming of the auction process.
- **PortfolioMain.sol:**
    - **DEFAULT_ADMIN_ROLE:**
        - **Role Management:** Add or remove admin roles. Ensures at least one admin remains.
        - **Portfolio Bridge Management:** Set and pause the PortfolioBridge contract. Manage bridge providers.
        - **Token Management:** Add or remove tokens. Update token details after upgrades. Manage bridge parameters for specific tokens.
        - **Contract Operations:** Pause or unpause contract. Set minimum deposit multiplier.
        - **Trusted Contract Management:** Add or remove trusted contracts.
        - **Banned Accounts Management:** Set BannedAccounts contract.
        - **Bridge Fee Collection:** Collect bridge fees for both ERC20 tokens and native currency.
    - **PORTFOLIO_BRIDGE_ROLE:**
        - **Cross-Chain Transfer Processing:** Manage and process cross-chain transfer payloads. Authorized to set bridge parameters including fees, gas swap ratios, and usage flags for gas swap.
- **PortfolioSub.sol:**
    - **DEFAULT_ADMIN_ROLE:**
        - **Portfolio Bridge Management:** Set the PortfolioBridge contract and manage bridge parameters.
        - **Contract Operations:** Pause or unpause the contract, manage deposit pauses, and withdraw accumulated fees.
        - **Token Management:** Add or remove tokens and set specific token parameters.
        - **Auxiliary Contract Management:** Set PortfolioSubHelper, GasStation, Treasury, and PortfolioMinter contracts.
        - **Auction Mode Management:** Enable or disable auction mode for tokens.
    - **PORTFOLIO_BRIDGE_ROLE:**
        - **Cross-Chain Transfer Processing:** Handle processing of cross-chain transfer payloads.
    - **EXECUTOR_ROLE:**
        - **Trade Execution and Management:** Manage trade executions, autofill operations, and adjust available assets for traders.
- **PortfolioSubHelper.sol:**
    - **DEFAULT_ADMIN_ROLE:**
        - **Admin Account Management for Rates:** Add or remove admin accounts related to rate adjustments, typically tied to specific organizations.
        - **Rebate Account Management:** Add rebate accounts with preferential rates for specific trade pairs; remove trade pairs from existing rebate accounts.
        - **Token Convertibility Post-Upgrade:** Manage the list of convertible tokens to support multi-chain trading post-March 2024 upgrade. This includes adding and removing tokens from the convertible symbol mapping, essential for renaming symbols like BTC.b, WETH.e, and USDt to BTC, ETH, USDT for multichain trading compatibility.
- **PortfolioBridgeMain.sol:**
    - **DEFAULT_ADMIN_ROLE:**
        - **Bridge Provider Management:** Enable/disable bridge providers and set default bridge providers.
        - **Cross-Chain Communication Setup:** Set trusted remote addresses and default destination chains for cross-chain communication.
        - **Resource Allocation:** Configure gas for destination chains and determine who pays the fees.
        - **Contract Management:** Pause/unpause contract operations, set Portfolio and MainnetRFQ contract addresses.
        - **Fee Refunds:** Facilitate refunding of native currency held by the contract.
        - **LayerZero Endpoint Configuration:** Set the LayerZero Endpoint address.

        - **LayerZero Configuration Management:** Set various configuration options for LayerZero User Application, including send and receive message versions.
        - **Force Resume Receive:** Ability to force resume receive functionality in stuck scenarios using LayerZero.
        - **Retry Payload:** Retry stuck messages in LayerZero bridge.
    - **BRIDGE_USER_ROLE:**
        - **Contract Operation Control:** Grant the ability to pause and unpause the bridge operations.
        - **Cross-Chain Message Handling:** Send cross-chain messages and manage user fee payments for these messages.
        - **LayerZero Message Sending:** Internal function for sending messages via LayerZero.
    - **BRIDGE_ADMIN_ROLE:**
        - **Bridge Parameter Configuration:** Set bridge parameters for different tokens.
        - **Bridge Message Recovery:** Handle stuck messages in the bridge with retry or destructive recovery options.
        - **LayerZero Destructive Message Recovery:** Utilize the destructive option for stuck messages in LayerZero bridge as a last resort.
- **PortfolioBridgeSub.sol:**
    - **DEFAULT_ADMIN_ROLE:**
        - **Bridge Provider Management:** Enable or disable bridge providers; set default bridge providers for cross-chain communication.
        - **Cross-Chain Communication Configuration:** Set trusted remote addresses, default destination chains, and gas allocations for cross-chain transactions.
        - **Token Management and Renaming:** Add or remove tokens; rename tokens for cross-chain communication.
        - **Contract Configuration:** Set addresses for DelayedTransfers, InventoryManager, and other key contracts in the bridge ecosystem.
        - **Resource Allocation:** Configure bridge fees and gas swap ratios; decide whether users or the bridge pays fees.
        - **Fee Refunds:** Enable the refund of native currency held by the contract.
    - **BRIDGE_ADMIN_ROLE:**
        - **Delayed Transfer Execution:** Manage and execute delayed transfers for cross-chain transactions.
        - **Bridge Message Recovery:** Manage stuck messages in the bridge, including retrying or destructively recovering funds.
    - **BRIDGE_USER_ROLE:**
        - **Contract Operation Control:** Pause and unpause bridge operations.
        - **Cross-Chain Message Handling:** Send cross-chain messages, manage user fee payments, and execute delayed transfers.
- **DelayedTransfers.sol:**
    - **DEFAULT_ADMIN_ROLE:**
        - **Threshold Management:** Set and check delay thresholds for different tokens, crucial for controlling the timing of transfers.
        - **Delay Configuration:** Configure the delay period for transfers, key to managing the execution timeline of transactions.
        - **Transfer Execution:** Execute delayed transfers, ensuring that transactions are processed after the specified delay period.
        - **Epoch Configuration:** Set the epoch length, fundamental for volume control and managing transaction flows within a specified timeframe.
        - **Volume Cap Management:** Establish volume caps for tokens, a vital aspect of controlling the quantity of token transfers within an epoch.
        - **Volume Update:** Update the volume for tokens, an essential function for managing withdrawals and maintaining the balance within the system.
- **TradePairs.sol:**
    - **DEFAULT_ADMIN_ROLE:**
        - **Trade Pair Management:** Add or remove trade pairs; pause or unpause trade pairs; set auction modes, trade amounts, and rates.
        - **Order Handling:** Pause adding orders, set post-only mode for testing, update rate types (Maker/Taker).
        - **Display Configuration:** Adjust display decimals for base or quote assets in a trade pair.
        - **Market Management:** Set allowed slippage percent for market orders.
        - **Order Book Administration:** Perform unsolicited order cancellations in case of delisting a trade pair.
    - **EXCHANGE_ROLE:**
        - **Trade Pair Setup:** Responsible for adding new trade pairs with specific token details, display decimals, trade amount limits, and auction modes.

