Bento.Fun Protocol

Smart Contract Security Audit Report

Audited by: TenArmor
Audit dates: Dec 2, 2025 - Dec 10, 2025

Content

1. Introduction

1.1 About TenArmor

TenArmor is a leading Web3 security firm dedicated to safeguarding blockchain ecosystems. We offer cutting-edge security solutions tailored to the complexities of decentralized technologies. With our flagship products, TenMonitor for real-time threat detection and response and TenTrace for a one-stop AML solution, we provide comprehensive protection against evolving threats. Our expertise spans smart contract auditing, cryptocurrency tracing, and beyond, making TenArmor a trusted partner for organizations aiming to secure their digital assets in the decentralized world.

Learn more about us:
Official Website: TenArmor.com
X: @TenArmor, @TenArmorAlert
E-mail: team@tenarmor.com
TenMonitor: TenMonitor.com
TenTrace: TenTrace.com

1.2 Disclaimer

This report represents an analysis performed within a defined scope and time frame, based on the materials and documentation provided. It is not exhaustive and does not encompass all potential vulnerabilities.

A smart contract security review aims to identify as many vulnerabilities as possible within the given constraints of time, resources, and expertise. However, it cannot guarantee the complete absence of vulnerabilities or the complete security of the project. This report does not serve as an endorsement of any specific project or team.

To enhance security, we strongly recommend conducting subsequent security reviews, implementing bug bounty programs, and maintaining ongoing on-chain monitoring.

1.3 Risk Classification

Severity Level	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2. Executive Summary

2.1 About Bento.Fun Protocol

Bento.Fun is an upgradeable on-chain binary options and prediction market protocol implemented with the Diamond (EIP-2535) pattern, where users create “duels” (YES/NO style markets) under public or private leagues, place tokenized wagers (minted as ERC20 option tokens), and later settle outcomes with a contest/dispute period and chunked payout/refund distribution. The system supports both USDC and configurable “conviction” tokens, enforces league- and duel-level privacy and curator/creator permissions, and exposes a secondary marketplace facet for peer-to-peer trading of option tokens before expiry. Core execution, admin configuration, settlement, and marketplace logic are split across dedicated facets sharing a single AppStorage, while auxiliary libraries (e.g. LibBento, LibSwap) handle ID generation, validation, fee accounting, and Uniswap v4–based token swaps via Permit2 and a Universal Router.

2.2 Audit Scope

Repository:
https://github.com/FilamentFinance/bento.fun/tree/testnet/packages/contracts

Audit Commit Hash: c2ebfa74bb74306207b8773729932ed52c29c51d

2.3 Findings Count

Severity	Count
Critical	1
High	2
Medium	4
Low	7
Informational	5
Total Findings	**19**

3. Findings Summary

ID	Description	Status
[C-1]	Double-counting losers' pot in chunked distribution with transferable option tokens.	Fixed
[H-1]	Incorrect swap output computation in LibSwap._executeSwap.	Fixed
[H-2]	Resolving period checks can lock duels in an unrecoverable Live state.	Fixed
[M-1]	Pending duel index mismatch between per-user and global arrays.	Fixed
[M-2]	Non-intuitive and dimensionally brittle minimum wager check in joinDuel.	Fixed
[M-3]	Missing slippage protection in swapLosingToWinningTokens.	Acknowledged
[M-4]	Hard deletion of leagues breaks privacy and fee semantics.	Fixed
[L-1]	Missing basic validation on duel options and option prices.	Fixed
[L-2]	Ambiguous use of zero as a sentinel for per-option minimum wager thresholds.	Fixed
[L-3]	Inconsistent and unsafe use of raw ERC20 transfers instead of SafeERC20.	Fixed
[L-4]	settleDuel does not validate winning option index range.	Fixed
[L-5]	Settlement functions lack duel ID validity checks and uniform pause protection.	Fixed
[L-6]	Missing zero-address validation in DiamondInit.init.	Fixed
[L-7]	Insufficient validation of admin-set configuration parameters.	Fixed
[I-1]	Inconsistent comments across multiple functions.	Fixed
[I-2]	Comment mismatch for _totalWinnings in calculateFeesAndPayoutForSwappedTokens.	Fixed
[I-3]	Unused local variable in finalizeAndDistribute.	Fixed
[I-4]	Leftover Hardhat console logging.	Fixed
[I-5]	Duplicate duel getter in BentoViewFacet.	Fixed

Feedback Details: https://docs.google.com/spreadsheets/d/1DbckJhACi1n6tuR1ZsddR51O1hJoSkYSxM49HnUi6K0

4. Findings

4.1 Critical Risk

[C-1] Double-counting losers' pot in chunked distribution with transferable option tokens

Description:
LibBento.distributeWinningsInChunks computes each winner's shareOfLosersPot from the current OptionToken balance and the current totalSupply, while _losersPot remains a fixed total for the whole distribution process. Because option tokens are freely transferable and distribution is processed in chunks, the same underlying tokens can be moved between multiple winner addresses across batches and repeatedly counted for a share of the same _losersPot.

libraries/LibBento.sol Line#363

```
359     // Winner's net bet after ALL fees (protocol already deducted, now deduct additional)
360     uint256 netBetReturn = netBetAfterProtocolFee - additionalFeesOnBet;
361
362     // Get winner's proportional share of the loser's pot based on option tokens
363     uint256 winnerShare = IBentoView(_bentoViewAddress).getUserDuelOptionShare(_duelId, _optionIndex, winner);
364     uint256 shareOfLosersPot = (winnerShare * _losersPot) / 1e18;
365
366     // Total payout = net bet (after all fees) + share of loser's pot
367     uint256 totalPayout = netBetReturn + shareOfLosersPot;
368
369     if (totalPayout > 0) {
370         s.allTimeEarnings[winner][_associatedToken] += totalPayout;
371         emit WinningsDistributed(_duelId, option, _optionIndex, winner, totalPayout, block.timestamp);
372     }
373
374     // Clear the user's wager after distribution
375     s.userWager[winner][_duelId][_optionIndex] = 0;
```

Impact:
A coordinated attacker controlling multiple winner addresses can transfer option tokens between them between distribution batches so that the same tokens are used to claim a share of the losers' pot multiple times. This leads to over-distribution of _losersPot (more than 100% paid out), effectively draining funds from the protocol or other users and breaking economic correctness.

