

FLOWROUTER · SECURITY REVIEW

FlowRouter Protocol Review.

A security review of Protocol Review.

CLIENT	PROJECT	AUDITORS	AUDIT WINDOW	FINAL REPORT
FlowRouter	Protocol Review	Roberto Reigada, Vlad Zotov, Michael Lett, Ferran Celades, OxCiphky .	September 3 – September 21, 2026	September 21, 2026

§ 00

Contents.

47 pages: overview, confidence ranking, scope, 28 findings, disclaimer, and about Guardian.

§ 01	Overview	P. 03
§ 02	Confidence Ranking	P. 04
§ 03	Scope & Classifications	P. 05
	Files in Scope	P. 06
§ 04	Findings	P. 07
	Main Review · 28 findings	P. 09
§ 05	Disclaimer	P. 46
§ 06	About Guardian	P. 47

§ 01

Overview.

Guardian reviewed Protocol Review across 2 review rounds. The final report includes 10 Low, 18 Informational findings.

AUDIT FIRM	REPOSITORY
Guardian	https://github.com/GammaStrategies/FlowRouter
CLIENT	METHODOLOGY
FlowRouter	Smart contract · Manual · Main Review · Remediation Review
PROJECT	MAIN REVIEW COMMIT
Protocol Review	3b6ce6b14bd8
PREPARED BY	REMEDATION REVIEW COMMIT
Roberto Reigada, Vlad Zotov, Michael Lett, Ferran Celades, OxCiphky .	e6413301a269
AUDIT WINDOW	
September 3 – September 21, 2026	
FINAL REPORT	
September 21, 2026	

1.1 Confidence Ranking

5 / 5

Very High Confidence

Guardian assigns a Confidence Ranking of 5 out of 5, combining finding severity, remediation outcome, and the reviewer’s assessment of latent risk. The full scale is on the next page.



1.2 Findings at a Glance

SEVERITY	TOTAL	PENDING	ACK	PARTIAL	RESOLVED
■ Critical	0	0	0	0	0
■ High	0	0	0	0	0
■ Medium	0	0	0	0	0
■ Low	10	0	2	0	8
■ Informational	18	0	16	0	2
All severities	28	0	18	0	10

§ 02

Confidence Ranking.

Guardian assigns a project-level confidence score using the scale below. The ranking combines finding severity, remediation outcome, and the reviewer's assessment of latent risk.

CONFIDENCE RANKING	DEFINITION AND RECOMMENDATION	RISK PROFILE
5 Very High Confidence ASSIGNED	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.	0 High/Critical findings and few Low/Medium severity findings.
4 High Confidence	Code is clean, well-structured, and adheres to best practices. 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 findings.
3 Moderate Confidence	Code is functional but shows gaps in defensive programming or test coverage. Recommendation: remediate the findings and re-review the changes before deployment.	Some High/Critical findings, with various Medium severity findings.
2 Low Confidence	Recurring issues suggest systemic gaps rather than isolated mistakes. Recommendation: remediate, then commission a follow-up review at a frozen commit.	Over 6 High and Critical findings across the engagement.
1 Very Low Confidence	The volume and severity of findings indicate the codebase is not ready for production. Recommendation: substantial rework, then a full re-review.	Over 12 High and Critical findings across the engagement.

§ 03

Scope & Classifications.

Guardian reviewed Protocol Review in

<https://github.com/GammaStrategies/FlowRouter>. The scope includes 2,207 source lines across 6 files.

3.1 Severity Matrix

	IMP · HIGH	IMP · MED	IMP · LOW
LIK · HIGH	Critical	High	Medium
LIK · MED	High	Medium	Low
LIK · LOW	Medium	Low	Low

IMPACT · HIGH

Significant loss of assets, harm to a group of users, or core protocol functionality is disrupted.

IMPACT · MEDIUM

Small amount of funds can be lost, or ancillary functionality is affected. Users may experience reduced or delayed receipt.

IMPACT · LOW

Unexpected behavior in some functionality that is notable but does not meet the criteria for higher severity.

LIKELIHOOD · HIGH

Possible with reasonable assumptions that mimic on-chain conditions; cost is low compared to amount gained.

LIKELIHOOD · MEDIUM

Possible only in uncommon cases or requires a large amount of capital relative to disruption caused.

LIKELIHOOD · LOW

Unlikely to ever occur in production.

Files in Scope.

FILE	NSLOC	TOTAL
MAIN REVIEW		
contracts/FlowRouter.sol	218	321
contracts/libraries/EkuboV30peration.sol	41	49
contracts/libraries/TokenFlowProgram.sol	65	82
contracts/OnlyApproved.sol	20	27
contracts/TokenFlowExecutorV2.sol	1,606	1,879
contracts/TokenHelper.sol	23	39
REMEDIATION REVIEW		
contracts/FlowRouter.sol	11	329
contracts/libraries/EkuboV30peration.sol	16	70
contracts/libraries/TokenFlowProgram.sol	3	87
contracts/OnlyApproved.sol	0	27
contracts/TokenFlowExecutorV2.sol	204	1,979
contracts/TokenHelper.sol	0	39

§ 04

Findings.

Grouped by round. Visible IDs reset by severity within each round, so a finding is named the same way in the report as it was in the channel it was written in. Status is where each finding ended up; its own page says how.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
MAIN REVIEW					
L-01	First upper half referral code rejects fees	Logical Error	LOW	RESOLVED	p. 9
L-02	Zero amount bypasses spender revocation	Access Control	LOW	RESOLVED	p. 10
L-03	Native V2 multi hop quotes always revert	Compatibility	LOW	RESOLVED	p. 11
L-04	V4 swaps cannot supply required hook data	Compatibility	LOW	RESOLVED	p. 12
L-05	Legacy Ekubo router rejects Ve33 pools	Compatibility	LOW	RESOLVED	p. 13
L-06	Pinned Pons factory rejects current curves	Compatibility	LOW	RESOLVED	p. 15
L-07	V4 partial fills trigger delta mismatches	Compatibility	LOW	RESOLVED	p. 17
L-08	Terminal routes strand split final output	Logical Error	LOW	ACKNOWLEDGED	p. 19
L-09	Native RFQ operations cannot consume variable AMM output	Compatibility	LOW	RESOLVED	p. 21
L-10	Self-settled V4 hook output bypasses fixed referral and protocol fees	accounting / hook integration	LOW	ACKNOWLEDGED	p. 22
I-01	Opaque Bebop calls expose executor authority	Validation	INFORMATIONAL	RESOLVED	p. 23
I-02	Renunciation erases explicit sweeper authority	Access Control	INFORMATIONAL	ACKNOWLEDGED	p. 25
I-03	Revocation permits referral code reassignment	Business logic errors	INFORMATIONAL	ACKNOWLEDGED	p. 26
I-04	Nested swaps can reclaim retained surplus	Reentrancy	INFORMATIONAL	RESOLVED	p. 27
I-05	Native referral beneficiaries can veto swaps	DoS	INFORMATIONAL	ACKNOWLEDGED	p. 28
I-06	Transient output custody retains token rewards	Rewards	INFORMATIONAL	ACKNOWLEDGED	p. 29
I-07	Metric OMM swaps cannot supply extension data	Compatibility	INFORMATIONAL	ACKNOWLEDGED	p. 31
I-08	Generated executor address is stale	Configuration	INFORMATIONAL	ACKNOWLEDGED	p. 33
I-09	Rialto requires optional token getters	Compatibility	INFORMATIONAL	ACKNOWLEDGED	p. 34
I-10	Native output to an executor reverts	Compatibility	INFORMATIONAL	ACKNOWLEDGED	p. 36
I-11	Fixed referral fees overflow on large outputs	Math	INFORMATIONAL	ACKNOWLEDGED	p. 37
I-12	Signed slippage casts reject large amounts	Math	INFORMATIONAL	ACKNOWLEDGED	p. 38