- **MainnetRFQ.sol:**
  - **DEFAULT_ADMIN_ROLE:**
    - **Portfolio Bridge Management:** Set the PortfolioBridge contract address and configure the PortfolioMain contract.
    - **Contract Management:** Pause or unpause contract operations.
    - **Admin Account Management:** Add or remove admin accounts with DEFAULT_ADMIN_ROLE, ensuring essential administrative control.
  - **REBALANCER_ADMIN_ROLE:**
    - **Rebalancer Management:** Add or remove rebalancer accounts, key to managing the rebalancing of assets within the system.
    - **Order Expiry Management:** Update the expiry of swap orders, crucial for maintaining the timeliness and relevancy of trading activities.
    - **Asset Claim:** Claim balance for specific assets, either individually or in batches, vital for asset management and redistribution.
- **InventoryManager.sol:**
  - **PORTFOLIO_BRIDGE_ROLE:**
    - **Inventory Increment:** Increment the inventory of a token, crucial for managing token stock.
    - **Inventory Decrement:** Decrement the inventory of a token, necessary for adjusting stock levels.
    - **Inventory Removal:** Remove a token from the inventory, essential for inventory control.
    - **Symbol Conversion:** Convert the inventory of a token from one subnet symbol to another, important for maintaining accurate inventory across different subnets.
  - **DEFAULT_ADMIN_ROLE:**
    - **Portfolio Bridge Sub Update:** Update the PortfolioBridgeSub contract address, key to keeping the bridge connection current.
    - **Amplification Coefficient Adjustment:** Update the amplification coefficient for the invariant, crucial for controlling inventory dynamics.
    - **Inventory Initialization:** Set host chains' inventories for each token, fundamental for establishing baseline inventory levels.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are detailed.
  - Project overview is detailed
  - All roles in the system are described.
  - Use cases are described and detailed.
  - For each contract, all futures are described.
  - All interactions are described.
- Technical description is limited.
  - Run instructions are provided.
  - Technical specification is provided.
  - The NatSpec documentation is sufficient.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.

Test coverage

Code coverage of the project is **96.67%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is present.
- Interactions by several users are tested.

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **0** medium, and **1** low severity issues, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **9.9**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- **Upgradability and Future Version Modifications**:
  - The contract's upgradable nature means that its implementation can be modified in future versions. This flexibility allows for necessary updates and improvements but also introduces uncertainty for users regarding future changes in the contract's behavior and rules. Users should stay informed about any contract upgrades and understand their implications.
- **Cross-Chain Communication Risks**:
  - Operating across multiple chains (Avalanche, Ethereum, Arbitrum, Gunzilla) introduces risks associated with cross-chain communication and bridging. This includes potential delays, miscommunications, or exploits that target the bridge mechanisms.
- **Governance and Access Control Risks**:
  - The system relies heavily on role-based access control for administrative tasks. Mismanagement of these roles or exploitation of these privileges could lead to unauthorized actions, including pausing key functionalities or misconfiguring critical parameters.
- **Dependency on External Price Feeds**:
  - The reliance on external price feeds (e.g., for the AVAX/USD price in the randomized auction closing sequence) introduces external risk factors, including manipulation or inaccuracies in these feeds.
- **RFQ Withdrawal Risk**:
  - In cases where users haven't received funds on the Dexalot subnet, Dexalot maintains the authority to execute withdrawals from the mainnet. This poses a risk of delayed or failed withdrawals if not properly managed
- **Inventory-Based Withdrawal Limitations**:
  - There's a risk associated with user deposits not being withdrawable to the originally deposited chain if the inventory on that chain drops significantly. This could lead to situations where users are unable to access their funds on their preferred chain, potentially causing liquidity and user experience issues.
- **Decimal Variation in Cross-Chain Transactions**:
  - The protocol operates under the assumption that each token supported has a consistent number of decimal places across different blockchains. This approach is designed to simplify the management of token inventories and transactions across chains. However, it's important to note that certain tokens, such as USDC and USDT, have varying decimal places on different blockchains (e.g., 6 decimals on Ethereum and 18 on BSC - Binance Smart Chain). To ensure consistency and reduce complexity, the protocol currently supports only those tokens that maintain the same number of decimal places across all chains. This limitation means that tokens with variable decimal places on different blockchains, like USDC on BSC, will not be supported at this time. The protocol takes on the responsibility of validating and managing the decimal variations between different blockchains for the tokens it supports.
- **Maximum Fee Cap:**
  - Users are advised to remain aware of the potential for high withdrawal fees, particularly in scenarios where inventory is low on their desired withdrawal chain. They should also pay attention to frontend notifications and consider alternative options provided by the platform to minimize fees. Users should stay informed about any future updates or changes to the fee structure and amplification coefficient management.

