

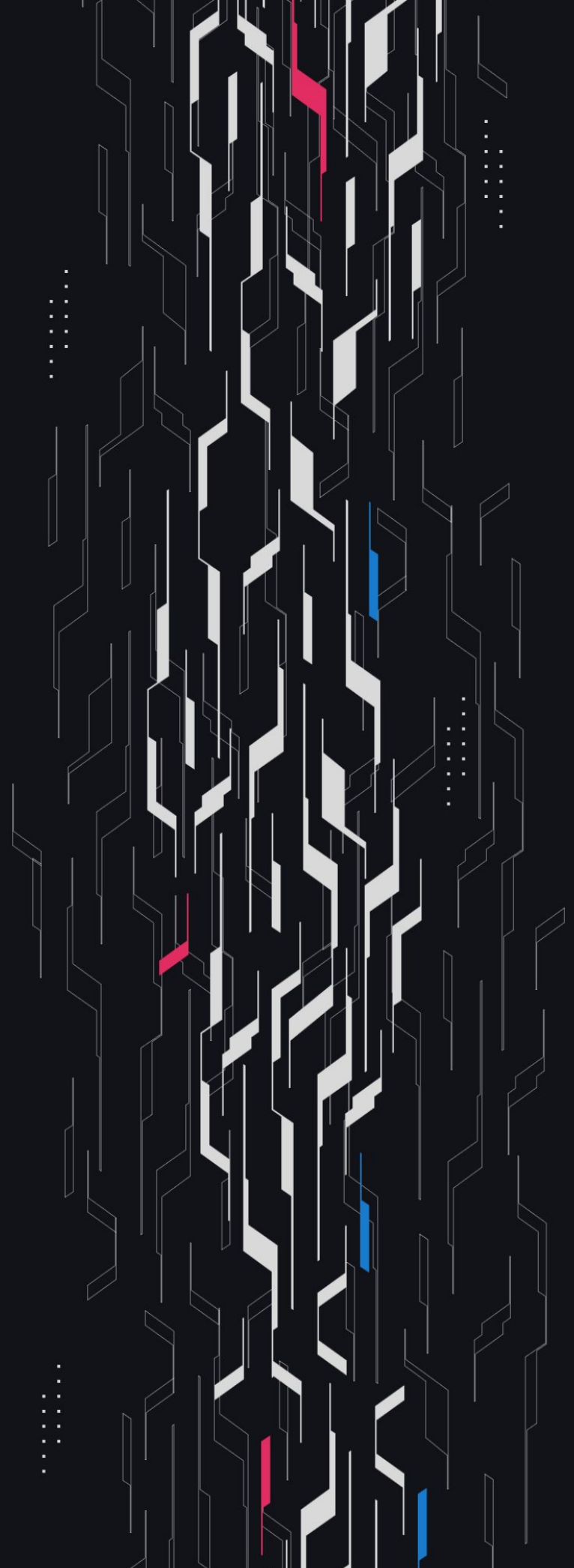
GA GUARDIAN

Gamma

**Unilaunch -
LaunchPad +
LimitOrderBook**

Security Assessment

March 5th, 2026



Summary

Audit Firm Guardian

Prepared By Michael Lett, CrypticDefense, Jakub Zmyslowski, 0xCiphky

Client Firm Gamma

Final Report Date March 5, 2026

Audit Summary

Gamma engaged Guardian to review the security of their Unilaunch - LaunchPad + LimitOrderBook. From the 2nd of January 2026 to the 4th of February 2026, a team of 4 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

Given the minimal High and Critical issues detected during the main review, Guardian assigns a Confidence Ranking of 4 to the protocol. Guardian advises the protocol to consider periodic review with future changes. For detailed understanding of the Guardian Confidence Ranking, please see the rubric on the following page.

Note: Fixes to the findings uncovered in the remediation round 2 review have not been reviewed by Guardian.

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

Guardian Confidence Ranking

Confidence Ranking	Definition and Recommendation	Risk Profile
5: Very High Confidence	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. Recommendation: Code is highly secure at time of audit. Low risk of latent critical issues.	0 High/Critical findings and few Low/Medium severity findings.
4: High Confidence	Code is clean, well-structured, and adheres to best practices. Only 1 Significant issue was uncovered per week. Design patterns are sound, and test coverage is strong. Recommendation: Suitable for deployment after remediations; consider periodic review with changes.	0-1 High/Critical findings per engagement week and little to no Medium severity issues. Varied Low severity findings.
3: Moderate Confidence	Medium-severity and occasional High-severity issues found. Code is functional, but there are concerning areas (e.g., weak modularity, risky patterns). No critical design flaws, though some patterns could lead to issues in edge cases. Recommendation: Address issues thoroughly and consider a targeted follow-up audit depending on code changes.	1-2 High/Critical findings per engagement week.
2: Low Confidence	Code shows frequent emergence of Critical/High vulnerabilities. Audit revealed recurring anti-patterns, weak test coverage, or unclear logic. These characteristics suggest a high likelihood of latent issues. Recommendation: Post-audit development and a second audit cycle are strongly advised.	2-4 High/Critical findings per engagement week. Or additional High/Critical findings uncovered in remediation review which have not been resolved and confirmed by Guardian.
1: Very Low Confidence	Code has systemic issues. Multiple High/Critical findings (≥ 5/week), poor security posture, and design flaws that introduce compounding risks. Safety cannot be assured. Recommendation: Halt deployment and seek a comprehensive re-audit after substantial refactoring.	≥ 5 High/Critical findings and overall systemic flaws.