Findings continued.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
I-13	Pons refunds are retained by FlowRouter	Business logic errors	INFORMATIONAL	ACKNOWLEDGED	p. 39
I-14	V4 native settlement inherits stale sync	DoS	INFORMATIONAL	ACKNOWLEDGED	p. 40
I-15	Algebra swaps cannot supply plugin data	Compatibility	INFORMATIONAL	ACKNOWLEDGED	p. 42
I-16	Graduation and escrow accounting are incompatible with balance-changing quote tokens	fee-on-transfer accounting	INFORMATIONAL	ACKNOWLEDGED	p. 43
I-17	Callback ERC20 can capture in-flight V4 payments or poison settlement via unrestricted sync	reentrancy	INFORMATIONAL	ACKNOWLEDGED	p. 44
I-18	Positive quote-token balance growth enables zero-cost bonding-curve buys	cross-flow accounting / rebasing token	INFORMATIONAL	ACKNOWLEDGED	p. 45

LOW **L-01**

RESOLVED

First upper half referral code rejects fees

CATEGORY	Logical Error
LOCATION	contracts/FlowRouter.sol:187-200,262-268
REVIEW	Main Review

DESCRIPTION

The referral namespace reserves the lower half of `uint32` values for informational codes and allows upper half codes to apply a fixed fee. `REFERRAL_WITH_FEE_THRESHOLD` equals `1 << 31`, so `0x80000000` is the first value with its high bit set. However, `registerReferralCode()` checks `_referralCode <= REFERRAL_WITH_FEE_THRESHOLD`, which rejects every nonzero fee for that first upper half code. `_swap()` repeats the misclassification because it reads and applies a referral fee only when `referralCode > REFERRAL_WITH_FEE_THRESHOLD`.

An owner or referral allocator that assigns `0x80000000` as the first fee bearing code cannot configure the intended fee for it. Registration reverts with `InvalidFeeForCode`, so the integration either fails or must skip an otherwise valid identifier. If the code is registered with a zero fee, swaps using it also omit the fixed fee and prevent the intended protocol and referrer fee collection.

RECOMMENDATION

Treat `0x80000000` consistently as part of the upper half. Change the registration comparison from `<=` to `<` and the settlement comparison from `>` to `>=`.

RESOLUTION

Status: Resolved.

LOW **L-02**

RESOLVED

Zero amount bypasses spender revocation

CATEGORY	Access Control
LOCATION	contracts/TokenFlowExecutorV2.sol:1668-1671
REVIEW	Main Review

— DESCRIPTION

The owner can revoke a persistent token and spender pair by clearing its allowance and setting `persistentApprovalState[token][spender]` to the revoked state. The documented lifecycle requires that state to remain closed until the owner explicitly resets it. `_ensurePinnedAllowance()` instead returns immediately when the resolved operation amount is zero and reads the revocation state only afterward. Every fixed spender handler that resolves to zero can therefore continue into its external call despite the owner's revocation.

This is externally reachable through the permissionless FlowRouter. For example, an attacker can declare a zero input Bebop operation for a revoked token and supply `invalidateNonce(uint256)` as the opaque payload. The executor skips the revoked state check. The live pinned Bebop router records the chosen nonce against the executor because it sees the executor as `msg.sender`. An unrelated one wei native currency operation that wraps to WETH supplies the final output. This allows the route and nonce mutation to commit while `persistentApprovalState` still reports the pair as revoked.

The attacker can repeat the route for Bebop nonces used by executor quotes. Those quotes then revert even after the owner has invoked the documented administrative control. No token allowance is restored and this proof does not show protected fund loss. The impact is limited to defeating an explicit administrative revocation guarantee and enabling targeted quote denial of service.

— RECOMMENDATION

Read `persistentApprovalState[token][spender]` and reject the revoked state before returning for a zero amount. Keep the zero amount return only after the revocation check so fixed and dynamically admitted spenders enforce the same closed lifecycle.

— RESOLUTION

Status: Resolved.

LOW **L-03**

RESOLVED

Native V2 multi hop quotes always revert

CATEGORY	Compatibility
LOCATION	contracts/TokenFlowExecutorV2.sol:1201-1213,1543-1550
REVIEW	Main Review

— DESCRIPTION

The Native V2 operation is intended to execute an exact input RFQT quote through the pinned Native router. The executor accepts `quote.multiHop`, approves the router for the declared input and then requires the executor's input balance to decrease after `tradeRFQT()` returns.

The exact source that matches the deployed Native router uses a different input source for this quote mode. When `multiHop` is false, the router calls `transferFrom()` against `msg.sender`, which consumes the executor's approved balance. When `multiHop` is true, it calls `transfer()` from the Native router's own balance because that mode continues a sequence inside the Native router's multicall.

Consequently, the two accounting models cannot compose. A Native router without enough prefunded seller tokens reverts during its transfer. Even if the router is fully prefunded and sends the signed buyer output to the executor, the executor's seller token balance remains unchanged. `_finishPulledInput()` then reverts with `InputNotConsumed` and rolls back the complete swap. Every signed Native V2 quote with `multiHop = true` is therefore unusable through operation `200`, which wastes the submitter's transaction cost and makes the quoted route unavailable.

— RECOMMENDATION

Reject every Native V2 quote whose `multiHop` field is true before granting an allowance or calling the Native router. Restore a dedicated `NativeV2MultiHopUnsupported` error because the executor cannot supply Native router inventory within the venue's own multicall. Express multi step swaps as separate executor operations instead.

— RESOLUTION

Status: Resolved.

LOW **L-04**

RESOLVED

V4 swaps cannot supply required hook data

CATEGORY Compatibility

LOCATION `contracts/TokenFlowExecutorV2.sol:173-203,718-738,746-827,867-883,906-941`

REVIEW Main Review

— DESCRIPTION

Uniswap V4 passes caller-supplied `hookData` to a pool's hooks during each swap. Hooks can use this data to authenticate a trader, verify an offchain authorization, enforce a whitelist, or receive other pool-specific input. The local `PoolManager` interface exposes this argument, and the route schemas let the compiler select the complete pool key, including its hook address.

Neither V4 route schema can carry that payload. `UniswapV4PoolData` has no `hookData` field, and `UniswapV4BatchHop` has no per-hop equivalent. The derived single and batch callback jobs also omit it. `_uniswapV4RunSingle()` and `_uniswapV4RunBatch()` therefore pass a literal empty byte string to `PoolManager.swap()` for every pool and direction.

Consequently, any otherwise valid swap that requires nonempty hook data is impossible through the executor.

— RECOMMENDATION

Add versioned V4 single and batch operation encodings that include `bytes hookData` for a single swap and one `bytes hookData` value per batch hop. Propagate each value through the callback job and pass it unchanged to `PoolManager.swap()`.

Keep the current opcodes as explicitly empty-hook-data variants so existing version 2 route bytes retain their ABI meaning. A new opcode or route version is safer than appending dynamic fields to the existing structs because old encoders and decoders must continue to agree on the tuple layout. Preserve the current `PoolManager` pin, callback authentication, delta checks, debt settlement, balance floors, and output minima for the new variants.