Findings

Vulnerability Details

F-2024-1144 - Chainlink's latestRoundData might return stale or incorrect results - Medium

Description:

The `flipCoin` function in the smart contract uses Chainlink's `latestRoundData` to fetch the latest AVAX/USD price for determining the outcome of a pseudo-random coin flip, crucial for the randomized auction closing sequence. However, it's important to note that this method only provides pseudo-randomness. Pseudo-randomness is derived from predictable processes (like market prices) and, thus, can potentially be anticipated or influenced, unlike true randomness which is completely unpredictable.

Affected Code :

```
function flipCoin() external {
(uint80 r, int256 p, bool o) = isHead();
emit CoinFlipped(r, p, o);
}

function isHead() public view onlyRole(AUCTION_ADMIN_ROLE) returns (uint80 r, int
256 p, bool o) {
(r, p, , , ) = priceFeed.latestRoundData(); // example answer: 7530342847
int256 d1 = (p % 1000000) / 100000; // get 6th digit from right, d1=3 for example
int256 d2 = (p % 10000000) / 1000000; // get 7th digit from right, d2=0 for examp
le
o = d1 > d2; // head if d1>d2, 3>0=True=Heads for example
}
```

The function relies on Chainlink's `latestRoundData` to fetch the most recent price data. However, there's a risk associated with potential issues in Chainlink's data update process:

- Chainlink Data Update Issues: If Chainlink encounters delays or failures in updating the AVAX/USD price data, the flipCoin function might operate on stale or incorrect data.
- Arbitrum Sequencer Downtime: The function does not currently account for the status of Arbitrum's sequencer. If the sequencer is down, it could affect the availability and reliability of the data used in the flipCoin function.

Assets:

- ExchangeMain.sol [https://github.com/Dexalot/contracts]

Status: Mitigated

Classification

Severity: Medium

Impact:

Likelihood [1-5]: 3
Impact [1-5]: 4
Exploitability [0-2]: 1
Complexity [0-2]: 0
Final Score: 2.7 (Medium)

Hacken Calculator Version: 0.6

Recommendations

Recommendation:

Incorporate Chainlink VRF for True Randomness: To achieve true randomness and enhance the integrity of the auction process, it is recommended to replace the current pseudo-random method with Chainlink's Verifiable Random Function (VRF). Chainlink VRF provides cryptographic proof of the randomness that is verifiable on-chain, ensuring tamper-proof and fair randomness, essential for critical operations like auctions.

If there is a need to retain the current pseudo-random method, the following enhancements are recommended to improve its robustness:

- Integrate Sequencer Status Check: Ensure data integrity on Arbitrum by integrating a check for the sequencer's active status. This can be achieved by querying the sequencer's latestRoundData and assessing if the data is recent.

```
function isSequencerActive() internal view returns (bool) {
(, int256 answer, uint256 startedAt,) = sequencer.latestRoundData();
return (block.timestamp - startedAt <= GRACE_PERIOD_TIME) || (answer == 1);
}
```

- Improve Chainlink Data Checks: Enhance the data validation from Chainlink to ensure freshness and accuracy. This includes checks for positive price, completed data round, and non-stale data. The staleness threshold should correspond to the heartbeat of the oracle's price feed. This can be found on Chainlink's list of Ethereum mainnet price feeds by checking the "Show More Details" box, which will show the "Heartbeat" column for each feed.

```
require(isSequencerActive(), "L2SequencerUnavailable");
(r, p, , updateTime, answeredInRound) = priceFeed.latestRoundData();
require(p > 0, "Chainlink price <= 0");
require(block.timestamp - updateTime <= heartbeat, "Data is not fresh");
require(answeredInRound >= r, "Stale price");
```

Remediation (Revised commit: a85585e) (Mitigated): The team acknowledged the issue as non-critical and infrequently used, so they chose to defer a comprehensive fix to a later date. They updated the function's NatSpec comments to clearly state the nature of the pseudo-randomness used and its adequacy for the current application, citing additional layers of randomness in the off-chain application.

External References:

- l2-sequencer-feeds

F-2024-1156 - Unrestricted Modification of Amplification Coefficient (A) in InventoryManager Affecting Withdrawal Fees - Low

Description:

The InventoryManager contract in Dexalot plays a critical role in managing token inventory across different chains and calculating withdrawal fees. It uses the InvariantMathLibrary, which employs a stableswap invariant for fee calculation. A key parameter in this process is the amplification coefficient (**A**), significantly influencing the withdrawal fee computation.

Presently, in InventoryManager, both the **`initialize`** and **`updateA`** functions permit the setting and alteration of the amplification coefficient (**A**) without any predefined initial values, upper/lower boundaries, or a delay mechanism. This design allows for setting A to any value initially and subsequently modifying it freely at any point. Such unrestricted control over A's value can lead to unpredictable and significant fluctuations in withdrawal fees for users.

Affected Code:

```
function initialize(address _portfolioBridgeSub, uint256 _A) external initializer
{
require(_portfolioBridgeSub != address(0), "IM-ZADDR-01");
portfolioBridgeSub = IPortfolioBridgeSub(_portfolioBridgeSub);
_grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
_grantRole(PORTFOLIO_BRIDGE_ROLE, _portfolioBridgeSub);
A = _A;
}


function updateA(uint256 _A) external onlyRole(DEFAULT_ADMIN_ROLE) {
require(_A > 0, "IM-ZVFA-01");
A = _A;
}
```

Such unrestricted modification can impact the fee structure significantly, especially given the variable inventory levels on different chains. For example, a sudden change in A could lead to disproportionately high withdrawal fees, adversely affecting user experience and the platform's trustworthiness.