Table of Contents

Project Information

Project Overview 5

Audit Scope & Methodology 6

Smart Contract Risk Assessment

Findings & Resolutions 9

Addendum

Disclaimer 41

About Guardian 42

Project Overview

Project Summary

Project Name	Gamma
Language	Solidity
Codebase	https://github.com/GammaStrategies/TokenLaunch-OrderbookFactory-MultiPositionManager
Commit(s)	Main Review commit: a75e25f78f0600d500f22fe3d7bd6e74c9955b0e Remediation Review commit: c1890ffcf3655a22c7bb444b87b3aa9fa7bd3a31 Remediation 2 Review commit: f248c7ab5be9e7cafe25d2f7fac0ea2987d997f4

Audit Summary

Delivery Date	March 5, 2026
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	0	0	0	0	0	0
● High	7	0	0	0	2	5
● Medium	4	0	0	1	2	1
● Low	5	0	0	1	0	4
● Info	5	0	0	1	0	4

Audit Scope & Methodology

```
contract,source,total,comment
src/MultiPositionManager/MultiPositionDeployer.sol,25,71,37
src/MultiPositionManager/MultiPositionFactory.sol,178,368,145
src/MultiPositionManager/MultiPositionManager.sol,285,427,93
src/MultiPositionManager/Relayer.sol,371,777,291
src/MultiPositionManager/RelayerDeployer.sol,42,80,34
src/MultiPositionManager/RelayerFactory.sol,264,531,193
src/LimitOrderBook/OrderBookFactory.sol,500,766,175
src/LaunchPad/SuperchainLBPStrategy.sol,135,211,43
src/LaunchPad/SuperchainLBPStrategyDeployer.sol,43,74,28
src/LaunchPad/SuperchainLBPStrategyFactory.sol,75,135,38
src/LaunchPad/TokenLaunchFactory.sol,123,227,76
src/MultiPositionManager/base/Multicall.sol,21,40,13
src/MultiPositionManager/base/SharedStructs.sol,37,58,27
src/MultiPositionManager/libraries/DepositLogic.sol,214,363,106
src/MultiPositionManager/libraries/PoolManagerUtils.sol,339,434,26
src/MultiPositionManager/libraries/PositionLogic.sol,365,573,142
src/MultiPositionManager/libraries/RebalanceLogic.sol,852,1384,347
src/MultiPositionManager/libraries/RelayerLogic.sol,248,405,110
src/MultiPositionManager/libraries/WithdrawLogic.sol,343,620,194
src/LimitOrderBook/errors/Errors.sol,7,9,1
src/LaunchPad/libraries/LBPDeploymentLib.sol,42,62,16
source count: {
  total: 7615,
  source: 4509,
  comment: 2135,
  single: 972,
  block: 1163,
  mixed: 57,
  empty: 1028,
  todo: 0,
  blockEmpty: 0,
  commentToSourceRatio: 0.4734974495453537
}
```

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: <i>High</i>	Impact: <i>Medium</i>	Impact: <i>Low</i>
Likelihood: <i>High</i>	● Critical	● High	● Medium
Likelihood: <i>Medium</i>	● High	● Medium	● Low
Likelihood: <i>Low</i>	● Medium	● Low	● Low

Impact

- High**

Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.
- Medium**

A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.
- Low**

Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

- High**

The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.
- Medium**

An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.
- Low**

Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

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

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

The auditing process pays special attention to the following considerations:

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

Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	Deterministic Token Address Collision DoS	Frontrunning	● High	Resolved
H-02	Launch Griefing Via CREATE2 Strategy Predeploy	Access Control	● High	Resolved
H-03	Arbitrary auctionFactory Enables Token Siphon	Validation	● High	Resolved
H-04	Front-Run Pool Init To Block Limit Order Support	DoS	● High	Resolved
H-05	Migration Blocked Via Hook Owner Assignment	DoS	● High	Resolved
H-06	MPM Name Collision Bricks Migration	Logical Error	● High	Partially Resolved
H-07	Permissionless Factory Enables Migration DoS	DoS	● High	Partially Resolved
M-01	Hook Owner Can Brick Swaps Via Fee Params	Validation	● Medium	Partially Resolved
M-02	Pool Reservation Griefing Via Front-Running	DoS	● Medium	Partially Resolved
M-03	One-Sided Token Toggle Ignored On Migrate	Unexpected Behavior	● Medium	Acknowledged
M-04	No-Strategy Branch In Factory Always Reverts	Logical Error	● Medium	Resolved
L-01	Missing CCA Token And Supply Validation	Validation	● Low	Resolved
L-02	Fee Bypass Via Direct LiquidityLauncher Call	Validation	● Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
L-03	Token Launch Index Can Be Overwritten	Logical Error	● Low	Resolved
L-04	Migrate Does Not Enforce Auction Graduation	Logical Error	● Low	Resolved
L-05	Missing Hook Flag Checks In Dynamic Hooks	Validation	● Low	Resolved
I-01	Presale Function Should Reuse _launchPresale	Best Practices	● Info	Resolved
I-02	Migration Event Uses Wrong PoolKey	Events	● Info	Resolved
I-03	Pool Creator Stored As Strategy Contract	Informational	● Info	Resolved
I-04	Ambiguous Registry Update Event	Events	● Info	Resolved
I-05	Silent Failure On Staticcall Returns Address 0	Validation	● Info	Acknowledged

H-01 | Deterministic Token Address Collision DoS

Category	Severity	Location	Status
Frontrunning	● High	TokenLaunchFactory.sol	Resolved

Description [PoC](#)

When a user calls `deployTokenAndLaunchPresale` on `TokenLaunchFactory` contract, the launchpad deploys the token (via the selected token factory) and then transfers `launchAmount` to the `LiquidityLauncher`, while sending the remaining supply (`totalSupply - launchAmount`) to `msg.sender`.

