BAIL security

Camelot Router

FINAL REPORT

August ‘2025

Disclaimer:

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

The content of this assessment is not an investment. The information provided in this report is for general informational purposes only and is not intended as investment, legal, financial, regulatory, or tax advice. The report is based on a limited review of the materials and documentation provided at the time of the audit, and the audit results may not be complete or identify all possible vulnerabilities or issues. The audit is provided on an "as-is," "where-is," and "as-available" basis, and the use of blockchain technology is subject to unknown risks and flaws.

The audit does not constitute an endorsement of any particular project or team, and we make no warranties, expressed or implied, regarding the accuracy, reliability, completeness, or availability of the report, its content, or any associated services or products. We disclaim all warranties, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We assume no responsibility for any product or service advertised or offered by a third party through the report, any open-source or third-party software, code, libraries, materials, or information linked to, called by, referenced by, or accessible through the report, its content, and the related services and products. We will not be liable for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract.

The contract owner is responsible for making their own decisions based on the audit report and should seek additional professional advice if needed. The audit firm or individual assumes no liability for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract. The contract owner agrees to indemnify and hold harmless the audit firm or individual from any and all claims, damages, expenses, or liabilities arising from the use or reliance on the audit report or the smart contract.

By engaging in a smart contract audit, the contract owner acknowledges and agrees to the terms of this disclaimer.

1. Project Details

Important:
Please ensure that the deployed contract matches the source-code of the last commit hash.

Project	Camelot - Router
Website	app.camelot.exchange
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/CamelotLabs/exchange-contracts/commit/05737644e0167e288fcef00c75659cccfe2e41b6
Resolution 1	https://github.com/CamelotLabs/exchange-contracts/tree/db47ce7fab24b72ab193b2820aaf9edd1d2c7285
Resolution 2	https://github.com/CamelotLabs/exchange-contracts/tree/69bea558b2b628f2dd43c10e03334edaebe09032
Resolution 3	https://github.com/CamelotLabs/exchange-contracts/tree/1afdd402ef64c754acfbe665a509a08ed037bdac
Resolution 4	https://github.com/CamelotLabs/exchange-contracts/tree/12da3f10ac30a45ce6df9caf316728ebffcd19f7

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)	Failed resolution	Open
High	7	7				
Medium	6	1		5		
Low	10	6		4		
Informational	9	9				
Governance						
Total	32	23		9		

2.1 Detection Definitions

Severity	Description
High	The problem poses a significant threat to the confidentiality of a considerable number of users' sensitive data. It also has the potential to cause severe damage to the client's reputation or result in substantial financial losses for both the client and the affected users.
Medium	While medium level vulnerabilities may not be easy to exploit, they can still have a major impact on the execution of a smart contract. For instance, they may allow public access to critical functions, which could lead to serious consequences.
Low	Poses a very low-level risk to the project or users. Nevertheless the issue should be fixed immediately
Informational	Effects are small and do not post an immediate danger to the project or users
Governance	Governance privileges which can directly result in a loss of funds or other potential undesired behavior

Security Advice

At **BailSec**, our objective extends beyond the identification of vulnerabilities, we aim to deliver **high-confidence audits** that withstand post-audit change cycles.
Historically, one of the most significant residual risk vectors arises **during resolution rounds**, when architectural or behavioral changes are introduced to implement fixes.

To mitigate this, we provide **targeted security advice** intended to minimize the risk of regressions or new vulnerabilities emerging during final remediation.
These recommendations outline what we consider **necessary to finalize the audit with sustained confidence**, ensuring that post-resolution modifications do not invalidate the high-confidence assessment established by this report.

Change introduction

An additional resolution round (3) has been requested for minor changes, during the course of this review, our team identified two new low-severity issues. We highly recommend to be mindful with any further changes as this would essentially result in resolution round 4 which means there have been changes on changes on changes **on changes.** With every new iteration, the audit confidence is exponentially decreased and the risk of unidentified vulnerabilities increases exponentially.

We recommend to carefully review these new issues and in the scenario where these can be accepted, no further changes should be made to the code.

3. Detection

YakAdapter

The YakAdapter contract is a base contract used to implement certain functionality for all the adapters. It inherits the Maintainable contract, which implements role-based access control specifically for the Maintainer role. The contract allows the Maintainer role to recover ETH and tokens stuck in the various adapter contracts. The top-level query and swap functions are also implemented here, which call the internal functions that are implemented in the individual adapter contracts.

Core Invariants:

INV 1: swap destination always gets at least amountOut amount of tokens

Privileged Functions

- revokeAllowance
- recoverERC20
- recoverETH

Issue_01	setName function is unused
Severity	**Informational**
Description	The setName function is defined but never used. This can be invoked in the constructor to set the name, or removed from the contract.
Recommendations	Either use the setName function in the constructor or remove it from the contract.
Comments / Resolution	Resolved, the setName function has been removed.

VanillaV2Adapter

The VanillaV2Adapter contract is used to interact with Uniswap-v2-like pools. Like all adapters, it implements the swap and quote functions. The swap function swaps the input and output tokens by interacting directly with a pair contract, and the quote function estimates swap results used when building the swap path.