— RESOLUTION

Status: Resolved.

LOW **L-05**

RESOLVED

Legacy Ekubo router rejects Ve33 pools

CATEGORY Compatibility

LOCATION `contracts/TokenFlowExecutorV2.sol:48-49,594-602; contracts/libraries/EkuboV30peration.sol:38-55`

REVIEW Main Review

DESCRIPTION

Ekubo operation `210` accepts a complete pool configuration and builds an exact input call for the selected pool. The configuration includes the pool extension address, so the operation can represent the live Ve33 pool format.

The executor always sends this call to `0xd26f20001a72a18C002b00e6710000d68700ce00`. Ekubo's [deployment reference](https://docs.ekubo.org/reference/contracts/evm-v3/) identifies this address as the legacy `v3.1.1` MEV Capture router that was superseded by the configurable router. The official `v3.1.1` `_swap()` implementation forwards only when the pool extension equals the router's configured `MEV_CAPTURE` address. Every other extension is sent directly to `Core.swap()`.

Ve33 is a forward only extension. Its official `v3.2.0` `beforeSwap()` implementation rejects every direct Core swap with `SwapMustHappenThroughForward()`. Consequently, every operation `210` route for a Ve33 pool reaches the legacy router's direct branch and reverts even though `EkuboV30peration` can encode that pool key.

This affects a live pool on the target chain. At Robinhood block `54,617,722`, Ekubo's current forwarded router quotes `1 STONX` for `763,332` raw USDG units through the initialized STONX/USDG Ve33 pool. The legacy router returns `SwapMustHappenThroughForward()` for the same tokens, amount and configuration. A fresh `TokenFlowExecutorV2` reaches the same error through `executePath()`. The transaction rolls back, so the user loses transaction cost and cannot use Ve33 liquidity through the Ekubo operation.

RECOMMENDATION

Deploy Ekubo's `v3.2.0` configurable `Router` for Robinhood with the existing Core, MEV Capture and Ve33 addresses then pin the next executor deployment to that router. The configurable router preserves the seven argument `swap()` ABI used by `EkuboV30peration`, routes ordinary pools through Core and forwards pools whose extension matches its `MEV_CAPTURE` or `VE33` immutable. Check the deployed `MEV_CAPTURE()` and `VE33()` values during deployment. Preserve the current allowance, native receive, input budget, output minimum and residue controls when changing the pinned address.

L-05 Legacy Ekubo router rejects Ve33 poolsCONTINUED

— RESOLUTION

Status: Resolved.

LOW **L-06**

RESOLVED

Pinned Pons factory rejects current curves

CATEGORY Compatibility

LOCATION `contracts/TokenFlowExecutorV2.sol:38,1860-1903; contracts/interfaces/IPonsV2.sol:26-36`

REVIEW Main Review

— DESCRIPTION

Pons operation `230` executes an exact input trade against a bonding curve after proving that its token and quote asset match a Pons launch record. This proof is meant to prevent a caller from granting approval to an unrelated curve.

`PONS_V2_LAUNCH_FACTORY` fixes the proof to `0x7E1EAbd52Ae29598e6483F72dCf1a70b14284dB8`.

`_validatePonsV2Route()` always queries that address even though each Pons curve exposes the immutable factory that created it. The curve interface omits `factory()`, so validation has no way to select among trusted Pons deployments.

Pons replaced the V2 stack before the reviewed executor was introduced. The [current integration documentation](https://docs.ponsfamily.com/v2) identifies `0x7eD598BcEf8bd9Edd8C97A195C6d13f40801EC7e` as the live factory. It also states that a replacement creates a whole new stack while tokens from earlier stacks continue to trade, so an integration must resolve the stack that each token belongs to. The official repository changed the factory listing on August 11, 2026. FlowRouter's initial audit package added operation `230` on August 12 with the predecessor address.

At Robinhood block `55,030,674`, native curve `0xE9103855888D6d574E69274694FA20e4562dFEf9` reports the current factory and token `0x1F3699741c9C9EE9069ccd97f754FF6A2e94f348`. The current factory returns the matching existing launch record while the pinned predecessor returns an empty record. A direct `0.001 ETH` buy succeeds and returns `509,833,553,483,831,890,982,152` token units. The identical executor operation reverts with `InvalidPonsV2Curve` before it calls the curve.

Every curve launched by the current Pons factory is therefore unavailable through operation `230`. A user whose route selects current Pons liquidity loses the transaction cost and cannot complete the quoted swap through FlowRouter.

— RECOMMENDATION

Add `factory()` to `IPonsV2BondingCurve` and deploy the next executor with an explicit immutable allowlist of the predecessor and current Pons factories. During route validation,

L-06 Pinned Pons factory rejects current curves

CONTINUED

read the curve's factory, require that it belongs to this allowlist and query `getLaunchedToken()` on that factory. Continue matching the record's token, curve and pair token before granting approval or admitting the route. Do not accept an unrestricted factory address from route bytes. This preserves operation 230 and its existing encoding while supporting both Pons generations.

— RESOLUTION

Status: Resolved.

LOW **L-07**

RESOLVED

V4 partial fills trigger delta mismatches

CATEGORY Compatibility

LOCATION `contracts/TokenFlowExecutorV2.sol:888-899,946-947,971-977`

REVIEW Main Review

— DESCRIPTION

Uniswap V4 exact-input swaps can stop at `sqrPriceLimitX96` before consuming the complete requested amount. The pinned v4-core swap loop exits when either the amount is exhausted or the price limit is reached, then returns only `amountSpecified - amountSpecifiedRemaining` as the caller's input delta. This can happen when a request exceeds the input that available pool liquidity can consume before the executor's hardcoded extreme price limit.

Both V4 execution branches reject that valid result. `_uniswapV4RunSingle()` requires `inDelta` to equal the full negative `job.amountIn`, while `_uniswapV4RunBatch()` requires each input delta to equal its full negative hop allocation and later requires any negative declared-input net to equal the complete operation allocation. The executor therefore reverts `UniswapV4DeltaMismatch` before settling a smaller nonzero debt, even when the partial output satisfies the applicable minima. A failure at any batch hop reverts the complete route.

A local test against v4-core commit `46c6834698c48bc4a463a86d8420f4eb1d7f3b75`, matched to the pinned PoolManager, requests `1e18` input units against ordinary narrow-range liquidity. PoolManager consumes `6,035,841,794,200,769` input units, produces `5,981,737,760,509,662` output units and completes unlock after the actual debt is settled. Separate executor tests model a 60-input/50-output fill against a 100-input allocation and a 49-output minimum. Both current branches revert, while a strict accounting control settles 60, takes 50 and retains 40 unused input.

Users cannot execute these otherwise valid V4 partial fills through FlowRouter and incur failed-transaction costs. This contradicts README's operation-accounting rule that a venue may consume less than its allocation.

— RECOMMENDATION

Treat each requested V4 exact-input amount as an upper bound. Require each returned input delta to represent nonzero consumption no greater than its resolved allocation, retain the existing positive-output and minimum checks, and settle the actual negative debt rather than the requested amount.

L-07 V4 partial fills trigger delta mismatches

CONTINUED

For batches, retain currency netting, reject negative net debt in any currency other than the declared settlement currency, and require any negative declared-input net to be no greater than `job.amountIn`. Settle that actual net debt. Preserve the documented treatment of zero-net or cyclic batches; imposing a new requirement that the final declared-input net must always be negative is a separate policy change, not necessary to fix rejection of nonzero partial fills.