Because `TokenLaunchFactory` is always the caller, and because `graffiti` (and for `USUPERC20` also `creator`) is constant, an attacker can front-run a victim by calling `deployTokenAndLaunchPresale` first with the same visible token identity fields.

The victim's transaction then reverts due to a `CREATE2` collision and the front-runner receive the non-launch portion of supply. Consequently, any user can be permanently DoS'd from launching a desired launchpad, and an attacker can hijack a launch and receive the non-auction token allocation.

Recommendation

Make the `CREATE2` namespace unique per user/launch by using a non-constant `graffiti` derived from the original caller (and ideally a user-provided salt/nonce), and pass that into the token factory.

Resolution

Gamma Team: Resolved.

H-02 | Launch Griefing Via CREATE2 Strategy Predeploy

Category	Severity	Location	Status
Access Control	● High	SuperchainLBPStrategyDeployer.sol: 27	Resolved

Description [PoC](#)

In `SuperchainLBPStrategyDeployer` contract, `deploy` function is externally callable and permissionless, and deploys `SuperchainLBPStrategy` via `CREATE2` using a caller-supplied salt. In the intended flow, `LiquidityLauncher.distributeToken` derives a deterministic salt and passes it to `SuperchainLBPStrategyFactory.initializeDistribution`, so the strategy address becomes deterministic for a given salt and constructor inputs.

An attacker can observe a pending launch transaction, recover the same salt and constructor inputs, and front-run by calling `SuperchainLBPStrategyDeployer.deploy` directly. This pre-deploys `SuperchainLBPStrategy` at the deterministic `CREATE2` address the factory would later use.

When the legitimate launch proceeds, the factory attempts to deploy the same contract at the same `CREATE2` address via the deployer, but the address is already occupied, causing a `CREATE2` collision and reverting. This allows a denial-of-service against every launches.

Even if `SuperchainLBPStrategyDeployer.deploy` is restricted, the same collision/DoS can still be triggered by calling `SuperchainLBPStrategyFactory.initializeDistribution` directly with the same salt and `configData`, since `initializeDistribution` is also externally callable and not restricted to `LiquidityLauncher`.

Recommendation

Restrict the `SuperchainLBPStrategyDeployer.deploy` function so only the trusted `SuperchainLBPStrategyFactory` can call it, and restrict `SuperchainLBPStrategyFactory.initializeDistribution` so only the canonical `LiquidityLauncher` can call it.

Resolution

Gamma Team: Resolved.

H-03 | Arbitrary auctionFactory Enables Token Siphon

Category	Severity	Location	Status
Validation	● High	LBPStrategyBasic.sol: 126	Resolved

Description [PoC](#)

The launch flow allows creator to provide `MigratorParameters.auctionFactory`, and this address is forwarded into the deployed strategy unchanged.

The strategy then fully trusts that factory deploys a correct Continuous Clearing Auction and to follow the expected behaviour around raised funds, clearing price, and recipient handling.

In `LBPStrategyBasic` contract, during `onTokensReceived` execution, the strategy calls the user-specified factory and then transfers the auction's token supply to the returned contract.

A malicious creator can therefore point `auctionFactory` to a custom factory that returns a fake auction contract which can siphon the auction token supply, accept bidders' currency and make it unrecoverable, sweep it to an attacker or lock and misdirect the auction supply and raised currency.

Recommendation

Do not allow arbitrary `auctionFactory` in launches. Enforce a trusted allowlist (or a single immutable factory) in the `TokenLaunchFactory`.

Resolution

Gamma Team: Resolved.

H-04 | Front-Run Pool Init To Block Limit Order Support

Category	Severity	Location	Status
DoS	● High	OrderBookFactory.sol: 588	Resolved

Description [PoC](#)

OrderBookFactory allows users to create Uniswap V4 pools with limit order support, allowing users to register their pools with various different hook addresses.

An attacker can front-run pool creation by calling PoolManager.initialize() directly in Uniswaps PoolManager contract with the extracted pool parameters. The victim's transaction then reverts at _initializePool() since the pool already exists, rolling back all hook registration and configuration.

The resulting pool exists in Uniswap V4 but permanently lacks the intended dynamic fee/limit order support.

No recovery path exists since the registerPool() function within the hook contracts is only callable by OrderBookFactory, and no admin functions can manually configure the hook state.

The only hook that is not vulnerable to this is VolatilityDynamicFeeLimitOrderHook, which checks access control within the beforeInitialize hook.

Recommendation

For each hook that can be registered through OrderBookFactory, include a beforeInitialize hook to ensure msg.sender is equal to the OrderBookFactory contract, allowing only this contract to register the pools with the intended hooks.

Resolution

Gamma Team: Resolved.

H-05 | Migration Blocked Via Hook Owner Assignment

Category	Severity	Location	Status
DoS	● High	OrderBookFactory.sol: 308	Resolved

Description [PoC](#)

During a token launch, the system reserves a specific poolId in advance so that the later migration step can reuse that same poolId.

The createVolatilityDynamicLimitOrderPoolWithManager function derives the poolId inside computePoolIdForVolatilityLimitOrder, which depends on the hook address among other parameters. If creatorVolatilityDynamicLimitOrderHooks[hookOwner] is already set, the function uses the existing hook address.