Recommendation:
Snapshot each winner's share at the time distribution starts (e.g., via userWager/totalWagerForOption or a one-time snapshot of balanceOf/totalSupply), and base all chunked payouts on this immutable snapshot, not on mutable token balances. Alternatively, decrement a "remaining pot" as you pay each winner (so sum of all shareOfLosersPot cannot exceed the original pot), and disallow transfers of OptionTokens between start of distribution and completion.

4.2 High Risk

[H-1] Incorrect swap output computation in LibSwap._executeSwap

Description:
In LibSwap._executeSwap, amountOut is computed as the entire IERC20(outputToken).balanceOf(address(this)) after the Uniswap V4/Universal Router call, without subtracting the pre-swap balance. If the contract already holds some outputToken before the swap, that existing balance is incorrectly counted as part of the swap output.

libraries/LibSwap.sol Line#232

```
209         // Prepare parameters for each action
210         bytes[] memory params = new bytes[](3);
211         params[0] = abi.encode(
212             IV4Router.ExactInputSingleParams({
213                 poolKey: poolKey,
214                 zeroForOne: zeroForOne,
215                 amountIn: uint128(amountIn),
216                 amountOutMinimum: uint128(minAmountOut),
217                 hookData: bytes("")
218             })
219         );
220         params[1] = abi.encode(zeroForOne ? poolKey.currency0 : poolKey.currency1, amountIn);
221         params[2] = abi.encode(zeroForOne ? poolKey.currency1 : poolKey.currency0, minAmountOut);
222 
223         // Combine actions and params into inputs
224         inputs[0] = abi.encode(actions, params);
225 
226         // Execute the swap
227         uint256 deadline = block.timestamp + 300; // 5 minutes deadline
228         router.execute(commands, inputs, deadline);
229 
230         // Get the output amount - tokens are received by this contract
231         Currency outputCurrency = zeroForOne ? poolKey.currency1 : poolKey.currency0;
232         amountOut = IERC20(Currency.unwrap(outputCurrency)).balanceOf(address(this));
233     }
```

Impact:
This can cause amountOut to be significantly overstated, allowing swaps to pass the require(amountOut >= minAmountOut) check even when the actual tokens received from the swap are below the user's minimum, effectively disabling slippage protection. Downstream logic that assumes amountOut is the true delta can misprice swaps, mis-account funds, or over-credit users relative to actual DEX execution.

Recommendation:
Track the before/after balances of the output token and compute amountOut as their difference, e.g. amountOut = balanceAfter - balanceBefore. Use this delta consistently for both the minAmountOut check and any subsequent accounting.

[H-2] Resolving period checks can lock duels in an unrecoverable Live state

Description:
BentoSettlementFacet.settleDuel requires duel.duelStatus == Live and block.timestamp <= expiryTime + s.resolvingPeriod, but BentoCoreFacet.startDuel only checks block.timestamp >= duel.startTime and has no upper bound relative to expiryTime + resolvingPeriod. If the bot is delayed (or fails) and calls startDuel or settleDuel after expiryTime + resolvingPeriod, the duel can be left in Live status with no valid path to settlement or refund.

facets/BentoSettlementFacet.sol Line#42

```
27    /// @notice Settles the duel after it has expired, but defers distribution until contest period ends.
28    /// @param _duelId The ID of the duel to settle.
29    /// @param _winningOptionIndex The winning option index of the duel.
30    /// @dev This function records settlement and opens a contest period. No swaps or distributions happen here.
31    function settleDuel(string memory _duelId, uint256 _winningOptionIndex) external nonReentrant onlyBot {
32        Duel storage duel = s.duels[_duelId];
33        LibBento.LibBentoAppStorage storage libBentoStorage = LibBento.appStorage();
34        require(
35            libBentoStorage.isValidDuelId[_duelId] && duel.createTime != 0,
36            BentoCoreFacet__DuelDoesNotExist()
37        );
38        // Ensure the duel is live and has expired, but within resolving time
39        require(duel.duelStatus == DuelStatus.Live, BentoCoreFacet__DuelIsNotLiveOrSettled());
40        uint256 expiryTime = duel.expiryTime;
41        // Ensure the duel is within the resolving period
42        require(block.timestamp <= expiryTime + s.resolvingPeriod, BentoCoreFacet__ResolvingTimeExpired());
43
44        // Remove from live duels tracking
45        s.isLiveDuel[_duelId] = false;
```

Impact:
In such delayed scenarios, settleDuel will always revert due to the resolving period check, and cancelDuelIfThresholdNotMet can no longer be used (requires BootStrapped). This leaves a Live duel whose funds remain locked in the contract indefinitely, with users unable to recover their stakes or receive payouts, creating a hard funds-lock risk tied to off-chain bot liveness.

Recommendation:
Introduce robust time-window handling:

- Provide an explicit fallback path for duels that have exceeded expiryTime + resolvingPeriod without being settled (e.g. an owner/bot-only cancelExpiredDuelAndRefund that transitions the duel to a cancelled/refund state).
- require block.timestamp >= expiryTime in settleDuel to prevent premature settlement and align with joinDuel's expiry semantics.

4.3 Medium Risk

[M-1] Pending duel index mismatch between per-user and global arrays

Description:
In BentoCoreFacet._processPendingDuel and _revokePendingDuel, the same _index argument is used to index both s.pendingDuels[_user][_leagueId] (the per-creator array) and s.allPendingDuels[_leagueId] (the league-wide array). There is no guarantee that the same _index refers to the same PendingDuel in both arrays once multiple users' duels are appended in different orders.