Core Invariants:

INV 1: Quotes are given out only when the pair exists in the factory
INV 2: swap must result in at least _amountOut tokens of output

Issue_02	Incompatible with the current deployed contracts
Severity	**Medium**
Description	The contract calls the getAmountIn function to estimate the amount of funds needed in an exactOut swap. The issue is that Camelot pair contracts do not implement this function. This would lead to reverted quotes. Vanilla Uniswap contracts do not implement either of the 2 quote functions getAmountIn and getAmountOut. Thus, the current codebase is incompatible with both Camelot as well as Uniswap pair contracts. The developers expressed that the Camelot pair contracts will be migrated, adding the missing getAmountIn functionality. However, any liquidity present in the current Camelot pair contracts will still remain out of reach of the router unless the pair contracts themselves are upgraded. The same issue is also valid for the CamelotAdapter contract.
Recommendations	Consider implementing a separate quoter contract to estimate the getAmountIn value. This way, liquidity from both the current versions as well as the soon-to-be-migrated version can be used by the router.

Comments / Resolution	Acknowledged. The CamelotAdapter contract implements a backup calculation method for calculating the amount input estimates in case the getAmountIn function is not present. However, this is not implemented for the VanillaV2Adapter contract.

CamelotAdapter

The CamelotAdapter contract is the adapter contract designed to specifically interact with Camelot pools, which implement directional fees and stableswap pools. The contract is almost identical to the VanillaV2Adapter contract, since both target pair contracts implement a similar interface.

Core Invariants:

INV 1: Quotes are given out only when the pair exists in the factory
INV 2: swap must result in at least _amountOut tokens of output

UniswapV3likeAdapter

The UniswapV3likeAdapter contract is the base contract for interacting with Algebra CLAMM pools. This contract is inherited by the AlgebraAdapter contract, which then interacts with the Uniswap v3-like pools. The contract here implements the quote and swap functions, as well as static calls to the quoter contract to calculate the swap results.

Core Invariants:

INV 1: swap must result in at least _amountOut tokens of output
INV 2: priceLimit for swaps always ensures that the entire liquidity is available for the swap

Privileged Functions

- setQuoter

Issue_03	Missing event emission in setQuoter
Severity	**Informational**
Description	The setQuoter function performs a storage edit but does not emit an event.
Recommendations	Consider emitting a QuoterSet event similar to what is done in the AlgebraV2Adapter.
Comments / Resolution	Resolved, QuoterUpdated event is now emitted on a quoter update.

Issue_04	Fee field is not populated
Severity	Informational
Description	The QParams struct contains a fee field, meant to contain the fee of the target pool. However, this field is never populated and thus remains 0. While all other fields are populated in the getQuoteForPool function, the fee remains 0. This issue is also present in the AlgebraV2Adapter contract.
Recommendations	Consider either removing the fee field or populating it with the current fee rate of the pool.
Comments / Resolution	Resolved, the fee field has been removed.

AlgebraAdapter

The AlgebraAdapter contract inherits the UniswapV2likeAdapter contract to interact with Algebra pools. This contract implements the algebraSwapCallback function to do the token payments and the _underlyingSwap function to interact with the actual pool.

Issue_05	Algebra adapter returns incorrect amount when fee-on-transfer tokens are involved
Severity	**High**
Description	The AlgebraAdapter contract uses the function swapSupportingFeeOnInputTokens from Algebra V1. The function returns the amount0 and amount1 variables. However, since the algebra pool holds fee on transfer tokens, when it transfers the output tokens to the AlgebraAdapter contract, the adapter receives slightly fewer tokens than intended. As per the implementation of the Algebra Pool contract in V1, the amount0 and amount1 variables returned do not reflect or take these fees into account. Thus, the returned value from the swap function can be erroneous. The main issue occurs in the _swap function of the UniswapV3likeAdapter contract. It then tries to transfer these tokens to the end user. *uint256 amountOut = _underlyingSwap(params, new bytes(0));* *if (amountOut < _amountOut) revert InsufficientOutputAmount(amountOut, _amountOut);* *_returnTo(_tokenOut, amountOut, _to);* This returnTo call will always fail, since it will be trying to transfer amountOut tokens, which will be higher than the actual amount of tokens available in the contract.
Recommendations	For Algebra V1 (and preferably for other adapters as well), consider checking the pre-balance/post-balance of the contract and returning the difference instead.

Comments / Resolution	Fixed by checking pre and post balances.

Issue_06	Missing protection on the callback function allows stealing of tokens from the adapter
Severity	**Low**
Description	The algebraSwapCallback function does not implement any checks on the caller. It is meant to be called only by the Algebra pool, but this isn't enforced anywhere. This allows any contract to call this contract and transfer out tokens from the Adapter. Since the Adapter isn't designed to hold tokens, this is classified as a low-severity issue. The same issue also applies to the AlgebraV2Adapter contract.
Recommendations	In the _underlyingSwap function, record the pool address in a global variable. Then, in the algebraSwapCallback function, ensure that msg.sender is the same pool address. Unset the pool address after the swap concludes.
Comments / Resolution	Fixed by checking if the caller is the target pool stored in the tempPoolAddress variable. Consider setting the tempPoolAddress to 0 after completion of the swap.