Because createVolatilityDynamicLimitOrderPoolWithManager is permissionless, a user can call it using the victim's hookOwner before migration runs. This sets creatorVolatilityDynamicLimitOrderHooks[hookOwner] ahead of time. When the migration later executes, computePoolIdForVolatilityLimitOrder uses the pre-set hook address and derives a different poolId than the one reserved during launch.

If the attacker created a pool with the same parameters, migration reverts inside VolatilityDynamicLimitOrderHook.registerPool with PoolParametersAlreadyUsed. If the attacker used different tokens or parameters, migration proceeds but treats the pool as unreserved and charges the creation fee, reverting with InsufficientETH.

This allows any user to grief a token launch by invalidating its pool reservation and causing migration to fail.

Recommendation

Modify the token launch process to bind the hook address to the reserved pool at launch so it can't be changed later.

Resolution

Gamma Team: Resolved.

H-06 | MPM Name Collision Bricks Migration

Category	Severity	Location	Status
Logical Error	● High	MultiPositionFactory.sol: 133	Partially Resolved

Description [PoC](#)

In `deployMultiPositionManager`, the `MultiPositionFactory` contract enforces a global uniqueness on the `name` string for every `MultiPositionManager` deployment via `usedNames` mapping. Any attempt to deploy another manager with the same name reverts.

`LaunchPad` migration supplies a user-chosen `mpmName`, which is forwarded into `MultiPositionFactory.deployMultiPositionManager`.

However, because this function is permissionless and the uniqueness check is global (not scoped to a pool or owner), an attacker can deploy an arbitrary MPM using the victim's intended `mpmName` (read from the deployed strategy).

This permanently sets `usedNames[mpmName] = true`. When `migrate` later attempts to deploy the MPM using the same `mpmName`, the call reverts with `NameAlreadyUsed` error, and migration cannot be completed, permanently blocking the launchpad's intended migration flow.

Recommendation

Remove the global `usedNames[name]` revert entirely and treat name as non-unique metadata.

Alternatively, keep name uniqueness but scope it to the owner (e.g. implement `mapping(address => mapping(string => bool)) usedNamesByOwner`) so each `managerOwner` can only use a given name once.

Resolution

Gamma Team: Partially Resolved.

H-07 | Permissionless Factory Enables Migration DoS

Category	Severity	Location	Status
DoS	● High	OrderBookFactory.sol: 770	Partially Resolved

Description [PoC](#)

MultiPositionFactory functions deployMultiPositionManager, deployDepositAndRebalance, deployDepositAndRebalanceSwap are all permissionless.

The deployDepositAndRebalance. function is called from OrderBookFactory during token launch migrations, which proceeds to deploy a new MultiPositionManager contract using poolKey, managerOwner, and name for the salt.

The problem is an attacker can front-run the migration and directly call deployMultiPositionManager which will proceed to deploy the same MultiPositionManager contract first. The victim's migration call will revert due to address collision.

This issue is distinct from the MPM Name Collision finding - even if name uniqueness validation is removed, the CREATE2 address collision would still cause the deployment to fail. Both fixes are required for complete mitigation.

Note that the attacker cannot execute the front-run attack for functions deployDepositAndRebalance and deployDepositAndRebalanceSwap because it requires the (reserved) pool to be initialized first.

Recommendation

Restrict deployDepositAndRebalance and deployDepositAndRebalanceSwap to only be callable by OrderBookFactory (preventing DoS to pool creation with limit order support), and change deployMultiPositionManager visibility to internal.

Resolution

Gamma Team: Partially Resolved.

M-01 | Hook Owner Can Brick Swaps Via Fee Params

Category	Severity	Location	Status
Validation	● Medium	VolatilityDynamicFeeLimitOrderHook.sol VolatilityDynamicFeeHook.sol	Partially Resolved

Description

VolatilityDynamicFeeLimitOrderHook and VolatilityDynamicFeeHook updates the pool’s dynamic LP fee inside beforeSwap by calling poolManager.updateDynamicLPFee. Uniswap v4 enforces MAX_LP_FEE = 1000000 (100%), and updateDynamicLPFee reverts if the passed fee exceeds that maximum.

The hook does not validate that its configured parameters (baseFee, surgeMultiplier, surgeDuration) can never produce a totalFee above this cap.

On every swap, the hook computes a time-decaying surge fee and adds it to the fixed baseFee, then attempts to write the sum as the pool’s LP fee. If the sum exceeds the v4 maximum, updateDynamicLPFee reverts and the entire swap reverts.

A malicious (or careless) hook owner can brick trading by setting baseFee greater than MAX_LP_FEE (permanent swap reverts), or by choosing parameters where baseFee + surgeFee exceeds the cap during surge mode.

Recommendation

On registerPool, updateBaseFee, and updateSurgeParams, enforce parameter constraints so totalFee can never exceed MAX_LP_FEE.

Resolution

Gamma Team: Partially Resolved.

M-02 | Pool Reservation Griefing Via Front-Running

Category	Severity	Location	Status
DoS	● Medium	OrderBookFactory.sol: 1127	Partially Resolved

Description PoC

An attacker can permanently DoS a victim's token launch via front-run by calling `LiquidityLauncher.distributeToken()` directly with the victim's `configData` without owning any of the victim's launch token.

This means `sorted(TOKEN, USDC) == sorted(USDC, TOKEN)`, allowing an attacker to flip the pair.