facets/BentoCoreFacet.sol Line#468

```
438     function _processPendingDuel(
439         address _user,
440         string memory _leagueId,
441         uint256 _index
442     ) internal returns (string memory) {
443         PendingDuel[] storage userPendingDuels = s.pendingDuels[_user][_leagueId];
444         require(_index < userPendingDuels.length, BentoCoreFacet__InvalidPendingDuelsIndex());
445         PendingDuel memory pendingDuel = userPendingDuels[_index];
446         require(!pendingDuel.isApproved, BentoCoreFacet__DuelAlreadyApproved());
447         string memory duelId = _createDuel(
448             _leagueId,
449             pendingDuel.category,
450             pendingDuel.description,
451             pendingDuel.options,
452             pendingDuel.associatedTokens,
453             _user,
454             pendingDuel.minBets,
455             pendingDuel.maxBets,
456             // pendingDuel.duration,
457             pendingDuel.startTime,
458             pendingDuel.endTime,
459             pendingDuel.privacyAccess
460         );
461         uint256 lastIndex = userPendingDuels.length - 1;
462         if (_index != lastIndex) {
463             userPendingDuels[_index] = userPendingDuels[lastIndex];
464         }
465         userPendingDuels.pop();
466         lastIndex = s.allPendingDuels[_leagueId].length - 1;
467         if (_index != lastIndex) {
468             s.allPendingDuels[_leagueId][_index] = s.allPendingDuels[_leagueId][lastIndex];
469         }
```

Impact:
Using the user-specific index to delete from the global allPendingDuels array can remove or shuffle the wrong PendingDuel entry at the league level. This can corrupt the pending-duel view for other creators, break external tooling relying on getAllPendingDuelsAndCount, and generally desynchronize per-user vs global pending duel state.

Recommendation:
Decouple indices between the per-user and global arrays. For example, store an explicit

globalIndex inside each PendingDuel, or maintain a mapping from a stable duel request ID -> its index in allPendingDuels. When processing/revoking a pending duel, use the correct index for each array independently rather than reusing a single _index across both.

[M-2] Non-intuitive and dimensionally brittle minimum wager check in joinDuel

Description:

BentoCoreFacet.joinDuel enforces a minimum wager with require((_amount * 1e6) / (10 ** decimals) >= _optionPrice, ...), where _amount is in _associatedToken units and _optionPrice is a price in micro-USDC. This effectively assumes a 1:1 relation between one unit of _associatedToken and one USDC and mixes token quantity (scaled to 1e6) with a USDC price per option, which only makes sense if _associatedToken is exactly USDC with 6 decimals. The code, however, allows arbitrary _associatedTokens.

facets/BentoCoreFacet.sol Line#181

```
149     function joinDuel(
150         string memory _duelId,
151         string memory _option,
152         uint256 _optionsIndex,
153         uint256 _optionPrice,
154         uint256 _amount,
155         uint256 _amountInUsdc,
156         address _associatedToken,
157         address _user
158     ) external nonReentrant whenNotPaused onlyBot {
159         require(_optionsIndex < 2 && _user != address(0) && _associatedToken != address(0), BentoCoreFacet__Invali
160         // Check if the option is valid
161         string memory option = s.duelIdToOptions[_duelId][_optionsIndex];
162         require(bytes(option).length > 0 && keccak256(bytes(option)) == keccak256(bytes(_option)), BentoCoreFacet_
163         require(s.optionIndexToAssociatedToken[_duelId][_optionsIndex] == _associatedToken, BentoCoreFacet__Invali
164
165         Duel memory duel = s.duels[_duelId];
166         LibBento.LibBentoAppStorage storage libBentoStorage = LibBento.appStorage();
167         require(
168             libBentoStorage.isValidDuelId[_duelId] && duel.createTime != 0,
169             BentoCoreFacet__DuelDoesNotExist()
170         );
171         require(
172             duel.duelStatus == DuelStatus.BootStrapped || duel.duelStatus == DuelStatus.Live,
173             BentoCoreFacet__DuelIsNotLive()
174         );
175         require(block.timestamp < duel.expiryTime, BentoCoreFacet__DuelHasExpired());
176
177         // Check privacy access for joining duels
178         LibBento.validateDuelAccess(s, _duelId, _user, duel.privacyAccess);
179
180         uint256 decimals = IERC20Custom(_associatedToken).decimals();
181         require((_amount * 1e6) / (10 ** decimals) >= _optionPrice, BentoCoreFacet__LessThanMinimumWager());
182         uint256 minForOption = s.duelMinBetPerOption[_duelId][_optionsIndex];
```

Impact:

For non-USDC associated tokens (or USDC variants with different decimals), this check becomes dimensionally incorrect and can either allow wagers below the intended economic minimum or erroneously reject valid wagers. The semantics of “minimum one option’s worth of bet” become unclear and misaligned with the documented meaning of _optionPrice.

Recommendation:
Base the minimum check directly on the USDC value: require _amountInUsdc >= _optionPrice (or a clearly documented multiple thereof), where _amountInUsdc is already provided as the USDC equivalent of the wager. This keeps both sides in the same unit (micro-USDC) and avoids implicit assumptions about token decimals or 1:1 pricing.

[M-3] Missing slippage protection in swapLosingToWinningTokens

Description:
LibBento.swapLosingToWinningTokens constructs LibSwap.SwapParams with minAmountOut hardcoded to 0, then delegates to LibSwap.executeSwap. This means the swap of the loser's pot into the winning token is executed without any minimum output constraint or price tolerance.

libraries/LibBento.sol Line#193

```
176     /// @notice Swaps losing tokens to winning tokens
177     /// @param s The AppStorage struct
178     /// @param _losingToken The losing token address
179     /// @param _winningToken The winning token address
180     /// @param _amount The amount to swap
181     /// @return swappedAmount The amount received from the swap
182     function swapLosingToWinningTokens(
183         AppStorage storage s,
184         address _losingToken,
185         address _winningToken,
186         uint256 _amount
187     ) internal returns (uint256 swappedAmount) {
188         // Prepare swap parameters
189         LibSwap.SwapParams memory swapParams = LibSwap.SwapParams({
190             tokenIn: _losingToken,
191             tokenOut: _winningToken,
192             amountIn: _amount,
193             minAmountOut: 0 // Set to 0 for now. In production, calculate based on price impact tolerance
194         });
195
196         // Execute the swap
197         swappedAmount = LibSwap.executeSwap(s, swapParams);
198
199         require(swappedAmount > 0, BentoCoreFacet__NoTokensReceivedFromSwap());
200     }
```

Impact:
If the Uniswap V4 pool is illiquid, manipulated, or front-run, the loser's pot can be swapped at an extremely unfavorable rate, resulting in drastically reduced swappedAmount and thus a much smaller losersPotAfterFees for winners. In the worst case, nearly the entire losers' pot could be lost to slippage/manipulation before distribution, even though the function superficially succeeds.