Assets:

- PortfolioSub.sol [https://github.com/Dexalot/contracts]
- InventoryManager.sol [https://github.com/Dexalot/contracts]

Status: Mitigated

Classification

Severity: Low

Impact:

Likelihood [1-5]: 3
Impact [1-5]: 3
Exploitability [0-2]: 2
Complexity [0-2]: 0
Final Score: 2.1 (Low)

Hacken Calculator Version: 0.6

Recommendations

Recommendation:

- Implement Initial Value and Boundaries for A: Define an initial value for A and establish clear minimum and maximum limits. This will ensure that changes in A stay within a controlled range, maintaining stability in withdrawal fees.

- Delay Mechanism for A Updates: Introduce a delay mechanism for changes to A, such as a mandatory waiting period before any update takes effect. This approach provides users with predictability and reduces the risk of abrupt fee changes.
- Maximum Fee Cap: Implement a cap on the maximum withdrawal fee percentage in the `getBridgeFee` function of the PortfolioBridgeSub contract. This cap would limit the bridge fee to a reasonable percentage of the token amount, safeguarding users from excessive fees.
- User Communication: Ensure transparent communication with users regarding any planned changes to A, along with their potential impact. Clear documentation and updates about A's role in fee calculations will enhance user understanding and trust.

Remediation (Revised commit: a85585e) (Mitigated):

- Scaling Factors and Limits: The protocol introduced scaling factors and limits for the amplification coefficient (A) to address the issue of unrestricted modification and to provide more predictability and control over withdrawal fee calculations.
- Maximum Fee Cap (protocol statement): Intended behaviour, for these cases we're going to have a popup on the frontend which tells users the high fees and directs them to other chains where the fee is close to 0 or tells them to reduce the quantity withdrawn for a lower fee. In addition to start with we're going to have a high A to mitigate the high fees until we understand how much will quantity from each chain. To help manage this too we added a scalingFactor so we can say manage the ratio between chains rather than expecting a 50:50 split.

F-2024-1159 - Insufficient Fee Withdrawal Flexibility in Multi-Chain Environment - Low

Description:

The PortfolioSub contract's `withdrawFees` function, initially designed for a single-chain environment, exhibits limitations when adapted to Dexalot's expanding multi-chain operations. This function currently defaults to withdrawing fees to the mainnet, without considering other chains that might host the tokens. Additionaly, The current implementation uses a while loop to iterate through all tokens, potentially leading to high costs, especially when the token list is extensive.

The function's inefficiency arises from iterating over the entire token list to process withdrawals, which can be costly in terms of gas usage. Additionally, the function lacks flexibility as it defaults to withdrawing fees to the mainnet for every token, without considering the possibility of withdrawing to different chains based on specific needs or inventory availability.

Affected Code:

```
function withdrawFees(address _from, uint8 _maxCount) external onlyRole(DEFAULT_A
DMIN_ROLE) {
require(_from == feeAddress || _from == treasury, "P-OWTF-01");
bytes32 symbol;
uint256 feeAccumulated = assets[_from][native].available;
if (feeAccumulated > 0) {
this.withdrawToken(
_from,
native,
feeAccumulated,
portfolioBridge.getDefaultBridgeProvider(),
portfolioBridge.getDefaultDestinationChain()
);
}
uint256 tokenCount = tokenList.length();
uint256 i;
while (_maxCount > 0) {
symbol = tokenList.at(i);
feeAccumulated = assets[_from][symbol].available;
if (feeAccumulated > 0) {
this.withdrawToken(
_from,
symbol,
feeAccumulated,
portfolioBridge.getDefaultBridgeProvider(),
portfolioBridge.getDefaultDestinationChain()
);
_maxCount--;
}
unchecked {
++i;
}
if (i == tokenCount) {
_maxCount = 0;
}
}
}
```

The limitation leads to:

- High Gas Costs: The iteration over a potentially large token list can lead to increased transaction costs, making the function expensive to execute.
- Limited Withdrawal Flexibility: By defaulting to the default destination chain for every token, the function does not account for tokens that might be more efficiently withdrawn to other chains, potentially leading to suboptimal financial management.

Assets:

- PortfolioSub.sol [https://github.com/Dexalot/contracts]

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 3
Impact [1-5]: 2
Exploitability [0-2]: 2
Complexity [0-2]: 0
Final Score: 1.8 (Low)

Hacken Calculator Version: 0.6

Recommendations

Recommendation:

- Function Modification or Removal: The issue can be addressed by either removing the `withdrawFees` function or modifying it to allow for the parameterization of the destination chain. This would enable the withdrawal of different portions of fees from appropriate chains, ensuring the successful collection of total fees across the multi-chain environment.
- Implement Parameterization:
  - Enhanced Multi-Chain Withdrawal Strategy: Develop a strategy to determine the most efficient chain for withdrawing fees for each token. This could involve analyzing inventory levels, transaction costs, and other factors to optimize fee withdrawals across the multi-chain environment.
  - Implement Range Parameters: Modify the `withdrawFees` function to include `fromIndex` and `toIndex` parameters, allowing administrators to specify a range of tokens to process for withdrawals. This approach provides more control over the withdrawal process, reducing gas costs by limiting the number of tokens processed in a single transaction.

Remediation (Revised commit: a85585e): The Dexalot team addressed this issue by removing the withdrawFees function from the PortfolioSub contract. This decision eliminates the inefficiencies and limitations previously present in the contract's multi-chain fee withdrawal process.

F-2024-1289 - Expanded Access Control Risks in depositToken Functions Allowing Unauthorized Token Transfers - Low

Description:

Trusted contracts in Dexalot's ecosystem are specific smart contracts that have been granted elevated privileges to perform certain actions which ordinary contracts or addresses cannot. These trusted contracts are typically used for coordinated activities related to auction tokens, especially in collaboration with partners like the Avalaunch Team. The primary purpose of these trusted contracts is to facilitate controlled token distribution, ensuring orderly price discovery and maintaining the integrity of auction processes within Dexalot's platform.