Attack Flow:

1. Victim submits launch for (token=LAUNCH_TOKEN, currency=USDC)
 2. Attacker sees pending tx and extracts `hookOwner`, `hookSalt`, `tickSpacing`
 3. Attacker calls `LiquidityLauncher.distributeToken()` directly with flipped parameters:
`token = USDC` (attacker holds this) `migratorParams.currency = LAUNCH_TOKEN`
1. Both compute identical `poolId` since currencies are sorted.
 2. Attacker reserves `poolId` with dust amount (2 wei USDC), and victim's tx reverts with `PoolAlreadyReserved`:

Although `LiquidityLauncher` derives a unique salt as `keccak256(abi.encode(msg.sender, userSalt))`, this salt is not used in the `poolId` computation. Only the user-provided `hookOwner` and `hookSalt` from `configData` are used, causing the victim's transaction to revert:

```
if (reservedPools[poolId] != address(0)) revert PoolAlreadyReserved();
```

This attack does not work for ETH-paired launches (`currency = address(0)`) because `LiquidityLauncher.distributeToken()` requires an `ERC20` token and cannot accept `ETH` for the distribute token.

The victim must change their pool parameters to launch, while the attacker can repeat this indefinitely at minimal cost.

Recommendation

Utilize the caller-unique salt parameter in `SuperchainLBPStrategyFactory.initializeDistribution()` when deriving the `poolId`. Ensure access control is restricted only to `LiquidityLauncher`.

Resolution

Gamma Team: Partially Resolved.

M-03 | One-Sided Token Toggle Ignored On Migrate

Category	Severity	Location	Status
Unexpected Behavior	● Medium	SuperchainLBPStrategy.sol: 152	Acknowledged

Description

In the base LBP implementation, the token transfer amount is conditional - it only transfers the full `reserveSupply` when a one-sided position is actually going to be created (encoded as `data.hasOneSidedParams`), otherwise it transfers only what is needed for the main position and leaves leftover tokens inside the strategy.

In `SuperchainLBPStrategy`, `_createPositionPla` is overridden to return empty, so `hasOneSidedParams` is never produced and the strategy replaces the transfer sizing with `_calculateTransferAmounts`, but the token side is computed.

Because `data.initialTokenAmount` is always less than or equal to `reserveSupply` by construction, this effectively always transfers the whole `reserveSupply`. This means that when `reserveSupply > data.initialTokenAmount` and the user has `createOneSidedTokenPosition == false` (so a one-sided token position is not desired), `SuperchainLBPStrategy` still transfers all reserved tokens.

In the migration path, those amounts are then used as the deposit amounts into the `OrderBookFactory` pool and MPM creation call.

The factory pulls the tokens from the strategy and deposits them into the newly deployed `MultiPositionManager`, so the extra tokens do not remain in the strategy for later sweeping as intended, and they are moved into the MPM regardless of the one-sided token setting.

Recommendation

Adjust `_calculateTransferAmounts` functions so leftover reserved tokens are only included when a one-sided token position is intended.

Resolution

Gamma Team: Acknowledged.

M-04 | No-Strategy Branch In Factory Always Reverts

Category	Severity	Location	Status
Logical Error	● Medium	RebalanceLogic.sol: 158	Resolved

Description

In `OrderBookFactory`, `_deployMultiPositionManager` includes a branch to support deploying an MPM with `rebalanceParams.strategy == address(0)` by calling `deployDepositAndRebalance` on `MultiPositionFactory`.

However, this function always deploys a fresh MPM and immediately executes `MultiPositionManager.rebalance` right after deployment.

Since a newly deployed MPM has no previously stored strategy, `RebalanceLogic` deterministically reverts with `NoStrategySpecified` when the passed strategy is zero.

This makes the “no strategy” branch unreachable and breaks integrations that try to create a pool and MPM without a strategy.

Recommendation

Add an explicit initialization path that deploys and deposits without rebalancing (skip the initial rebalance when `strategy == address(0)`) and route the mentioned branch to it.

Resolution

Gamma Team: Resolved.

L-01 | Missing CCA Token And Supply Validation

Category	Severity	Location	Status
Validation	● Low	TokenLaunchFactory.sol: 270	Resolved

Description

The Uniswap CCA documentation states the following:

1. "Do NOT use the Auction with low-decimal (< 6) tokens. Bidders will lose significant amounts of token due to rounding errors in price and amount calculations."
2. "The maximum total supply that can be sold in the auction is 1e30 wei of token. For a token with 18 decimals, this is 1 trillion tokens."

TokenLaunchFactory does not validate that launched tokens have at least 6 decimals before forwarding to Uniswap's ContinuousClearingAuction. The CCA explicitly warns that low-decimal tokens cause bidders to lose significant amounts due to rounding errors in price calculations.

In addition, there is no validation regarding the total supply for the auction (other than checking against type(uint128).max, which is insufficient in this case). Since the maximum that can be sold is 1e30, sending more can lead to unexpected behaviour or stuck funds.

Recommendation

Add decimal and total auction supply validation, for example:

```
uint8 decimals = IERC20(token).decimals();
if (decimals < 6) revert TokenDecimalsTooLow();
if (auctionSupply > 1e30) revert SupplyTooLarge();
```

Resolution

Gamma Team: Resolved.

L-02 | Fee Bypass Via Direct LiquidityLauncher Call

Category	Severity	Location	Status
Validation	● Low	SuperchainLBPStrategyFactory.sol: 65	Acknowledged

Description

Upon presale token launches, the `TokenLaunchFactory` contract collects a `creationFee` before calling `LiquidityLauncher::distributeToken`, which proceeds to call `SuperchainLBPStrategyFactory::initializeDistribution`.