Keep existing balance floors and residue handling. Unused referenced non-final input goes back to FlowRouter under that handling; this change does not introduce a wallet refund guarantee. Add regressions for single swaps and batches that consume less than their allocation while meeting output minima, plus overconsumption, insufficient output, unexpected-currency debt and existing cyclic-batch behavior.

— RESOLUTION

Status: Resolved.

LOW **L-08**

ACKNOWLEDGED

Terminal routes strand split final output

CATEGORY	Logical Error
LOCATION	contracts/TokenFlowExecutorV2.sol:408-441
REVIEW	Main Review

— DESCRIPTION

The finalizer for a terminal operation that can deliver output directly to `FlowRouter` is intended to measure the final token at both contracts. `_buildFinalOutputContext()` instead sets `skipExecutorAccounting` for every single operation route with `externalTake`. `_completeFinalOutput()` then leaves `producedOnExecutor` at zero. It checks `minFinalOutput` against only the router balance increase and never transfers newly received final tokens from the executor.

A Uniswap V4 hook receives the executor as the swap `sender` and can transfer ordinary tokens independently of the credit returned through `PoolManager`. If a hook supplies 99 units through `PoolManager` and sends one additional unit of the same final token to the executor, a 99 unit program minimum succeeds while the additional unit remains in the executor. A 100 unit program minimum reverts even though the operation produced 100 final token units across the two documented destinations.

This contradicts the documented terminal output behavior. It causes coherent routes whose quote includes both destinations to revert or converts part of their final output into a standing executor balance with no direct recovery function. The issue affects a single terminal operation listed by `_operationCanTakeFinalOutputExternally()` when its venue or hook pays part of the declared final token to the executor. Outer `FlowRouter.outputMin` remains enforced against the amount that reached the router, so this is a limited accounting and integration failure rather than a protected funds loss.

— RECOMMENDATION

Always snapshot the executor's final token balance in `_buildFinalOutputContext()`, including for single operation routes that use an external take. Sum the positive executor and router balance increases in `_completeFinalOutput()` and transfer the executor increase to `FlowRouter`. Remove `skipExecutorAccounting` so callbacks and venues cannot create untracked final output at the executor.

L-08 Terminal routes strand split final outputCONTINUED

— RESOLUTION

Status: Acknowledged.

LOW **L-09**

RESOLVED

Native RFQ operations cannot consume variable AMM output

CATEGORY	Compatibility
LOCATION	contracts/TokenFlowExecutorV2.sol:1180-1213,1562-1575
REVIEW	Main Review

— DESCRIPTION

Native supports adjusting a signed quote's input within bounded limits to consume the actual output of a preceding AMM swap. Its pool calculates the corresponding output proportionally.

The executor requires exact-input mode, checks that the resolved amount equals `quote.sellerTokenAmount`, and always passes zero as the actual-input override. Consequently, a quote signed for 100 tokens cannot consume 98 tokens returned by a preceding AMM. Allocating 100 fails the available-balance check, allocating 98 fails the quote-amount check, and full-balance mode is prohibited.

The route therefore reverts even when Native could execute the adjusted amount and satisfy all output minima. An included transaction costs the submitter gas and leaves the swap unexecuted. This concerns ordinary Native quotes with `multiHop = false` following separate executor operations.

— RECOMMENDATION

Add a versioned Native operation that resolves the available input and forwards it as `actualSellerAmount`, preserving Native's deviation limits and quote validation, the executor's input accounting, and all output minima.

— RESOLUTION

Status: Resolved.

LOW **L-10**

ACKNOWLEDGED

Self-settled V4 hook output bypasses fixed referral and protocol fees

CATEGORY	accounting / hook integration
LOCATION	packages/mirror-GammaStrategies-FlowRouter-01a03991/contracts/TokenFlowExecutorV2.sol:_uniswapV4RunSingle():L867-L904
REVIEW	Main Review

— DESCRIPTION

The V4 runners accept caller-selected hooks with return-delta permissions and use only PoolManager.swap's net caller BalanceDelta for terminal output. In afterSwap, a hook can call PoolManager.take(outputToken, recipient, X), creating a -X transient hook delta, then return +X as its unspecified hook delta. PoolManager credits that return to the hook, exactly cancelling the debt while subtracting X from the executor-visible delta. Unlock therefore finishes cleanly, but FlowRouter snapshots and charges referral/surplus economics only on the reduced output. The hook can activate conditionally when sender is the production executor, while a trusted gateway's precommitted route and terms remain unchanged.

Impact: An untrusted hook operator can direct nearly all V4 output to a chosen recipient and leave only the configured minimum visible to FlowRouter. This bypasses the fixed referral fee and protocol-retained fee share (and outputQuote surplus capture) on otherwise unchanged trusted-gateway swaps. The live PoC shows honest execution generating >1.5 tokens for the beneficiary and >0.3 protocol fee, while attacked execution of the same committed route yields only 0.016 and 0.004 respectively, with no unsettled PoolManager delta or calldata modification.

— RECOMMENDATION

Reject V4 PoolKeys whose hook address advertises BEFORE_SWAP_RETURNS_DELTA or AFTER_SWAP_RETURNS_DELTA in both single and batch paths, or admit such hooks only through a governance-reviewed allowlist with direct-payment behavior incorporated into referral/quote accounting. Ensure terminal output cannot bypass FlowRouter measurement.

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-FlowRouter-Audit---Round-1-01a07f4d5ff770c6bfe5dd9f81f76daf/blob/refinery/poc-01a08026-6e11-7137-8b5b-eb0e2363b1ef/test/refinery-poc/self-settled-v4-hook-output-bypasses-fixed-referral-and-prot-ff103665.sol>

— RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-01

RESOLVED

Opaque Bebop calls expose executor authority

CATEGORY	Validation
LOCATION	contracts/TokenFlowExecutorV2.sol:1151-1177
REVIEW	Main Review

DESCRIPTION

Bebop operation `250` is intended to perform an exact input swap through the pinned Bebop router. `_swapBebopRfqExactInput` does not require `operation.data` to select `IBebopRouter.swap` and does not decode the payload. Instead, it measures only the balances of the declared `tokenIn` and `tokenOut` before and after forwarding the payload through `BEBOP_ROUTER.call(operation.data)`.

Any caller can submit a route through FlowRouter that reaches operation `250`. Because the executor uses a low level call, the Bebop router treats the executor as the caller even when the payload selects another function. A payload can call `invalidateNonce` while the operation declares zero input. The declared input token remains unchanged, so the executor records zero spent and accepts the result. The caller can add a second operation that wraps one wei of native currency into WETH to satisfy FlowRouter's required nonzero final output. The transaction then completes and Bebop records the selected nonce as invalid for the executor, preventing a later RFQ order with the same nonce from executing.

The missing binding also affects token accounting. A normal Bebop operation gives the router a persistent maximum allowance. A later operation can declare zero units of an unrelated token while its Bebop payload pulls a previously approved token from the executor. Since the executor measures the unrelated declared token, it sees no balance change and accepts the expected zero spend. If the payload returns the declared output token, FlowRouter settles it to the caller. With a compatible signed order, the executor's standing balance can therefore be consumed without naming that token as the operation input.