The `depositTokenFromContract` and `depositToken` functions in the contract extend significant privileges to Trusted Contracts, allowing them to deposit tokens on behalf of any user.

Affected Code:

```
function depositTokenFromContract(address _from, bytes32 _symbol, uint256 _quanti
ty) external override {
require(trustedContracts[msg.sender], "P-AOTC-01"); // keeping it for backward co
mpatibility
this.depositToken(_from, _symbol, _quantity, portfolioBridge.getDefaultBridgeProv
ider());
}


function depositToken(
address _from,
bytes32 _symbol,
uint256 _quantity,
IPortfolioBridge.BridgeProvider _bridge
) external whenNotPaused nonReentrant {
require(
_from == msg.sender ||
msg.sender == address(this) || // allow calls made by depositTokenFromContract
trustedContracts[msg.sender], // keeping it for backward compatibility
"P-OODT-01"
);
require(tokenList.contains(_symbol), "P-ETNS-01");
require(tokenDetailsMap[_symbol].isVirtual == false, "P-VTNS-01"); // Virtual tok
ens can't be deposited
require(_quantity <= tokenMap[_symbol].balanceOf(_from), "P-NETD-01");

tokenMap[_symbol].safeTransferFrom(_from, address(this), _quantity);
deposit(_from, _symbol, _quantity, _bridge);
}
```

While this feature serves the purpose of auction token coordination, it has a broader impact:

- Unexpected Withdrawals: Users might not anticipate that a Trusted Contract can initiate transfers on their behalf. This can lead to unexpected token withdrawals from their accounts.
- Fee Deduction Concerns: If the system introduces fee deductions for such transactions in the future, users could inadvertently incur costs due to transfers initiated by others. While the deposit is limited to the user's own address, and cannot be used to transfer funds from one user to another, the possibility of fee deductions adds an extra layer of complexity.
- Trust and Control Issues: The trust placed in these Trusted Contracts is significant. Although they are currently used for specific auction-related activities, the broad powers they hold could be a point of concern, especially if these contracts are compromised.

Assets:

- PortfolioMain.sol [https://github.com/Dexalot/contracts]

Status: Mitigated

Classification

Severity: Low

Impact:

Likelihood [1-5]: 2
Impact [1-5]: 5
Exploitability [0-2]: 2
Complexity [0-2]: 0
Final Score: 2.3 (Low)

Hacken Calculator Version: 0.6

Recommendations

Recommendation:

- Restrict Use to Auction-Related Scenarios: Modify `depositTokenFromContract` and `depositToken` functions to include checks or limitations that specifically limit their usage to auction-related activities. This approach ensures these functions cannot be misused for general ERC20 token transactions.
- Enhance User Awareness: Increase transparency by clearly communicating to users about the operations of Trusted Contracts. This includes information on how these contracts can initiate token withdrawals and the possibility of fee deductions.
- Implement Notifications and Controls: Introduce mechanisms for notifying users when Trusted Contracts perform transactions on their behalf. Additionally, consider implementing tighter controls for such transactions to prevent any unforeseen user impact.

Remediation (Revised commit: a85585e) (Mitigated): Given the design choices and existing control mechanisms, Dexalot does not plan to alter the current implementation of these functions. The protocol ensures that the broad powers of Trusted Contracts are specifically utilized for auction-related activities, mitigating the risk of misuse for general ERC20 token transactions.

Protocol statement :

- Only the Tokendetails in the subnet has the auction status, and it is not reflected in the mainnet to simplify the logic & synchronization between chains. (A single chain/contract in charge of auction status as a design choice)
- Avalaunch deploys a different contract every time there is a new auction for a specific token and we add this new contract address specific to the auction token is added to the trusted contracts. As a result assuming that Avalaunch contracts are error free, the scope is already limited to the auction token.
- Even though it is possible, we never charged deposit fees in the last 2 years and we are not planning to do so in the future, hence fee deductions are currently not relevant.
- Implement Notifications and Controls: Notification controls are implemented in the Avalaunch side where the user has to explicitly chose to deposit to Dexalot via the Avalaunch front-end.

Observation Details

F-2024-1146 - Missing checks for `address(0)` - Info

Description:

In Solidity, the Ethereum address `0x0000000000000000000000000000000000000000` is known as the "zero address". This address has significance because it's the default value for uninitialized address variables and is often used to represent an invalid or non-existent address. The " Missing zero address control" issue arises when a Solidity smart contract does not properly check or prevent interactions with the zero address, leading to unintended behavior. For instance, a contract might allow tokens to be sent to the zero address without any checks, which essentially burns those tokens as they become irretrievable. While sometimes this is intentional, without proper control or checks, accidental transfers could occur.

- Exchange: `addAdmin`, `addAuctionAdmin`, `setPortfolio`
- ExchangeMain: `setPriceFeed`
- ExchangeSub: `setOrderBooks`, `setTradePairs`
- PortfolioMain: `addTrustedContract`, `setBannedAccounts`

Assets:

- Exchange.sol [https://github.com/Dexalot/contracts]
- ExchangeMain.sol [https://github.com/Dexalot/contracts]
- ExchangeSub.sol [https://github.com/Dexalot/contracts]
- PortfolioMain.sol [https://github.com/Dexalot/contracts]

Status: Accepted

Recommendations

Recommendation:

Implement zero address validations for all address parameters in the affected functions. Ensure that addresses provided to these functions are always non-zero to maintain the integrity of contract operations.

Remediation (Revised commit: a85585e): The identified issue of "Missing Zero Address Control" across various functions in the Exchange, ExchangeMain, ExchangeSub, and PortfolioMain contracts was acknowledged and accepted.