This deploys a new strategy for the token, and then reserves the pool for the strategy. After this, the launch is officially registered.

A user can bypass the creation fee by directly calling `LiquidityLauncher::distributeToken`. This will still process the strategy deployment and pool reservation, allowing users to launch tokens with full protocol functionality while paying zero fees, causing a loss of revenue for the protocol.

Recommendation

Within `initializeDistribution`, check if the creation fee has already been paid for the given unique launch parameters.

Resolution

Gamma Team: Acknowledged.

L-03 | Token Launch Index Can Be Overwritten

Category	Severity	Location	Status
Logical Error	● Low	TokenLaunchFactory.sol: 463	Resolved

Description

Token launches are indexed in a single `_tokenToIndex` mapping that uses the token address as a unique key, but the mapping is overwritten on every new launch.

The registration path always updates `_tokenToIndex[token]` without checking whether that token was already registered, while `_allLaunches` retains all entries.

Because `launchPresaleWithExistingToken` is permissionless, any holder of the token can create a second launch and overwrite the index. Consequently, integrators that rely on `getLaunchByToken` will be misled.

Recommendation

Consider supporting multiple launches per token by replacing `_tokenToIndex` with a per-token array of indices.

Resolution

Gamma Team: Resolved.

L-04 | Migrate Does Not Enforce Auction Graduation

Category	Severity	Location	Status
Logical Error	● Low	LBPStrategyBasic.sol: 139	Resolved

Description

LBPStrategyBasic.migrate() can be executed once block.number >= migrationBlock and proceeds using auction-derived values (e.g., currencyRaised and clearingPrice). However, migration does not require that the auction has graduated (met requiredCurrencyRaised).

_validateMigration() only checks that currencyRaised > 0 and that the strategy holds at least currencyRaised of the auction currency, so migration can be forced even when the owner’s minimum-raise requirement is not met.

In particular, a dust bid that makes currencyRaised > 0, combined with the strategy contract being funded, can allow any third party (since migrate() is permissionless) to trigger listing and liquidity deployment despite the auction failing its graduation condition.

This is especially impactful when launching with an existing token, since a failed auction would otherwise allow the owner to reuse that token allocation in a future launch; premature migration can consume the reserved supply and commit the token to an unintended listing event.

Recommendation

Require auction graduation as a prerequisite to migration. For example, in _validateMigration() enforce auction.isGraduated().

Resolution

Gamma Team: Resolved.

L-05 | Missing Hook Flag Checks In Dynamic Hooks

Category	Severity	Location	Status
Validation	● Low	OrderBookFactory.sol	Resolved

Description

Dynamic limit-order and fee-only hooks rely on CREATE2 salts to achieve required Uniswap v4 hook flags.

Unlike the volatility variants, these creation paths do not verify that the deployed hook’s address actually has the required flags, allowing pools to be created with hooks that will not receive necessary callbacks, breaking fee logic/limit-orders.

Recommendation

After deploying the dynamic limit-order/dynamic fee hook, validate the deployed address against the required Hook flags (as done for volatility hooks) and revert if they don’t match.

Resolution

Gamma Team: Resolved.

I-01 | Presale Function Should Reuse _launchPresale

Category	Severity	Location	Status
Best Practices	● Info	TokenLaunchFactory.sol: 203-220	Resolved

Description

launchPresaleWithExistingToken duplicates the logic already present in _launchPresale rather than reusing the internal helper. Only the token transfer mechanism differs (transferFrom vs transfer). Consolidating shared logic into a single internal function is a best practice.

Recommendation

Add a boolean parameter to _launchPresale to differentiate transfer type:

```
function _launchPresale(
LaunchParams calldata params,
address token,
bool useTransferFrom
) internal returns (address) {
...
if (useTransferFrom) {
IERC20(token).safeTransferFrom(msg.sender, address(strategy), tokenAmount);
} else {
IERC20(token).safeTransfer(address(strategy), tokenAmount);
}
...
}
```

Then update both external functions:

```
function deployTokenAndLaunchPresale(...) external returns (...) {
address token = _deployToken(...);
return _launchPresale(params, token, false);
}
function launchPresaleWithExistingToken(...) external returns (...) {
return _launchPresale(params, token, true);
}
```

Resolution

Gamma Team: Resolved.

I-02 | Migration Event Uses Wrong PoolKey

Category	Severity	Location	Status
Events	● Info	LBPStrategyBasic.sol: 150	Resolved

Description

The SuperchainLBPStrategy inherits the base migration flow, which emits Migrated(key, data.sqrtPriceX96) event using the PoolKey returned by _initializePool.

In SuperchainLBPStrategy, this function is overridden to intentionally avoid initializing the pool and returns a placeholder PoolKey with hooks = address(0) and fee = poolLPFee.

However, the actual pool created during migration is deployed and initialized by OrderBookFactory in createVolatilityDynamicLimitOrderPoolWithManager, which builds a different PoolKey using the deployed hook address and the dynamic fee flag.

As a result, the Migrated event emitted during migrations process reports a pool identity that does not correspond to the real pool used by the deployed MultiPositionManager, what can mislead off-chain integrators that rely on this event.

Recommendation

Consider overriding migrate function in SuperchainLBPStrategy to emit migration metadata or emit a specific event from _transferAssetsAndExecutePlan after the factory call, using the PoolKey returned by createVolatilityDynamicLimitOrderPoolWithManager.

Resolution

Gamma Team: Resolved.