These effects do not meet the README's protected funds eligibility rules. The README protects user wallet funds beyond authorized input, successful settlement of at least `outputMin` and FlowRouter balances. It excludes executor balance loss, preexisting executor inventory, persistent maximum approvals and denial of service without protected fund loss. This behavior affects executor inventory and executor owned Bebop state. The README also documents opaque Bebop forwarding as intended and assigns quote shape and token binding to the provider, compiler and gateway. The issue was therefore considered informational.

I-01 Opaque Bebop calls expose executor authority

CONTINUED

— RECOMMENDATION

No code change is necessary if opaque Bebop forwarding and its effects on executor inventory and executor owned state match the intended trust model. If operation `250` should only perform an exact input swap, require `IBebopRouter.swap.selector`, decode the payload and bind the signed order's input token, output token, token owner, receiver and exact amount to the resolved operation. Reject a zero resolved amount. Give any other supported Bebop method its own opcode and method specific validation.

— RESOLUTION

Status: Resolved.

INFORMATIONAL I-02

ACKNOWLEDGED

Renunciation erases explicit sweeper authority

CATEGORY Access Control

LOCATION contracts/OnlyApproved.sol:7-21; contracts/FlowRouter.sol:40-61

REVIEW Main Review

— DESCRIPTION

`FlowRouter` gives the owner implicit sweep authority while explicit sweeper grants are meant to remain active across ownership changes and renunciation until the owner removes them. Both sources of authority use the same boolean value in `OnlyApproved.approved`, so the contract cannot preserve them independently.

If an address already has an explicit sweeper grant when it becomes the owner, `_addApprovedAddress()` only writes `true` over the existing value. When that address later renounces ownership, `_removeApprovedAddress()` changes the shared value to `false`. This removes both the owner's implicit authority and the earlier explicit grant even though no one called `removeApprovedAddress()` for that grant.

A normal role transition reaches the failure: the current owner explicitly approves an independent sweeper, transfers ownership to that sweeper and the new owner later renounces. The contract clears the sweeper's approval and leaves `owner()` equal to the zero address. If no other sweeper remains, no account can withdraw existing or future ERC-20 and native balances from the router. No owner remains to restore the intended explicit grant, so those funds are permanently inaccessible.

— RECOMMENDATION

Store explicit sweeper grants separately from the owner's implicit authority. Make the sweep authorization check accept either the current owner or an explicitly approved sweeper and stop adding or removing owners in the explicit approval mapping during ownership transitions. This preserves a prior explicit grant when its holder becomes owner and later renounces while still revoking only the departing owner's implicit authority.

— RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-03

ACKNOWLEDGED

Revocation permits referral code reassignment

CATEGORY	Business logic errors
LOCATION	contracts/FlowRouter.sol:255-272,302-309
REVIEW	Main Review

DESCRIPTION

The referral registry is intended to prevent a code from being assigned more than once.

`registerReferralCode()` rejects a code whose `registered` field is true and its adjacent comment states that referral codes must not be overwritten. However, `revokeReferralCode()` deletes the complete `referralLookup` entry. This clears the same `registered` field that guards registration, so the revoked identifier immediately passes the registration check again with an unrelated beneficiary and fee.

This makes revocation change the identity of a code instead of only disabling its benefits. For example, the owner can revoke a one percent upper half code and later assign the same identifier to a new beneficiary with the maximum two percent fixed fee. Integrations and pending swaps that retain the old identifier then use the new terms. A swap whose `outputMin` still passes completes after the unexpected additional fee deduction and sends the beneficiary share to the replacement account.

RECOMMENDATION

Track whether each referral code has ever been assigned separately from whether its benefits are active. Set the permanent assignment marker during registration and have `registerReferralCode()` reject every previously assigned identifier. Revocation can then clear the active fee and surplus configuration without making the identifier available for reuse.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-04

RESOLVED

Nested swaps can reclaim retained surplus

CATEGORY	Reentrancy
LOCATION	contracts/FlowRouter.sol:179-185,208-213
REVIEW	Main Review

— DESCRIPTION

`FlowRouter` is intended to cap receiver settlement at `outputQuote` and retain its share of both positive execution surplus and fixed referral fees. The `_swap()` function snapshots the router output balance before it calls the executor then treats the entire balance increase after that call as output of the current swap. The executor can reenter either public swap entry point before the outer balance delta is calculated.

An unprivileged caller can start an outer swap whose executor uses the received input to perform a nested swap through the same router. If the nested execution returns 100 output tokens with an inner quote of 90, the inner frame pays 90 to its receiver and records the remaining 10 as the protocol share. When the nested call returns, the outer frame sees those 10 tokens as an increase over its earlier snapshot. It therefore treats the retained inner surplus as outer output and transfers it to the receiver controlled by the caller.

The same defect reaches fixed fee revenue even when positive slippage is zero. With 100 gross output, a one percent referral fee and an inner quote of 99, the inner frame pays 0.8 to the beneficiary and 99 to the receiver, leaving the router's 0.2 protocol fee share. The outer frame then transfers that 0.2 to the caller. A caller can therefore wrap a swap and remove either form of router revenue created during the transaction without sweeper authority.

— RECOMMENDATION

Apply one transient reentrancy lock to both public swap paths and keep it active through fee, surplus and receiver settlement. This ensures that each measured output balance delta belongs to only one active swap.

— RESOLUTION

Status: Resolved.

INFORMATIONAL I-05

ACKNOWLEDGED

Native referral beneficiaries can veto swaps

CATEGORY	DoS
LOCATION	contracts/FlowRouter.sol:187-194, 227-245
REVIEW	Main Review

— DESCRIPTION

`FlowRouter` synchronously pushes referral proceeds to the configured beneficiary before it pays the swap receiver. For native output, `TokenHelper.universalTransfer()` performs an unrestricted call to the beneficiary and reverts the entire transaction if that call fails. This gives a beneficiary control over whether every native output swap using its referral code can settle, even though the beneficiary is otherwise only a payment recipient.

The dependency affects both referral payment mechanisms. A fixed fee code calls its beneficiary during `_swap()`, while a zero fee code with a positive surplus configuration calls its beneficiary during `_settlePositiveSurplus()`. A contract beneficiary that does not accept native value therefore disables the affected native routes. A beneficiary with toggleable receive logic can selectively veto them. Because the revert unwinds the complete transaction, the user's input transfer, venue execution and all settlement are rolled back.

The fixed fee path also calls the beneficiary when the computed beneficiary payment rounds to zero. With two wei of gross output and a one percent fee, the beneficiary share is zero while the user's output after fees is one wei and satisfies a minimum of one wei. A rejecting callback with zero value still reverts that otherwise valid swap.

— RECOMMENDATION

Use a force safe native transfer for referral beneficiary payouts. A helper such as `SafeTransferLib.forceSafeTransferETH()` first attempts delivery with bounded gas and falls back to forced native value delivery when the beneficiary rejects the call. Continue using `SafeERC20` for token payouts. Apply the force safe path only to referral beneficiary payments unless the receiver settlement semantics are also intended to guarantee forced delivery.

Skip the fixed fee beneficiary call when its computed amount is zero. This avoids invoking beneficiary code when no payment is due.

— RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-06

ACKNOWLEDGED

Transient output custody retains token rewards

CATEGORY	Rewards
LOCATION	contracts/FlowRouter.sol:179-185,212-213; contracts/TokenFlowExecutorV2.sol:364-375,979-985
REVIEW	Main Review

DESCRIPTION

`FlowRouter` snapshots only the final `outputToken` balance before calling the executor. It later measures that token's increase and transfers the settled amount to `outputReceiver`. For supported terminal operations, the executor can direct output to `FlowRouter` before the outer settlement completes.