F-2024-1148 - Lack of Events in Key Functions - Info

Description:

In the smart contract ecosystem, events are crucial for tracking changes and providing transparency. However, in several key functions across multiple contracts, events are notably absent.

The functions in question are:

ExchangeSub: `setOrderBooks`

InventoryManager: `updateA`, `setInventoryBySymbolId`, `convertSymbol`

PortfolioBridgeMain: `enableBridgeProvider`, `setDefaultBridgeProvider`

PortfolioBridgeSub: `addToken`, `removeToken`, `updateInventoryBySource`, `setDelayedTransfer`, `setInventoryManager`

PortfolioMain: `updateTokenDetailsAfterUpgrade`, `setBannedAccounts`

PortfolioSub: `updateTokenDetailsAfterUpgrade`

PortfolioSubHelper: `addConvertibleToken`, `removeConvertibleToken`

TradePairs: `pauseTradePair`, `pauseAddOrder`, `postOnly`, `setMaxNbrOfFills`

LzApp: `setLzEndPoint`

The absence of events in these key functions can lead to a lack of transparency and hinder effective monitoring. This can be particularly concerning in scenarios where contract administrators or users need to verify that certain actions have been performed correctly.

Assets:

- ExchangeSub.sol [https://github.com/Dexalot/contracts]
- bidgeApps/LzApp.sol [https://github.com/Dexalot/contracts]
- PortfolioBridgeMain.sol [https://github.com/Dexalot/contracts]
- PortfolioBridgeSub.sol [https://github.com/Dexalot/contracts]
- PortfolioMain.sol [https://github.com/Dexalot/contracts]
- PortfolioSub.sol [https://github.com/Dexalot/contracts]
- PortfolioSubHelper.sol [https://github.com/Dexalot/contracts]
- TradePairs.sol [https://github.com/Dexalot/contracts]
- InventoryManager.sol [https://github.com/Dexalot/contracts]

Status: Accepted

Recommendations

Recommendation:

To address this issue, it is recommended to add appropriate event emissions in each of these key functions. The events should be designed to log relevant information that reflects the changes made by the function calls. The events should include parameters that provide a clear and comprehensive log of the actions taken within the function. This may include identifiers, new settings, timestamps, and any other relevant data.

Remediation (Revised commit: a85585e): The recommendation to include event emissions in several key functions across multiple contracts was partially implemented. Selective modifications were made to enhance transparency and monitoring capabilities in some areas, though not all proposed events were added.

F-2024-1151 - `public` functions not called by the contract should be declared `external` instead - Info

Description:

In Solidity, function visibility is an important aspect that determines how and where a function can be called from. Two commonly used visibilities are `public` and `external`. A `public` function can be called both from other functions inside the same contract and from outside transactions, while an `external` function can only be called from outside the contract. A potential pitfall in smart contract development is the misuse of the `public` keyword for functions that are only meant to be accessed externally. When a function is not used internally within a contract and is only intended for external calls, it should be labeled as `external` rather than `public`. Using `public` unnecessarily can introduce potential vulnerabilities and also make the contract consume more gas than required. This is because `public` functions have to add additional code to handle both internal and external calls, while `external` functions can be more optimized since they only handle external calls.

The functions in question are:

- Exchange: `initialize`, `addAdmin`, `removeAdmin`, `isAdmin`
- PortfolioBridgeMain: `initialize`
- PortfolioSubHelper: `initialize`
- TradePairs: `initialize`

Assets:

- Exchange.sol [https://github.com/Dexalot/contracts]
- PortfolioBridgeMain.sol [https://github.com/Dexalot/contracts]
- PortfolioSubHelper.sol [https://github.com/Dexalot/contracts]
- TradePairs.sol [https://github.com/Dexalot/contracts]

Status: Fixed

Recommendations

Recommendation:

To optimize gas usage and improve code clarity, declare functions that are not called internally within the contract and are intended for external access as `external` rather than `public`. This ensures that these functions are only callable externally, reducing unnecessary gas consumption and potential security risks.

Remediation (Revised commit: a85585e): The issue of inappropriate use of public function visibility in various contracts was effectively addressed.

F-2024-1153 - Function Parameter Shadowing in Portfolio and TradePairs Contracts - Info

Description:

In the Portfolio.sol and TradePairs.sol contracts, which inherit PausableUpgradeable from OpenZeppelin, there are instances of function parameter shadowing. Specifically, the parameter `_pause` is used in functions that shadow the `_pause` function from the inherited PausableUpgradeable contract.

- In Portfolio.sol, the `pauseDeposit` function uses `_pause` as a parameter, which shadows the `_pause` function.

```
function pauseDeposit(bool _pause) external override onlyRole(DEFAULT_ADMIN_ROLE)
{
allowDeposit = !_pause;
}
```

- In TradePairs.sol, the `pauseTradePair` and `pauseAddOrder` functions also use `_pause` as a parameter, creating a similar shadowing issue.

```
function pauseTradePair(bytes32 _tradePairId, bool _pause) public override onlyRo
le(DEFAULT_ADMIN_ROLE) {
tradePairMap[_tradePairId].pairPaused = _pause;
}

function pauseAddOrder(bytes32 _tradePairId, bool _pause) external override onlyR
ole(DEFAULT_ADMIN_ROLE) {
tradePairMap[_tradePairId].addOrderPaused = _pause;
}
```

- The inherited `_pause` function from PausableUpgradeable:

```
function _pause() internal virtual whenNotPaused {
_paused = true;
emit Paused(_msgSender());
}
```

While this issue does not pose a direct security risk, it can lead to confusion, readability, and maintainability issues in the code. Using the same name for a parameter and an inherited function can be misleading and may cause errors, especially during code updates or maintenance.