Recommendation:
Introduce configurable or computed slippage protection for this settlement swap. For example, derive a conservative minAmountOut from an oracle price or recent TWAP, or from on-chain observables with a maximum tolerated slippage percentage (e.g. 1 – 3%). If no robust price source is available, consider avoiding automatic swaps and keeping

the pot in the original token, or exposing a separate governance/bot-driven path where a safe minAmountOut parameter is provided.

[M-4] Hard deletion of leagues breaks privacy and fee semantics

Description:
BentoAdminFacet.removeLeague deletes s.leagueIdToLeague[_leagueId] and removes the ID from s.curatorLeagues, but does not check whether the league has active duels, members, or outstanding economic relationships. Because delete resets the struct to all zero values, subsequent reads treat the league as if it were a default "public, no curator" league.

facets/BentoAdminFacet.sol Line#307

```
304     /// @notice Removes a league from the contract.
305     /// @dev This function can only be called by the contract owner.
306     /// @param _leagueId The ID of the league to remove.
307     function removeLeague(string memory _leagueId) external onlyOwner {
308         League memory league = s.leagueIdToLeague[_leagueId];
309         string[] storage leagues = s.curatorLeagues[league.leagueCurator];
310         for (uint256 i = 0; i < leagues.length; i++) {
311             if (keccak256(bytes(leagues[i])) == keccak256(bytes(_leagueId))) {
312                 // Move the last element to the position being deleted
313                 leagues[i] = leagues[leagues.length - 1];
314                 leagues.pop();
315                 break;
316             }
317         }
318         delete s.leagueIdToLeague[_leagueId];
319         emit LeagueRemoved(_leagueId);
320     }
```

e.g., finalizeAndDistribute using curator in facets/BentoSettlementFacet.sol Line#110

```
108         // Calculate additional fees (creator, curator, tokenAdmin) on the loser's pot only
109         // Winner's fees are calculated individually per winner during distribution
110         address curator = s.leagueIdToLeague[duel.leagueId].leagueCurator;
111         address tokenAdminWinner = s.convictionTokenAdmin[winningAssociatedToken];
112         address tokenAdminLooser = s.convictionTokenAdmin[losingAssociatedToken];
113
114         // Calculate fees on loser's pot (to be distributed to winners)
115         (uint256 losersPotAfterFees) = LibBento.calculateFeesAndPayoutForSwappedTokens(
116             s,
117             swappedAmount,
118             winningAssociatedToken,
119             losingAssociatedToken,
120             curator,
121             duel.creator,
122             tokenAdminWinner,
123             tokenAdminLooser
124         );
```

Impact:
Deleting a league mid-lifecycle can silently:

- Turn previously private leagues into effectively public ones (since privacyAccess defaults to Public), bypassing league membership checks;
- Remove the curator address, so future curator fees for that league accrue to no one;
- Confuse any logic or UI that assumes a nonzero League struct implies a valid, configured league.

All of this occurs without explicit indication that the league was “disabled” and without guaranteeing that there are no live duels or members, leading to broken access control and fee semantics.

Recommendation:

Replace hard deletion with a soft-delete/disabled flag (e.g. bool active inside League or a separate leagueDisabled[_leagueId] mapping), and require active == true wherever league semantics matter (duel creation, access checks, fee routing). When removing a league, enforce preconditions (no active or pending duels, optional no members) and set active = false instead of deleting the struct, so that other code paths can reliably detect “inactive league” rather than misinterpreting default zeroed fields.

4.4 Low Risk

[L-1] Missing basic validation on duel options and option prices

Description:
LibBento.validateDuelParameters does not check that entries in _options are non-empty strings, while joinDuel later enforces that the stored option string for an index is non-empty. In addition, BentoCoreFacet.joinDuel does not explicitly reject _optionPrice == 0, relying instead on an eventual division by _optionPrice to revert.

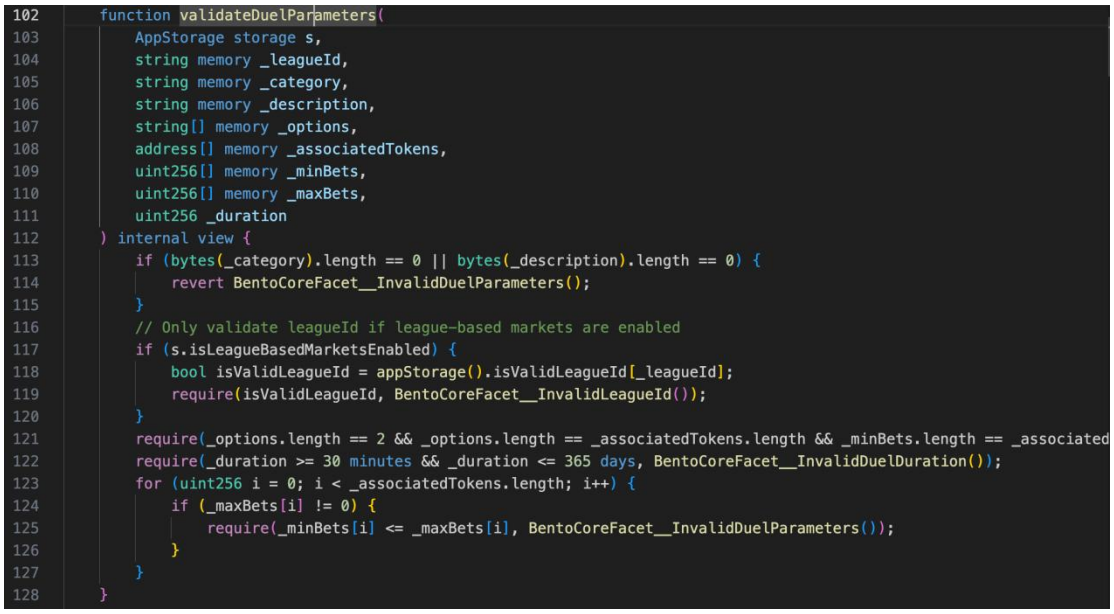
facets/BentoCoreFacet.sol Line#162

```
149     function joinDuel(
150         string memory _duelId,
151         string memory _option,
152         uint256 _optionsIndex,
153         uint256 _optionPrice,
154         uint256 _amount,
155         uint256 _amountInUsdc,
156         address _associatedToken,
157         address _user
158     ) external nonReentrant whenNotPaused onlyBot {
159         require(_optionsIndex < 2 && _user != address(0) && _associatedToken != address(0), BentoCoreFacet__InvalidJo
160         // Check if the option is valid
161         string memory option = s.duelIdToOptions[_duelId][_optionsIndex];
162         require(bytes(option).length > 0 && keccak256(bytes(option)) == keccak256(bytes(_option)), BentoCoreFacet__In
163         require(s.optionIndexToAssociatedToken[_duelId][_optionsIndex] == _associatedToken, BentoCoreFacet__InvalidAs
164 
165         Duel memory duel = s.duels[_duelId];
166         LibBento.LibBentoAppStorage storage libBentoStorage = LibBento.appStorage();
```