Some output tokens execute external reward accounting while transferring. A deployed dividend token observes `FlowRouter` as the temporary output holder and pays a reward in a different token before `FlowRouter` forwards the complete final output. Because `_swap()` accounts only for `outputToken`, this reward remains in the router and becomes withdrawable by an approved sweeper.

Source pool ID `0xbbbcfb87db0b05ba937c2e386a56824c7f6ff8c14974d5bc14906777d9c408e2` converts native currency into bridge and reward token `0xfB5b5778d45AE47F15323fb59B666c655174A79C`. Target pool ID `0x6d01881cc8c66cf6c301c5b74f4ced290e79d0c06cab026481e4163a455d6777` converts that token into output token `0x7B237dc3e88DDc4B7BF71c40c5d767708EE24Ef0` through hook `0xeE3de28732F24d15fB188139AE1471B2B6A940cC`.

The route consumes `53,081,966,819` wei and delivers `1,152,255,084,428,067,091,171` output units to the designated receiver. During settlement, the dividend mechanism pays `3,386,327,335` bridge-token units to `FlowRouter`. The input bound, output minimum, and received output remain correct.

This treatment is consistent with the documented policy that incidental residue is protocol inventory. The Informational impact is that integrators may otherwise assume rewards generated by a user's route follow its output, while these side rewards instead accrue to protocol custody.

RECOMMENDATION

Document that rewards in assets other than `outputToken` which are paid to `FlowRouter` during execution belong to the protocol. Route pricing and user interfaces should exclude

I-06 Transient output custody retains token rewards

CONTINUED

these side rewards from user proceeds, while protocol accounting should identify them separately from ordinary positive surplus.

— RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-07

ACKNOWLEDGED

Metric OMM swaps cannot supply extension data

CATEGORY Compatibility

LOCATION contracts/TokenFlowExecutorV2.sol:211-215,1404-1432

REVIEW Main Review

DESCRIPTION

Metric OMM's six argument `swap()` separates callback data from extension data. The pool forwards the sixth argument to its configured `beforeSwap` and `afterSwap` extensions and Metric's reference router defines it as opaque data forwarded to the pool. Its multiple hop form carries one extension payload per pool. Metric pools may be created with up to seven arbitrary nonzero extension addresses. The factory validates their uniqueness and call order but does not restrict them to implementations that accept empty data.

The executor cannot represent this protocol input. `MetricOmmPoolData` includes only the pool, direction and price limit. `_swapMetricOmmPoolExactInput()` reconstructs the pool call with literal empty byte strings for both callback data and extension data. Appending bytes to `Operation.data` does not help because the executor decodes only the fixed tuple and still supplies `""` as the sixth call argument.

An otherwise valid Metric pool whose swap extension authenticates an opaque payload therefore cannot execute through operation 32, even when the route compiler possesses the correct payload. A complete stack proof appends a valid authorization payload to the operation data. A direct Metric compatible call forwards that payload and swaps successfully while the same `FlowRouter` route reaches the pool with the empty data hash and reverts with `RequiredExtensionData`. The complete transaction rolls back, so the user loses transaction cost and cannot use the affected pool through this executor.

We did not identify a currently deployed Robinhood pool that requires nonempty extension data. The four published Metric extensions reviewed during validation do not read this argument. The proof therefore establishes that operation 32 is incompatible with valid Metric pools that use a custom or future extension which consumes this data. It does not establish that any currently deployed pool is unusable.

RECOMMENDATION

I-07 Metric OMM swaps cannot supply extension data

CONTINUED

Add a new Metric OMM opcode or route version whose pool data includes `bytes extensionData` and pass that value unchanged as the sixth `swap()` argument. Keep operation `32` as an explicitly empty extension data variant so existing version 2 encoders retain their tuple meaning.

Callback data can remain empty because the executor authenticates the active pool and stores its payment context transiently. Preserve the existing pool sender check, input cap, delta sign validation, direct final recipient accounting and residue handling in the new path.

— RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-08

ACKNOWLEDGED

Generated executor address is stale

CATEGORY	Configuration
LOCATION	script/generated/RobinhoodDeployment.sol:10
REVIEW	Main Review

DESCRIPTION

The generated Robinhood deployment projection is intended to identify the executor for the reviewed release. Its `ACTIVE_EXECUTOR` constant still points to `0x23dF315E5bEFE989Bd9259BE66557E580eaB14a7`.

The current Kipseli adapter at `0xc0010CFEce8bbD3D7c964099a12f1C4159816161` authorizes only `0x9a4212cf08B59a2C4B3d383Df67Be6bA43126E0f`. The latter runtime exactly matches the executor compiled from commit `3b6ce6b`. The generated address has a different runtime, is no longer authorized by the adapter and does not expose the release's `BEBOP_ROUTER()` integration. Both executors accept TokenFlow version 2, while the version 1 behavior belongs only to the retired executor at `0x8e9A5e1195095F2baDDD4911993e1B0a787f5312`.

A route service, deployment process or monitor that consumes `ACTIVE_EXECUTOR` can therefore select an obsolete feature set. Kipseli calls through that address revert because the adapter rejects it and release checks can incorrectly assess Bebop support. No onchain asset exposure was observed because the current authorized executor has no standing venue allowances or LiquidCore admission at the pinned block.

RECOMMENDATION

Regenerate `RobinhoodDeployment.sol` so `ACTIVE_EXECUTOR` identifies the intended release executor. Make the release pipeline reject a projection unless its address, runtime code hash, accepted TokenFlow version, immutable integration addresses and adapter authorization all match the intended deployment.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-09

ACKNOWLEDGED

Rialto requires optional token getters

CATEGORY	Compatibility
LOCATION	contracts/TokenFlowExecutorV2.sol:1709-1710
REVIEW	Main Review

DESCRIPTION

Rialto's Prop AMM guide at commit `b5c3347314a3b124c0336af405d21ac42777fac7` defines `getAmountOut` and `swapExactIn` as the required pool surface. It expressly makes `token0()` and `token1()` optional because Rialto obtains the pair and its direction from configuration.

`_ensureRialtoPropAmmAllowance()` first confirms that the active Rialto router allows the pool's `swapExactIn` selector. It then unconditionally calls both optional getters and compares their results with the route tokens. A pool that omits either getter therefore reverts during admission before it receives an allowance or a settlement call. This happens even when the pool implements both required selectors and the active Rialto router permits it.

A route compiler cannot avoid the failure by changing operation data because operation `170` has no alternate source for the configured token pair. Every FlowRouter swap that selects such a pool rolls back. The current deployed Fermi pool census contains getters, so the demonstrated impact applies when Rialto admits a pool that uses the smaller supported interface.

The local compatibility proof passed both tests against the unmodified executor. A direct quote and standard-token swap succeed through the required pool methods, while the executor route reverts at the missing token0 selector after a mocked successful allowlist response. No currently affected deployed pool was identified. This is submitted as an informational integration limitation, with no protected-fund loss or current production outage claimed.

Reference: [Rialto Prop AMM Integration Guide](https://github.com/rialto-org/rialto-api-docs/blob/b5c3347314a3b124c0336af405d21ac42777fac7/PROP_AMM_INTEGRATION.md#L26-L64).