Assets:

- Portfolio.sol [https://github.com/Dexalot/contracts]
- TradePairs.sol [https://github.com/Dexalot/contracts]

Status: Fixed

Recommendations

Recommendation:

To resolve this shadowing issue, it's advisable to rename the `_pause` parameters in the `pauseDeposit`, `pauseTradePair`, and `pauseAddOrder` functions to distinct names that do not conflict with the inherited `_pause` function.

Remediation (Revised commit: a85585e): The issue of function parameter shadowing in the Portfolio.sol and TradePairs.sol contracts was effectively resolved, with parameters previously named `_pause` in several functions being renamed to avoid inadvertently shadowing the inherited `_pause` function from PausableUpgradeable.

F-2024-1186 - Potential Reentrancy Vulnerabilities in MainnetRFQ.sol - Info

Description:

A reentrancy attack is a common vulnerability in smart contracts, particularly when a contract makes an external call to another untrusted contract and then continues to execute code afterwards. If the called contract is malicious, it can call back into the original contract in a way that causes the original function to run again, potentially leading to effects like multiple withdrawals and other unintended actions.

The MainnetRFQ.sol contract in the Dexalot protocol contains several functions that potentially expose the system to reentrancy vulnerabilities:

- `simpleSwap`, `partialSwap`, `xChainSwap`: These functions act as external entry points for swaps and lack the `nonReentrant` modifier.
- `_releaseFunds` and `_refundNative`: These functions execute external transfers (ERC20 or ETH) without reentrancy checks.

These functions interact with external addresses, either through ERC20 transfers or direct ETH transfers:

```
function _releaseFunds(SwapData memory _swapData) private {
if (_swapData.destAsset == address(0)) {
(bool success, ) = payable(_swapData.destTrader).call{value: _swapData.destAmount
}("");
require(success, "RF-TF-01");
} else {
IERC20Upgradeable(_swapData.destAsset).safeTransfer(_swapData.destTrader, _swapDa
ta.destAmount);
}
}

function _refundNative(SwapData memory _swapData) private {
if (_swapData.srcAsset == address(0) && msg.value > _swapData.srcAmount) {
(bool success, ) = payable(msg.sender).call{value: msg.value - _swapData.srcAmoun
t}("");
require(success, "RF-TF-02");
}
}
```

While currently mitigating direct reentrancy risks through the early update of `completedSwaps[bucket]` in the `_verifySwapInternal` function, still presents potential vulnerabilities due to its upgradeable nature. This contract, responsible for handling swaps and cross-chain swaps, could become susceptible to reentrancy attacks in future upgrades if modifications are not carefully implemented and reviewed.

Status: Mitigated

Recommendations

Recommendation:

Apply OpenZeppelin's `nonReentrant` modifier to `simpleSwap`, `partialSwap`, and `xChainSwap` to prevent re-entry of these functions during execution.

Remediation (Revised commit: a85585e): The MainnetRFQ.sol contract in the Dexalot protocol was updated to include OpenZeppelin's `nonReentrant` modifier for the `simpleSwap` and `partialSwap` functions, enhancing their security against reentrancy attacks. However, the `xChainSwap` function remains without this added protection.

F-2024-1197 - Absence of Storage Gap (__gap) in Parent Upgradable Contracts - Info

Description:

In upgradeable smart contracts, particularly those using a proxy pattern, the storage layout is crucial. While PortfolioBridgeMain.sol includes a `__gap`, making it adaptable for future upgrades, LzApp.sol lacks this important feature. This omission restricts the ability to introduce new state variables or make storage adjustments in LzApp.sol in future versions. In a proxy contract system, storage variables must follow a strict order to avoid collisions and maintain state integrity. Therefore, not having a `__gap` in a parent contract like LzApp.sol could complicate or even prevent essential upgrades or optimizations in the future.

For contracts within the system that will undergo upgrades rather than fresh deployments, it's crucial to note that no modifications should alter the storage in parent contracts if they do not contain `__gaps`. The only permissible modifications are in child contracts to ensure storage integrity.

The primary risk is the potential limitation on future upgrades. If new state variables need to be added or storage layout changes are required in LzApp.sol, the absence of `__gap` would necessitate more complex and riskier strategies for upgrading. While there's no immediate impact on contract functionality or security, this limitation can become significant in the context of long-term contract maintainability and scalability.

Assets:

- bidgeApps/LzApp.sol [https://github.com/Dexalot/contracts]

Status: Fixed

Recommendations

Recommendation:

Add a storage gap, such as `uint256[50] private __gap;`, to the end of the LzApp.sol contract. This will provide necessary flexibility for future upgrades, allowing for storage layout changes without affecting existing functionality.

Remediation (Revised commit: a85585e): The Dexalot team successfully implemented a storage gap in the LzApp.sol contract.

F-2024-1288 - Improvement in State Management and External Interactions in _processXFerPayloadInternal Function - Info

Description:

The `_processXFerPayloadInternal` function in the smart contract serves a critical role in managing swaps by releasing swap funds to traders or adding swaps to a queue. However, it exhibits a potential lapse in the implementation of the Check-Effects-Interactions pattern, specifically in the ordering of state updates and external calls.

Affected Code:

```
function _processXFerPayloadInternal(
address _trader,
bytes32 _symbol,
uint256 _quantity,
uint256 _nonceAndMeta
) private returns (bool) {
uint256 bucket = _nonceAndMeta >> 8;
uint256 mask = 1 << (_nonceAndMeta & 0xff);
require(completedSwaps[bucket] & mask == 0, "RF-IN-02");

bool success = false;
// External calls for fund transfer
// ...

if (!success) {
_addToSwapQueue(_trader, _symbol, _quantity, _nonceAndMeta);
return false;
}
// State update after external interactions
completedSwaps[bucket] |= mask;
return true;
}
```

In this implementation, the state update (`completedSwaps[bucket] |= mask;`) occurs after external calls for fund transfer. This approach does not fully adhere to the Check-Effects-Interactions pattern, which suggests that state changes should precede external interactions.