facets/BentoCoreFacet.sol Line#204

```
190         require(IERC20(_associatedToken).transferFrom(_user, address(this), _amount), BentoCoreFace
191         // uint256 amountTokenToMint = (_amountInUsdc * 1e18) / (_optionPrice);
192         // Collect protocol fee at join time (flat 0.3% = 30 BPS)
193         uint256 protocolFeeToken = (_amount * s.feePercentages.protocolFee) / BPS;
194         // uint256 protocolFeeUsdc = (_amountInUsdc * s.feePercentages.protocolFee) / BPS;
195         // uint256 netAmountToken = _amount - protocolFeeToken;
196         // uint256 netAmountInUsdc = _amountInUsdc - protocolFeeUsdc;
197         uint256 netAmountToken = _amount - protocolFeeToken;
198         uint256 netAmountInUsdc = (_amount > 0) ? (_amountInUsdc * netAmountToken) / _amount : 0;
199 
200         // Account protocol fee (kept in contract balance until withdrawn)
201         s.totalProtocolFeeGenerated[_associatedToken] += protocolFeeToken;
202 
203         // Mint option tokens against net USDC value
204         uint256 amountTokenToMint = (netAmountInUsdc * 1e18) / (_optionPrice);
```

libraries/LibBento.sol Line#102



```
102     function validateDuelParameters(
103         AppStorage storage s,
104         string memory _leagueId,
105         string memory _category,
106         string memory _description,
107         string[] memory _options,
108         address[] memory _associatedTokens,
109         uint256[] memory _minBets,
110         uint256[] memory _maxBets,
111         uint256 _duration
112     ) internal view {
113         if (bytes(_category).length == 0 || bytes(_description).length == 0) {
114             revert BentoCoreFacet__InvalidDuelParameters();
115         }
116         // Only validate leagueId if league-based markets are enabled
117         if (s.isLeagueBasedMarketsEnabled) {
118             bool isValidLeagueId = appStorage().isValidLeagueId[_leagueId];
119             require(isValidLeagueId, BentoCoreFacet__InvalidLeagueId());
120         }
121         require(_options.length == 2 && _options.length == _associatedTokens.length && _minBets.length == _associated
122         require(_duration >= 30 minutes && _duration <= 365 days, BentoCoreFacet__InvalidDuelDuration());
123         for (uint256 i = 0; i < _associatedTokens.length; i++) {
124             if (_maxBets[i] != 0) {
125                 require(_minBets[i] <= _maxBets[i], BentoCoreFacet__InvalidDuelParameters());
126             }
127         }
128     }
```

Impact:

Creators can accidentally deploy duels with empty option labels that only fail at join time, leading to inconsistent UX and harder debugging. A zero option price causes opaque "division by zero" errors instead of a clear validation revert, making misconfiguration more confusing for integrators and off - chain bots.

Recommendation:

Extend validateDuelParameters to require that each _options[i] has non-zero length, so invalid options are rejected at creation time. In joinDuel, add an explicit require(_optionPrice > 0, ...) with a descriptive error to guard against misconfiguration.

[L-2] Ambiguous use of zero as a sentinel for per-option minimum wager thresholds

Description:

The per-option minimum thresholds in s.optionMinimumWagerThresholds are treated as "0 = no limit" in both BentoAdminFacet.setOptionMinimumWagerThreshold and BentoViewFacet.checkIfThresholdMet (IndividualOption mode), whereas the global bootstrapLiquidityThreshold explicitly disallows 0 and must be non-zero. This asymmetric handling means that for some options a zero threshold is a deliberate "no constraint" sentinel, while elsewhere a zero threshold is forbidden.

facets/BentoViewFacet.sol Line#45

```
27      function checkIfThresholdMet(string calldata _duelId) public view returns (bool) {
28          // Private markets skip threshold checking
29          Duel memory duel = s.duels[_duelId];
30          if (duel.privacyAccess == PrivacyAccess.Private) {
31              return true;
32          }
33
34          uint256 optionsLength = s.duelIdToOptions[_duelId].length;
35
36          // Handle based on configured threshold check mode
37          if (s.thresholdCheckMode == ThresholdCheckMode.IndividualOption) {
38              // Mode 1: Check if ANY individual option meets its minimum threshold
39              for (uint256 i = 0; i < optionsLength; i++) {
40                  uint256 totalWagerForOption = s.totalWagerForOption[_duelId][i];
41                  address associatedToken = s.optionIndexToAssociatedToken[_duelId][i];
42                  uint256 optionMinThreshold = s.optionMinimumWagerThresholds[i][associatedToken];
43
44                  // If any option meets or exceeds its minimum threshold, return true
45                  if (totalWagerForOption >= optionMinThreshold) {
46                      return true;
47                  }
48              }
49              // If no option meets its threshold, return false
50              return false;
51
52          } else {
```

facets/BentoAdminFacet.sol Line#86

```
80      /// @notice Sets the minimum threshold for a specific option indices across all duels.
81      /// @dev This function can only be called by the contract owner.
82      /// It updates the optionMinThresholds mapping with the new threshold value for the specific option indices.
83      /// @param _optionIndices The indices of the options (0 = first option, 1 = second option, etc.).
84      /// @param _minThresholds The minimum thresholds for the specific option indices with decimals.
85      /// @param _associatedTokens The associated tokens for the specific option indices.
86      function setOptionMinimumWagerThreshold(
87          uint256[] memory _optionIndices,
88          uint256[] memory _minThresholds,
89          address[] memory _associatedTokens
90      ) external onlyOwnerOrBot {
91          require(_optionIndices.length ==  2 && _optionIndices.length == _minThresholds.length && _optionIndices.length
92          for (uint256 i = 0; i < _optionIndices.length; i++) {
93              s.optionMinimumWagerThresholds[_optionIndices[i]][_associatedTokens[i]] = _minThresholds[i];
94              emit OptionMinimumWagerThresholdUpdated(_optionIndices[i], _minThresholds[i], _associatedTokens[i]);
95          }
96      }
```