RECOMMENDATION

Treat the token getters as an optional validation aid. Probe each getter with `staticcall` and compare any well formed address it returns. When both getters are available and

I-09 Rialto requires optional token getters

CONTINUED

match, retain the existing cached admission and persistent allowance path. When either getter is unavailable, do not cache the caller supplied pair. Confirm the active Rialto router's allowlist on every execution, approve only the resolved input amount, execute `swapExactIn` and reset the allowance to zero before residue accounting.

— RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-10

ACKNOWLEDGED

Native output to an executor reverts

CATEGORY Compatibility

LOCATION `contracts/TokenFlowExecutorV2.sol:343-345,1638-1646; contracts/FlowRouter.sol:179-185,212-213; contracts/TokenHelper.sol:31-36`

REVIEW Main Review

DESCRIPTION

`FlowRouter` accepts an address as `outputReceiver` and transfers the measured output to that address after the executor returns. An ERC-20 output can therefore settle to `TokenFlowExecutorV2` through an ordinary token transfer.

Native settlement follows a different path. `FlowRouter` calls the executor after `executePath()` has cleared its route lock. The executor's `receive()` function accepts native currency only from the wrapped native token or from selected venues while that lock is active. It does not accept `FlowRouter`, so the final native transfer fails. `TokenHelper.universalTransfer()` converts the failed call into `TokenHelper.InvalidNativeTransfer()` and the complete swap rolls back.

This does not cause asset loss or a persistent denial of service because the caller chooses the recipient and the revert is atomic. However, route construction cannot treat the executor as an interchangeable output destination across ERC-20 and native assets. A route that works with an ERC-20 output deterministically fails when its output is native currency.

RECOMMENDATION

Mark executor addresses as unsupported native output recipients in route construction policy and document this restriction next to the `outputReceiver` parameter.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-11

ACKNOWLEDGED

Fixed referral fees overflow on large outputs

CATEGORY Math
LOCATION contracts/FlowRouter.sol:187-198
REVIEW Main Review

DESCRIPTION

`FlowRouter` applies a registered fixed referral fee before it compares the measured output with `outputQuote`. The beneficiary share and the remaining user output both use multiplication before division. In particular, line 198 evaluates `amountOut * (FEE_DENOM - referralFee)` with checked `uint256` arithmetic even though the following division would produce a valid `uint256` result.

The interface does not limit `amountOut` to keep that intermediate product in range. With a one percent fee, a gross output of `116961706300319389316738368695644351366939378450141983878239` settles. Increasing the gross output by one raw unit overflows the net output multiplication. This value is below `type(int256).max`, so the overflow occurs before the separate signed slippage conversion.

Fixed fee swaps therefore support a smaller amount range than swaps without a fixed referral fee. An affected call reverts after executor output is delivered and the EVM rolls back every transfer. Users with an extreme raw token amount cannot settle the route and still pay its transaction cost.

RECOMMENDATION

Calculate both fixed fee amounts with full precision multiplication and division. Use `Math.mulDiv(amountOut, uint256(referralFee) * 8, FEE_DENOM * 10)` for the beneficiary share and `Math.mulDiv(amountOut, FEE_DENOM - referralFee, FEE_DENOM)` for the remaining user output. This preserves the current floor rounding and removes the overflowing intermediate products.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-12

ACKNOWLEDGED

Signed slippage casts reject large amounts

CATEGORY Math
LOCATION `contracts/FlowRouter.sol:202-209`
REVIEW Main Review

DESCRIPTION

`swapTokenInfo` represents `outputQuote` and `outputMin` as `uint256`. `_swap` also measures `amountOut` as a `uint256`. The settlement logic nevertheless converts both `amountOut` and `outputQuote` with `SafeCast.toInt256()` to calculate the signed `slippage` value. Either conversion reverts when its operand is at least `2255`.

This signed representation places an undocumented bound on otherwise valid unsigned swap amounts. A no fee swap that receives and quotes exactly `type(int256).max` settles normally. Increasing both values by one makes the same path revert after the executor has delivered the output token. The quote conversion can also trigger the failure independently. For example, an output of one unit with `outputMin` set to one still reverts when `outputQuote` is `2255`, even though the mathematical slippage of `1 - 2255` is representable by `int256`.

All state changes roll back, so the failure does not lose the transferred tokens. It prevents `swap` and `swapPermit2` from settling ordinary ERC20 routes that use the upper half of the advertised amount range. The practical impact is limited to tokens or quotes with extreme raw values, but affected users pay gas for a route that completes execution before the router rejects its telemetry calculation.

RECOMMENDATION

Keep settlement accounting unsigned. Compare `amountOut` with `outputQuote` directly and calculate the surplus or shortfall magnitude only in the selected branch. Replace the signed slippage field with a versioned event that records a sign flag and a `uint256` magnitude so event encoding does not restrict the amounts that the swap functions can settle.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-13

ACKNOWLEDGED

Pons refunds are retained by FlowRouter

CATEGORY Business logic errors

LOCATION `contracts/FlowRouter.sol:179-213; contracts/TokenFlowExecutorV2.sol:388-392,1321-1324,1543-1560`

REVIEW Main Review

DESCRIPTION

`FlowRouter` delivers the caller's full `inputAmount` to the executor before executing a route. When the supplied input would buy more launch tokens than remain available for sale, the Pons curve limits the purchase to the remaining amount. It recalculates the input required for those tokens and returns the excess to its caller, which is `TokenFlowExecutorV2`.

For ERC-20 input, `_finishPulledInput()` measures the lower actual spend and `_returnInputResidue()` transfers the balance above the route floor to the immutable `router`. For native input, the curve returns excess wei to the executor during the call and the program finalizer transfers the balance above `nativeBalanceFloor` to the router. `FlowRouter._swap()` only measures and transfers the output token. It does not account for input assets returned by the executor or forward them to the input payer.

Consequently, a successful partial Pons buy charges the caller the full supplied input while the unused portion becomes `FlowRouter` inventory that only an approved sweeper can withdraw through the intended interface. The caller receives the clamped launch token output, so the output minimum can still pass. The caller therefore loses the difference between the supplied input and the amount Pons actually charged. This loss is limited only by the route's supplied input and can exceed negligible rounding or dust amounts.

RECOMMENDATION

Track the input asset balance around the executor call in `FlowRouter` and return any positive route scoped increase to `msg.sender` after successful execution. For native input, calculate the opening router balance without the current `msg.value` so that only wei returned by the executor is refunded. Preserve the executor's existing token and native floors so that balances from earlier routes cannot be included in the refund.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-14

ACKNOWLEDGED

V4 native settlement inherits stale sync

CATEGORY	DoS
LOCATION	contracts/TokenFlowExecutorV2.sol:867-900,906-978
REVIEW	Main Review

— DESCRIPTION

The Uniswap V4 `PoolManager` uses one transient synced currency context for settlement. Calling `sync(token)` selects an ERC-20 and snapshots its reserves. Calling `sync(address(0))` restores the native currency context. If `settle()` receives native value while a nonzero currency remains selected, the manager reverts with `NonzeroNativeValue()` rather than crediting the native debt.

Both native debt paths in `TokenFlowExecutorV2` assume that this context is already native. `_uniswapV4RunSingle()` calls `manager.settle{value: job.amountIn}()` directly and `_uniswapV4RunBatch()` does the same when the batch's negative net currency is native. Neither path first calls `manager.sync(address(0))`.