I-03 | Pool Creator Stored As Strategy Contract

Category	Severity	Location	Status
Informational	● Info	OrderBookFactory.sol: 594	Resolved

Description

OrderBookFactory stores the pool creator as msg.sender during initialization.

```
poolCreators[poolId] = msg.sender;
userToPoolInfo[msg.sender].push(PoolInfo({
  poolId: poolId,
  poolKey: poolKey,
  isDynamic: isDynamic
}));
```

In the LaunchPad flow, msg.sender is the deployed strategy contract, not the initial creator. This causes pools created via LaunchPad to be indexed under the strategy address, so getPoolsByCreator will not return the pool and off-chain attribution becomes misleading.

Recommendation

Consider recording the pool creator as hookOwner or managerOwner (and optionally also storing msg.sender separately) so LaunchPad pools remain discoverable under the expected creator address.

Resolution

Gamma Team: Resolved.

I-04 | Ambiguous Registry Update Event

Category	Severity	Location	Status
Events	● Info	OrderBookFactory.sol: 910	Resolved

Description

OrderBookFactory uses the same DynamicFeeRegistryUpdated event for two different setters, setDynamicFeeLimitOrderRegistry and setDynamicFeeRegistry, which makes the emitted logs indistinguishable.

Off-chain indexers and monitoring tools cannot determine whether the dynamic fee registry or the dynamic fee limit-order registry was updated what may lead to ambiguity.

Recommendation

Emit a distinct event for each function so off-chain consumers can unambiguously identify which registry was updated.

Resolution

Gamma Team: Resolved.

I-05 | Silent Failure On Staticcall Returns Address 0

Category	Severity	Location	Status
Validation	● Info	TokenLaunchFactory.sol: 362	Acknowledged

Description

TokenLaunchFactory uses staticcalls to fetch the auction address and token metadata, but silently continues when the call fails:

```
(bool success, bytes memory data) =
strategy.staticcall(abi.encodeWithSignature("auction()"));
if (success && data.length >= 32) {
    auction = abi.decode(data, (address));
}
```

If the staticcall fails, auction is registered with address(0) for the launch, thus making the factory's indexing and query functions incorrect for the launch's auction.

Recommendation

Consider reverting on staticcall failure.

Resolution

Gamma Team: Acknowledged.

Remediation Findings & Resolutions V1

ID	Title	Category	Severity	Status
H-01	Migration Blocked Via Front-Run Attack	DoS	● High	Resolved
M-01	Pool Reservation Bypass Via Duplicate poolId	DoS	● Medium	Resolved
M-02	Token Launch Flip Attack Still Possible	DoS	● Medium	Resolved
M-03	Migration Blocked By Fee Validation Mismatch	DoS	● Medium	Resolved
M-04	Hook Disable Bypassed Via OrderBookFactory	Validation	● Medium	Resolved
I-01	Unbounded Pool Creation Fee	Informational	● Info	Acknowledged

H-01 | Migration Blocked Via Front-Run Attack

Category	Severity	Location	Status
DoS	● High	MultiPositionFactory.sol: 351-359	Resolved

Description [PoC](#)

The `_ensureFreshManager` check in `deployDepositAndRebalance` can be bypassed to block token migrations.

During token launch, the `pool key` is reserved and the hook restricts initialization to the `OrderBookFactory` as `msg.sender`, meaning the pool can only be initialized during the migration call itself.

However, an attacker can front-run the migration by calling `deployDepositAndRebalance` with a dust deposit and a `malicious strategy` that specifies zero base ranges and `limitWidth=0`.

Because the pool is `uninitialized`, a normal rebalance would revert when minting liquidites, but with zero base ranges the minting step is skipped entirely, and `limitWidth=0` skips limit ranges, allowing the rebalance to succeed without the pool needing to be initialized.

This marks the MPM as non-fresh (`totalSupply != 0`, strategy is set), causing the victim's migration transaction to revert at `_ensureFreshManager`.

Recommendation

At the beginning of rebalance, revert if the pool has not yet been initialized:

```
(uint160 sqrtPriceX96,,, ) = poolManager.getSlot0(poolKey.toId());
if (sqrtPriceX96 == 0) {
  revert PoolNotInitialized(poolKey);
}
```

Resolution

Gamma Team: Resolved.

M-01 | Pool Reservation Bypass Via Duplicate poolId

Category	Severity	Location	Status
DoS	● Medium	OrderBookFactory.sol: 1224-1230	Resolved

Description

The updated `OrderBookFactory::reservePoolForStrategy` function keys reservations on `keccak256(abi.encode(poolId, callerSalt))` instead of `poolId` alone.

This allows two callers to reserve the same `poolId` by supplying different `callerSalt` values, bypassing the `PoolAlreadyReserved` check. An attacker can front-run `launchPresaleWithExistingToken()`, using the victim's same `token`, `positionRecipient`, and `hookSalt` (which determine the `poolId`) but with a different `salt` (`callerSalt`), producing a distinct `reservationKey` for the same pool.

Both reservations succeed, but during migration only the first to execute will initialize the pool. The second migration reverts at `poolManager.initialize()` since the pool already exists, effectively blocking the victim's migration.

New token launches via `deployTokenAndLaunchPresale()` are only affected if the attacker can obtain some of the launch token, which cannot be done until the auction is complete (front-run not possible since token address is derived from `msg.sender` via graffiti). This makes it such that it is very unlikely the attacker can execute `migration()` prior to the victim, thus mitigating the attack.

Recommendation

Key the reservation on `poolId` alone, as was done in the previous implementation.

Resolution

Gamma Team: Resolved.