Impact:

Without clear, consistent semantics, protocol operators and integrators may misinterpret a zero per-option threshold as “no threshold configured” versus “misconfigured/invalid”, leading to unintended behavior (e.g., options that can always satisfy the IndividualOption threshold regardless of actual liquidity). This increases the risk of configuration mistakes and inconsistent UI/monitoring assumptions.

Recommendation:

Document explicitly that optionMinimumWagerThresholds[optionIndex][token] == 0 is a sentinel meaning “no per-option minimum”, while bootstrapLiquidityThreshold must be strictly positive. Optionally, enforce this in setOptionMinimumWagerThreshold (e.g., emit events and/or reject obviously accidental zero thresholds when leagues rely on IndividualOption mode), and reflect the semantics clearly in any admin tooling.

[L-3] Inconsistent and unsafe use of raw ERC20 transfers instead of SafeERC20

Description:

Several places in the codebase use bare IERC20.transfer / transferFrom / approve and manual require checks:

- LibBento.processRefundsInChunks uses IERC20(associatedToken).transfer(participant, wager);
- BentoCoreFacet.joinDuel uses IERC20(_associatedToken).transferFrom(_user, address(this), _amount);
- BentoAdminFacet._processWithdrawal, withdrawCreatorFee, withdrawCuratorFee, withdrawTokenAdminFee, withdrawAllFees, and withdrawProtocolFee all use raw transfer;
- LibSwap._approveTokensIfNeeded uses a plain approve instead of SafeERC20's forceApprove pattern.

e.g., facets/BentoCoreFacet.sol Line#190

```
180    uint256 decimals = IERC20Custom(_associatedToken).decimals();
181    require((_amount * 1e6) / (10 ** decimals) >= _optionPrice, BentoCoreFacet__LessThanMinimumWager());
182    uint256 minForOption = s.duelMinBetPerOption[_duelId][_optionsIndex];
183    if (minForOption != 0) {
184        require(_amount >= minForOption, BentoCoreFacet__LessThanMinimumWager());
185    }
186    uint256 maxForOption = s.duelMaxBetPerOption[_duelId][_optionsIndex];
187    if (maxForOption != 0) {
188        require(_amount <= maxForOption, BentoCoreFacet__GreaterThanMaximumWager());
189    }
190    require(IERC20(_associatedToken).transferFrom(_user, address(this), _amount), BentoCoreFacet__TokenTransferFailed());
```

Impact:

Some ERC20 tokens do not strictly follow the standard (e.g., no boolean return value, non - standard revert behavior). Using raw calls may cause unexpected reverts or false positives/negatives on allowance/transfer success, which can leave funds stuck or cause flows that appear to succeed but actually fail at the token level. The inconsistent use of SafeERC20 also makes reasoning about failure modes harder.

Recommendation:

Standardize on OpenZeppelin's SafeERC20 across the protocol: use safeTransfer, safeTransferFrom, and safeIncreaseAllowance / safeApprove / forceApprove as appropriate in all facets and libraries, replacing raw transfer/transferFrom/approve plus manual require checks.

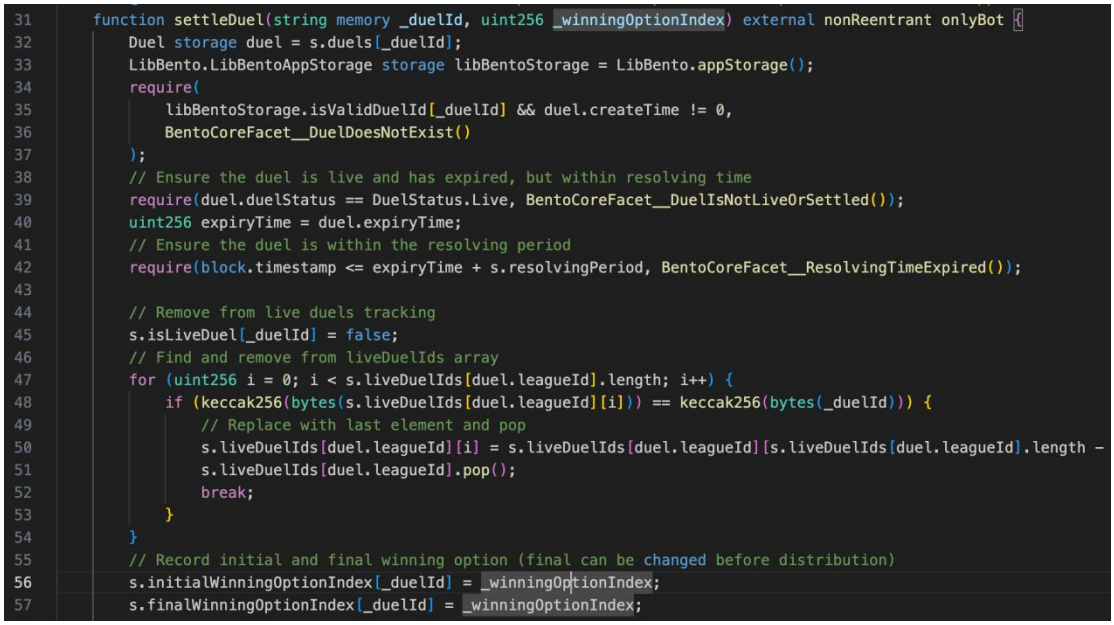
[L-4] settleDuel does not validate winning option index range

Description:

BentoSettlementFacet.settleDuel accepts _winningOptionIndex without checking that it is 0 or 1, while finalizeAndDistribute later enforces _finalWinningOptionIndex == 0 || _finalWinningOptionIndex == 1. An invalid index can thus be stored as