A hook executes inside the same `PoolManager.unlock()` and can call the public `sync()` function. If the selected hook leaves an ERC-20 synced after returning its valid swap deltas, the executor's swap and all amount/minimum checks pass, but the subsequent native payment reverts exactly with `NonzeroNativeValue()`. The same context dependency affects a single native input swap and a native input batch with a negative native net.

No affected deployed hook was found despite extensive hook census testing. This is a transaction local availability failure. The revert restores the caller's input, manager deltas, hook state and token movements, but the user pays gas and cannot execute the otherwise valid native route through the executor. Although route construction chooses the pool and hook, the executor exposes hooked native V4 swaps as a supported operation and can establish the required native context independently.

— RECOMMENDATION

Call `manager.sync(address(0))` immediately before every native `settle{value: amount}()` in `_uniswapV4RunSingle()` and `_uniswapV4RunBatch()`. Keep the existing ERC-20 `sync` then transfer then `settle` sequence unchanged.

I-14 V4 native settlement inherits stale sync

CONTINUED

— RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-15

ACKNOWLEDGED

Algebra swaps cannot supply plugin data

CATEGORY	Compatibility
LOCATION	contracts/TokenFlowExecutorV2.sol:159-163,640-675
REVIEW	Main Review

— DESCRIPTION

Algebra Integral forwards the swap's data argument to its configured plugin hooks. Plugins can use this input to verify an authorization or apply caller-specific swap behavior.

The executor's `V3PoolData` contains only the pool, direction, and price limit. Algebra operation `101` uses the common V3 execution path, which always passes an empty byte string to the pool. Appending a payload to `Operation.data` does not help because the executor decodes the fixed tuple and reconstructs the call with empty data.

Consequently, plugin features that depend on caller-supplied data are unavailable through operation `101`, even when the route compiler possesses the required payload. If a plugin requires that payload for execution, the complete swap reverts and an included transaction costs the submitter gas.

— RECOMMENDATION

Acknowledge that operation `101` cannot supply plugin payloads and document this restriction for route construction. No onchain change is necessary if payload-dependent plugin features are outside the intended integration scope.

— RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-16

ACKNOWLEDGED

Graduation and escrow accounting are incompatible with balance-changing quote tokens

CATEGORY	fee-on-transfer accounting
LOCATION	packages/onchain-robinhood-type4/src/__onchain/ponsv2launchfactory-7e1eabd52ae29598/src/src/v2/PonsV2LaunchFactory.sol:_sweepCurve():L963-L977
REVIEW	Main Review

DESCRIPTION

Launch graduation measures the quote balance delta and records that amount as the launch's swept claim, but does not ensure the resulting claim remains seedable after a non-standard transfer. During pool creation, transfers are validated only by the recipient's credited delta and not by the factory's own debit; balance-changing tokens can therefore leave a launch permanently unseedable or consume balances backing other launches that share the asset.

Impact: For approved fee-on-transfer, rebasing, or surcharge tokens, a permissionless graduation/pool-creation call can burn or mis-account reserves, strand that launch, and in shared-balance cases deplete escrow attributable to another launch. This is an incompatibility with non-standard balance-changing ERC-20s; severity is limited to Low under the protocol's unsupported-token assumptions.

RECOMMENDATION

Reject or explicitly quarantine balance-changing assets, or perform end-to-end debit/credit measurements and rerun seedability checks before committing graduation state. Isolate per-launch escrow so one launch's transfer cannot debit another launch's recorded claim.

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-FlowRouter-Audit---Round-1-01a07f4d5ff770c6bfe5dd9f81f76daf/blob/refinery/poc-01a08026-6e11-7137-8b5b-eb0e2363b1ef/test/refinery-poc/graduation-and-escrow-accounting-are-incompatible-with-balance-changing-quote-tokens-dee.sol>

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-17

ACKNOWLEDGED

Callback ERC20 can capture in-flight V4 payments or poison settlement via unrestricted sync

CATEGORY	reentrancy
LOCATION	packages/mirror-GammaStrategies-FlowRouter-01a03991/contracts/TokenFlowExecutorV2.sol:_uniswapV4RunSingle():L894-L903
REVIEW	Main Review

DESCRIPTION

During ERC20 V4 settlement the executor calls sync, transfers the token externally, and then settles. A callback-capable token can overwrite the synced state and poison settlement, or—if an externally sourced payment arrives—call settleFor to claim the combined credit, take it, resync, and return only the executor's nominal amount. The fork PoC shows the latter clears NonzeroDeltaCount so the outer FlowRouter swap succeeds while the intervening donation is diverted.

Impact: This is a materially distinct accounting outcome from the prior revert-only variant, but callback-capable token support and a production reward source are not established. Therefore severity remains Low: affected callback-token routes may revert, and any triggerable third-party payment during the window could be captured; ordinary ERC20 routes are unaffected and no live production theft is demonstrated.

RECOMMENDATION

Treat callback-capable ERC20 inputs as unsupported or use pre-funded claims. Alternatively bind PoolManager sync snapshots to the authenticated locker/payer and reject settle/settleFor or resync by a different callback caller while the snapshot is active.

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-FlowRouter-Audit----Round-1-01a07f4d5ff770c6bfe5dd9f81f76daf/blob/refinery/poc-01a08026-6e11-7137-8b5b-eb0e2363b1ef/test/refinery-poc/callback-erc20-can-capture-in-flight-v4-payments-or-poison-s-3058d1d9.sol>

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-18

ACKNOWLEDGED

Positive quote-token balance growth enables zero-cost bonding-curve buys

CATEGORY	cross-flow accounting / rebasing token
LOCATION	packages/onchain-robinhood-type4/src/__onchain/ponsv2launchfactory-7e1eabd52ae29598/src/src/v2/PonsV2BondingCurve.sol:_receiveQuote():L551-L562
REVIEW	Main Review

DESCRIPTION

Recipient-side balance deltas from quote transferFrom are treated as delivered payment without an upper bound. Positive reflection/rebase growth, including growth triggered by transferFrom(0), is credited as quoteIn; bonding-curve buy pricing then dispenses launch tokens against phantom payment, while FlowRouter can make excess reflected growth spendable as route input.

Impact: With a nonstandard balance-changing quote asset, a caller can obtain launch-token supply for zero or tiny quote debit, repeatedly consuming sellable reserves and distorting graduation accounting; analogous FlowRouter paths over-credit route budgets beyond Permit2 authorization. Impact is limited to unsupported balance-changing tokens and is Low severity.

RECOMMENDATION

Reject or quarantine positive-rebasing/reflection assets, or cap observed recipient deltas at requested/signed amounts and separately account excess. Enforce deliveredInputAmount <= signed Permit2 amount before applying route budgets.

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-FlowRouter-Audit----Round-1-01a07f4d5ff770c6bfe5dd9f81f76daf/blob/refinery/poc-01a08026-6e11-7137-8b5b-eb0e2363b1ef/test/refinery-poc/positive-quote-token-balance-growth-enables-zero-cost-bondin-4532a15b.sol>

RESOLUTION

Status: Acknowledged.

§ 05

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.

§ 06

About Guardian.

Guardian is a smart contract security firm. Every engagement is worked by two independent teams competing on the same scope, each with its own tooling and methods, so no single reviewer's blind spot survives.

\$45B+DIGITAL ASSETS UNDER
REVIEW**200+**

ENGAGEMENTS COMPLETED

500+CRITICAL & HIGH FINDINGS
REPORTED

Reach us at audits@guardianaudits.com.