M-02 | Token Launch Flip Attack Still Possible

Category	Severity	Location	Status
DoS	● Medium	OrderBookFactory.sol: 1224-1230,	Resolved

Description

The original finding, M-04, reported that an attacker could permanently DoS a victim's token launch via front-run by calling `LiquidityLauncher.distributeToken()` with flipped currency parameters, producing the same `poolId` (since `sorted(TOKEN, USDC) == sorted(USDC, TOKEN)`) and causing the victim's transaction to revert with `PoolAlreadyReserved`.

The recommendation was to include the caller-unique salt in the `poolId` derivation itself.

The implemented fix instead added `callerSalt` to the reservation key (`keccak256(abi.encode(poolId, callerSalt))`), rather than incorporating it into the `poolId`.

Although this prevents the immediate `PoolAlreadyReserved` revert (both callers now get distinct reservation keys), the `poolId` still remains identical for both parties.

As a result, both reservations succeed, but during migration only the first to execute will initialize the pool via `poolManager.initialize()`. The second migration reverts because the pool already exists.

Recommendation

Include `callerSalt` in the `hookSalt` used for `poolId` computation so each caller produces a unique pool: `SuperchainLBPStrategyFactory_reservePool()`:

```
bytes32 effectiveHookSalt = keccak256(abi.encode(hookSalt, salt));
bytes32 poolId = orderBookFactory.computePoolIdForVolatilityLimitOrder(
hookOwner, effectiveHookSalt, currency0, currency1, migratorParams.poolTickSpacing
);
```

Resolution

Gamma Team: Resolved.

M-03 | Migration Blocked By Fee Validation Mismatch

Category	Severity	Location	Status
DoS	● Medium	LBPStrategyBasic.sol: 201	Resolved

Description

The launch-time fee validation in LBPStrategyBasic accepts poolLPFee up to LPFeeLibrary.MAX_LP_FEE (1,000,000), but migration always routes through volatility dynamic limit-order pool creation with a hardcoded surgeMultiplier = 30000 (3x).

During migration, VolatilityDynamicFeeLimitOrderHook.registerPool enforces:
 $baseFee + (baseFee * surgeMultiplier / 10000) \leq MAX_LP_FEE$
With surgeMultiplier = 30000, this becomes $4 * baseFee \leq 1,000,000$, so baseFee must be $\leq 250,000$.

As a result, configurations with poolLPFee > 250,000 pass initial launch validation and auction proceeds normally, but migration deterministically reverts with InvalidFeeConfiguration() at pool registration.

This creates a permanent migration failure for affected launches because strategy parameters are immutable post-deployment.

Recommendation

Add an upfront validation in launch initialization to enforce compatibility with the hook constraints.

Resolution

Gamma Team: Resolved.

M-04 | Hook Disable Bypassed Via OrderBookFactory

Category	Severity	Location	Status
Validation	● Medium	OrderBookFactory.sol: 574	Resolved

Description

The `_initializePool` function in `OrderBookFactory` unconditionally calls `limitOrderManager.enableHook(address(poolKey.hooks))` whenever a pool is created, with no check for whether that hook was already initialized in earlier pool setups or explicitly disabled by the admin.

`LimitOrderManager.disableHook()` is intended to set `isHook[_hook] = false`, which blocks `createLimitOrder()`, `createScaleOrders()`, and `executeOrder()` with `DisabledHook`.

However, pool creation is permissionless, and `enableHook()` allows calls from the `orderBookFactoryAddr`. This means any external user can create a new pool that references the same hook address and, in doing so, re-enable the hook after it was disabled by an admin.

Consequently, an admin cannot rely on `disableHook` as an emergency stop. If a hook is disabled during an incident, it can be reactivated without admin approval, restoring order placement/execution paths that were intentionally shut down.

Recommendation

Add a hard disabled state (or equivalent) that `enableHook()` cannot override for factory-triggered calls.

Resolution

Gamma Team: Resolved.

I-01 | Unbounded Pool Creation Fee

Category	Severity	Location	Status
Informational	● Info	OrderBookFactory.sol: 881-894	Acknowledged

Description

OrderBookFactory allows DEFAULT_ADMIN_ROLE to set the pool creation fees for dynamic and volatility-dynamic pool types to any uint256 value, without an upper bound or activation delay.

These values are then enforced as a strict msg.value minimum during pool creation, so if the admin sets an excessively high fee, callers will fail the InsufficientETH check and pool creation will revert, effectively pausing those creation flows until the fee is corrected.

Recommendation

Add a reasonable maximum cap for these pool creation fees to prevent accidental misconfiguration from halting pool creation.

Resolution

Gamma Team: Acknowledged.

Remediation Findings & Resolutions V2

ID	Title	Category	Severity	Status
I-01	Pools Can Be Deployed With Disabled Hooks	Warning	● Info	Acknowledged

I-01 | Pools Can Be Deployed With Disabled Hooks

Category	Severity	Location	Status
Warning	● Info	LimitOrderManager.sol: 917	Acknowledged

Description

The original issue where a user could create a new pool with the same hook and effectively re-enable an admin-disabled hook is now fixed.

However, pool creation still proceeds even if that hook stays disabled. This means a pool can be deployed while its hook is still disabled, and limit-order execution and swap paths can later revert with DisabledHook until admin re-enables the hook.

Recommendation

Add clear UI/doc warnings so users can see when a hook is disabled and avoid launching pools against disabled hooks.

Resolution

Gamma Team: Acknowledged.

Disclaimer

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

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

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

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

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

About Guardian

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

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

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

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