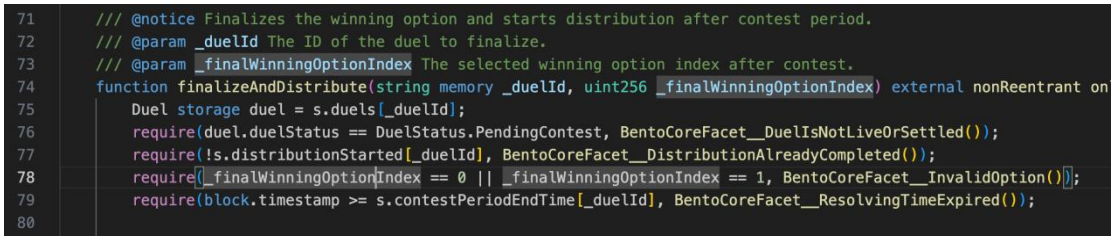
initialWinningOptionIndex / finalWinningOptionIndex in settleDuel before being rejected at finalization.

facets/BentoSettlementFacet.sol Line#56



```
31    function settleDuel(string memory _duelId, uint256 _winningOptionIndex) external nonReentrant onlyBot {
32        Duel storage duel = s.duels[_duelId];
33        LibBento.LibBentoAppStorage storage libBentoStorage = LibBento.appStorage();
34        require(
35            libBentoStorage.isValidDuelId[_duelId] && duel.createTime != 0,
36            BentoCoreFacet__DuelDoesNotExist()
37        );
38        // Ensure the duel is live and has expired, but within resolving time
39        require(duel.duelStatus == DuelStatus.Live, BentoCoreFacet__DuelIsNotLiveOrSettled());
40        uint256 expiryTime = duel.expiryTime;
41        // Ensure the duel is within the resolving period
42        require(block.timestamp <= expiryTime + s.resolvingPeriod, BentoCoreFacet__ResolvingTimeExpired());
43
44        // Remove from live duels tracking
45        s.isLiveDuel[_duelId] = false;
46        // Find and remove from liveDuelIds array
47        for (uint256 i = 0; i < s.liveDuelIds[duel.leagueId].length; i++) {
48            if (keccak256(bytes(s.liveDuelIds[duel.leagueId][i])) == keccak256(bytes(_duelId))) {
49                // Replace with last element and pop
50                s.liveDuelIds[duel.leagueId][i] = s.liveDuelIds[duel.leagueId][s.liveDuelIds[duel.leagueId].length -
51                s.liveDuelIds[duel.leagueId].pop();
52                break;
53            }
54        }
55        // Record initial and final winning option (final can be changed before distribution)
56        s.initialWinningOptionIndex[_duelId] = _winningOptionIndex;
57        s.finalWinningOptionIndex[_duelId] = _winningOptionIndex;
```

facets/BentoSettlementFacet.sol Line#78



```
71    /// @notice Finalizes the winning option and starts distribution after contest period.
72    /// @param _duelId The ID of the duel to finalize.
73    /// @param _finalWinningOptionIndex The selected winning option index after contest.
74    function finalizeAndDistribute(string memory _duelId, uint256 _finalWinningOptionIndex) external nonReentrant on
75        Duel storage duel = s.duels[_duelId];
76        require(duel.duelStatus == DuelStatus.PendingContest, BentoCoreFacet__DuelIsNotLiveOrSettled());
77        require(!s.distributionStarted[_duelId], BentoCoreFacet__DistributionAlreadyCompleted());
78        require(_finalWinningOptionIndex == 0 || _finalWinningOptionIndex == 1, BentoCoreFacet__InvalidOption());
79        require(block.timestamp >= s.contestPeriodEndTime[_duelId], BentoCoreFacet__ResolvingTimeExpired());
80
```

Impact:

Although finalizeAndDistribute will ultimately revert on an invalid index, an incorrect _winningOptionIndex in settleDuel can leave the duel stuck in PendingContest with confusing state and require an additional corrective transaction. This increases operational friction and makes error diagnosis harder for off-chain bots and operators.

Recommendation:

Add require(_winningOptionIndex == 0 || _winningOptionIndex == 1, ...) at the start of settleDuel, ensuring only valid option indices ever enter the contest/settlement pipeline. This aligns both settlement functions on the same invariant and surfaces configuration errors early.

[L-5] Settlement functions lack duel ID validity checks and uniform pause protection

Description:
BentoSettlementFacet.finalizeAndDistribute and continueWinningsDistribution assume a valid _duelId and rely on duel.duelStatus/distributionStarted/distributionCompleted checks, but do not explicitly verify isValidDuelId or duel.createTime != 0. In addition, these functions are not guarded by a whenNotPaused modifier, while other state-changing paths in the protocol are pause-protected via PausableUpgradeable.

facets/BentoSettlementFacet.sol Line#75

```
71    /// @notice Finalizes the winning option and starts distribution after contest period.
72    /// @param _duelId The ID of the duel to finalize.
73    /// @param _finalWinningOptionIndex The selected winning option index after contest.
74    function finalizeAndDistribute(string memory _duelId, uint256 _finalWinningOptionIndex) external nonReentrant on
75        Duel storage duel = s.duels[_duelId];
76        require(duel.duelStatus == DuelStatus.PendingContest, BentoCoreFacet__DuelIsNotLiveOrSettled());
77        require(!s.distributionStarted[_duelId], BentoCoreFacet__DistributionAlreadyCompleted());
78        require(_finalWinningOptionIndex == 0 || _finalWinningOptionIndex == 1, BentoCoreFacet__InvalidOption());
79        require(block.timestamp >= s.contestPeriodEndTime[_duelId], BentoCoreFacet__ResolvingTimeExpired());
```

Impact:
Calls with an invalid or typo'ed _duelId can revert with less clear or indirect error conditions, making operator mistakes harder to detect. The lack of pause gating means that, in an emergency pause scenario, settlement/distribution could still proceed while other write paths are frozen, undermining the intended safety guarantees of the pause mechanism.

Recommendation:
At the start of finalizeAndDistribute and continueWinningsDistribution, add explicit duel existence checks (e.g., isValidDuelId[_duelId] && s.duels[_duelId].createTime != 0). Also add whenNotPaused (or equivalent explicit pause checks) to these functions so that all critical state transitions, including settlement and payouts, respect the global pause state.