Although protected by `nonReentrant`, future modifications or overlooked scenarios could lead to reentrancy issues.

Assets:

- MainnetRFQ.sol [https://github.com/Dexalot/contracts]

Status: Accepted

Recommendations

Recommendation:

In the `_processXFerPayloadInternal` function, the state (`completedSwaps[bucket]`) should be updated before any external calls to mark the swap as completed. If a transfer fails, this state update can be reverted using the `^=` bitwise XOR operation with the same mask, ensuring the contract's state accurately reflects the transaction outcome.

Remediation (Revised commit: a85585e): The function continues to perform state updates after external fund transfer calls, diverging from the recommended practice of updating the state before any external interactions.

F-2024-1313 - Inadequate Parameter Validation and Loop Limitation in TradePairs Contract - Info

Description:

TradePairs Contract functions `cancelReplaceList` , `addLimitOrderList` and `cancelOrderList` are crucial for market makers, enabling efficient order management. However, two potential issues have been identified in its current implementation.

- Inconsistent Parameter Sides:
  - The lack of validation for the consistency of order sides in input parameters can lead to an 'Index Out of Bounds' error. In such cases, the user attempting the transaction would incur gas costs, only to have the operation fail due to improper parameter alignment. This not only results in wasted gas but also disrupts the intended trading activities of the user.
- Unlimited Loop Iterations:
  - Without a limit on the number of iterations in the loop, there is a significant risk of the transaction running out of gas, especially when processing a large number of orders. Similar to the first issue, this would result in the transaction being reverted, leading to a loss of gas without successful execution. This can be particularly problematic for market makers who rely on the efficiency and reliability of such functions for their trading strategies.

Affected Code:

```
// In cancelReplaceList
for (uint256 i = 0; i < _orderIds.length; ++i) {
// ...
addOrderPrivate(...);
// ...
}

// In addLimitOrderList
for (uint256 i = 0; i < _clientOrderIds.length; ++i) {
// ...
doOrderCancel(...);
addOrderPrivate(...);
// ...
}

// In cancelOrderList
for (uint256 i = 0; i < _orderIds.length; ++i) {
// ...
doOrderCancel(...);
// ...
}
```

Users attempting these transactions are likely to incur unnecessary gas costs without successful execution.

Assets:

- TradePairs.sol [https://github.com/Dexalot/contracts]

Status: Mitigated

Recommendations

Recommendation:

- Parameter Consistency Checks:
  - Implement checks in `cancelReplaceList` and `addLimitOrderList` to validate the consistency of input parameters.
  - Ensure each array correlates properly with others to prevent 'Index Out of Bounds' errors and ensure correct order alignment.
- Loop Iteration Limit:
  - Introduce a limit on the number of iterations within the loops of `cancelReplaceList`, `addLimitOrderList` and `cancelOrderList`.
  - Determine an optimal limit considering gas constraints of a single block to ensure efficient processing of batch orders without risking gas exhaustion.

Remediation (Revised commit: a85585e) (Mitigated): Dexalot opts not to modify the current implementation of these functions. The decision is aligned with the objective of maintaining efficient order management for market makers while balancing gas optimization and blockchain capabilities.

Protocol statement :

- Natspecs for these functions state that they may run out of gas and give suggested number of orders to pass.
  We refrained from adding additional checks
- To save gas because market makers are more sophisticated than the general public and they should be capable enough to deal with wrong parameters and or out of bound iteration limits
- We can increase our blockchain (our subnet) block gas limit if needed and we would need a contract upgrade to reflect the new iteration limits

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/Dexalot/contracts
Commit	2492aa22451ac6b8260d8864b68dd82b1cbdb1f5
Whitepaper	https://docs.dexalot.com/articles/litepaper/
Requirements	https://docs.dexalot.com/contracts
Technical Requirements	https://docs.dexalot.com/contracts; NatSpec

Contracts in Scope

./contracts/DelayedTransfers.sol

./contracts/Exchange.sol

./contracts/ExchangeMain.sol

./contracts/ExchangeSub.sol

./contracts/MainnetRFQ.sol

./contracts/Portfolio.sol

./contracts/PortfolioMain.sol

./contracts/PortfolioSub.sol

./contracts/PortfolioSubHelper.sol

./contracts/PortfolioBridgeMain.sol

./contracts/PortfolioBridgeSub.sol

./contracts/TradePairs.sol

./contracts/InventoryManager.sol

./contracts/bidgeApps/LzApp.sol

./contracts/library/UtilsLibrary.sol

./contracts/library/InvariantMathLibrary.sol

./contracts/interfaces/IInventoryManager.sol

./contracts/interfaces/IDelayedTransfers.sol

./contracts/interfaces/IERC1271.sol

./contracts/interfaces/layerZero/ILayerZeroUserApplicationConfig.sol

./contracts/interfaces/layerZero/ILayerZeroEndpoint.sol

./contracts/interfaces/layerZero/ILayerZeroReceiver.sol

./contracts/interfaces/IMainnetRFQ.sol

./contracts/interfaces/IPortfolio.sol

Contracts in Scope

./contracts/interfaces/IPortfolioBridgeSub.sol

./contracts/interfaces/IPortfolioMain.sol

./contracts/interfaces/IPortfolioSub.sol

./contracts/interfaces/IBannedAccounts.sol

./contracts/interfaces/IPortfolioBridge.sol

./contracts/interfaces/IPortfolioSubHelper.sol

./contracts/interfaces/IPortfolioMinter.sol

./contracts/interfaces/IGasStation.sol

interfaces/ITradePairs.sol