[L-6] Missing zero-address validation in DiamondInit.init

Description:
DiamondInit's init function sets important configuration addresses (e.g., USDC token, bot, protocol treasury, routers) on the diamond's shared storage, but does not enforce that these addresses are non-zero or otherwise sane at initialization time.

upgradeInitializers/DiamondInit.sol Line#77

```
48      function init(
49          address _protocolTreasury,
50          address _usdc,
51          address _bot,
52          address _universalRouter,
53          address _hooks,
54          address _permit2,
55          uint24 _dynamicFee,
56          int24 _tickSpacing
57      ) external onlyOwner initializer {
58          // adding ERC165 data
59          LibDiamond.DiamondStorage storage ds = LibDiamond.diamondStorage();
60          ds.supportedInterfaces[type(IERC165).interfaceId] = true;
61          ds.supportedInterfaces[type(IDiamondCut).interfaceId] = true;
62          ds.supportedInterfaces[type(IDiamondLoupe).interfaceId] = true;
63          ds.supportedInterfaces[type(IERC173).interfaceId] = true;
64
65          // add your own state variables
66          // EIP-2535 specifies that the `diamondCut` function takes two optional
67          // arguments: address _init and bytes calldata _calldata
68          // These arguments are used to execute an arbitrary function using delegatecall
69          // in order to set state variables in the diamond during deployment or an upgrade
70          // More info here: https://eips.ethereum.org/EIPS/eip-2535#diamond-interface
71
72          __Pausable_init();
73          __ReentrancyGuard_init();
74
75          // Initialize contract state variables
76          s.protocolTreasury = _protocolTreasury;
77          s.usdc = _usdc;
78          s.bot = _bot;
79          // s.bootstrapPeriod = 30 minutes;
80          s.resolvingPeriod = 48 hours;
81          s.winnersChunkSize = 15;
82          s.refundChunkSize = 30;
```

Impact:
If an incorrect or zero address is passed accidentally during deployment or an upgrade-initializer run, the protocol can be left in a misconfigured state where critical functionality (e.g., transfers, swaps, bot operations) silently fails at runtime. Such misconfiguration might only be detected after going live, making remediation costlier.

Recommendation:
Add explicit require(addr != address(0), ...) checks for all critical address parameters in DiamondInit.init (USDC, bot, protocol treasury, Universal Router, Permit2, hooks, etc.).

[L-7] Insufficient validation of admin-set configuration parameters

Description:
Several admin functions accept configuration values without sanity checks:
- BentoAdminFacet.setUniversalRouter, setHooks, setPermit2 accept arbitrary addresses without non-zero or contract checks;
- setDynamicFee and setTickSpacing do not validate fee tiers or tick spacing ranges;

- BentoMarketplaceFacet.updateSellerFees and updateBuyerFees accept arbitrary fee values without upper bounds in BPS.

e.g., facets/BentoAdminFacet.sol Line#197

```
193     /// @notice Sets the Universal Router contract address for swaps
194     /// @dev This function can only be called by the contract owner
195     /// @param _universalRouter The address of the Universal Router contract
196     function setUniversalRouter(address _universalRouter) external onlyOwner {
197         s.universalRouter = _universalRouter;
198         emit UniversalRouterUpdated(_universalRouter);
199     }
200
201
202     /// @notice Sets the Hooks contract address for Uniswap V4 pools
203     /// @dev This function can only be called by the contract owner
204     /// @param _hooks The address of the Hooks contract
205     function setHooks(address _hooks) external onlyOwner {
206         s.hooks = _hooks;
207         emit HooksUpdated(_hooks);
208     }
```

Impact:
Misconfiguration (e.g., pointing to the wrong router, setting an unsupported fee tier or zero tick spacing, or configuring extremely high marketplace fees) can render swaps and the marketplace unusable or economically unreasonable, effectively DoS-ing core protocol features without any on-chain guardrails.

Recommendation:
Introduce basic validation in these admin setters:
- Require non-zero addresses and (ideally) code presence for router/hooks/Permit2;
- Restrict dynamicFee and tickSpacing to known-good values or documented ranges;
- Enforce reasonable upper bounds on sellerFees and buyerFees in basis points (e.g., <= 1000 BPS or whatever your business rules allow).

4.5 Informational

[I-1] Inconsistent comments across multiple functions

- BentoCoreFacet._createDuel - Comment claims a USDC creation fee is required, but no fee is charged; update or remove the comment.

- BentoCoreFacet._revokePendingDuel - Comment mentions returning a refund amount and USDC checks, but the function only returns bool and does no refund; fix the comment.

- BentoCoreFacet.joinDuel - The _amountInUsdc parameter is undocumented in the NatSpec; add a brief description of its meaning and unit.

- BentoAdminFacet.setRefundChunkSizes - Comment says the chunk size is 10-25, but code enforces 20-50; align the comment with the actual bounds.

- BentoSettlementFacet.settleDuel - Comment refers to a default >= 12h contest period that is not enforced or initialized; either implement that default or update the comment.

- BentoAdminFacet.withdrawCuratorFee - Comment incorrectly says “duel creators” instead of “curators”; fix the role name.

[I-2] Comment mismatch for _totalWinnings in calculateFeesAndPayoutForSwappedTokens

In LibBento.calculateFeesAndPayoutForSwappedTokens, the comment describes _totalWinnings as “original + swapped” but BentoSettlementFacet.finalizeAndDistribute passes only swappedAmount, so the comment should be updated to reflect the actual usage.

[I-3] Unused local variable in finalizeAndDistribute

In BentoSettlementFacet.finalizeAndDistribute, the totalWagerToBeReturnedBack variable is assigned but never used and should be removed to avoid dead code.

[I-4] Leftover Hardhat console logging

In BentoAdminFacet and BentoViewFacet (e.g., updateWithdrawalRequestStatus), the import "hardhat/console.sol"; and console.log statements are debug artifacts and should be removed for production deployments.

[I-5] Duplicate duel getter in BentoViewFacet

In BentoViewFacet, getDuel and getDuels return the same s.duels[_duelId], so getDuels can be removed to avoid redundant API surface.