AlgebraV2Adapter

The AlgebraV2Adapter contract is the adapter to interact with Algebra V2 pools, which are based on Uniswap V4 pools, allowing custom logic and hook implementations. This implements functions similar to the UniswapV3likeAdapter contract, and also handles custom pools from whitelisted deployers.

Core Invariants:

INV 1: Only maintainer added custom pools are checked for quotes
INV 2: swap must result in at least _amountOut tokens of output

Privileged Functions

- addCustomDeployer
- removeCustomDeployer
- setQuoter

Issue_07	Custom pools aren't used in swaps, resulting in failed transactions
Severity	**High**
Description	The router has 2 main functions: findBestPath and swapNoSplit. The findBestPath function calculates a path, and the result of that can be used in the swapNoSplit function to carry out that swap. The findBestPath calls the query function on the adapter, while the swapNoSplit calls the swap function. For the system to work correctly, both the query and swap functions must operate on the same pool. However, in this contract, the query function checks queries from both normal pools as well as custom pools from whitelisted deployers. *for (uint256 i; i < deployersLength; ++i) {* *address customPool = getCustomPool(deployers[key][i], params.tokenIn, params.tokenOut);* *if (customPool != address(0)) {* *uint256 tempQuote = getQuoteForPool(customPool, params);* *quote = tempQuote > quote ? tempQuote : quote;* So the query function can return a path based on such a custom pool. However, the swap function ONLY operates on normal pools and not custom pools. *address pool = getBestPool(params.tokenIn, params.tokenOut);* *(bool zeroForOne, uint160 priceLimit) =* *getZeroOneAndSqrtPriceLimitX96(params.tokenIn, params.tokenOut);* *(int256 amount0, int256 amount1) = IAlgebraV2Pool(pool).swap(* *address(this), zeroForOne, int256(params.amount), priceLimit, callbackData* *);* So the query function generates a path based on the presence of

	custom pools, but this path cannot be used since the swap function actually doesn't support it. This will cause swaps to fail, since the normal pools might be giving a lower exchange rate than the custom pools, which is what the paths were based on.
Recommendations	Consider implementing a way to store the pool identifiers in the best path. This way, in the adapter, the correct pool can be chosen. This isn't an issue in the other adapters since each adapter only handles 1 pool for a given token pair. However, with Algebra V2, this assumption is broken, so the adapters must be able to distinguish between multiple available pools for each token pair.
Comments / Resolution	Fixed by returning deployer for algebraV2 custom pools. Additionally the findBestPath function returns the deployer addresses if applicable in the offer struct. _swapNoSplit has also been updated in order to support swaps with custom pools by using try-catch which attempts to call the algebra v2 adapter; if the call fails the default case is a regular swap attempt.

Issue_08	Bad comparison leads to inferior swaps
Severity	**High**
Description	In the getQuoteForBestPool function, the best quote is chosen among the available pool swaps. In algebra V2, since there can be multiple pools for a token pair, each allowed custom pair is also checked, and the best quote is chosen from amongst them. *for (uint256 i; i < deployersLength; ++i) {* *address customPool = getCustomPool(deployers[key][i], params.tokenIn, params.tokenOut);* *if (customPool != address(0)) {* *uint256 tempQuote = getQuoteForPool(customPool, params);* *quote = tempQuote > quote ? tempQuote : quote;* However, the issue is that the tempquote comparison is only valid for exactIn swaps. In the case of exactOut swaps, the lowest quote amount leads to the best swap, not the highest. This is correctly handled in the router, which compares quotes from different adapters, but not in the adapter itself, which compares quotes from multiple pools. Thus, for exactOut swaps, the Algebra V2 adapter is actually designed to quote the worst possible swap instead of the best.
Recommendations	Modify the tempQuote comparison to get the maximum or the minimum based on the exactIn value.
Comments / Resolution	Fixed by checking for the minimum quote for exact out swaps.

Issue_09	Custom pools cannot overwrite a failed quote from the main pool
Severity	**Low**
Description	The getQuoteForBestPool function checks quotes for the main pool as well as custom pools from other deployers. The issue is that if the main pool returns a quote of 0 due to exceeding the gas limit set in the staticcall, the custom pool queries will not be able to overwrite this for an exactOut swap. *if (bestPool != address(0)) quote = getQuoteForPool(bestPool, params);* In case the main pool returns a 0, the quote will be set to 0. *bool isBetter =* *params.exactIn ? tempQuote > quote : (tempQuote != 0 && tempQuote < quote);* Now, even if custom pools return a valid (non 0) tempQuote, the tempQuote<quote clause will fail, so this custom pool quote will never replace the failed main quote. So instead, a 0 quote will be returned, even though the custom pool was available.
Recommendations	A solution would be to allow tempQuote to overwrite quote if the quote is 0. Use the comparison already present in the _queryNoSplitWithDeployer function, comparing the quoteAmount with bestQuery for exactOut swaps. However, since that would result in a 4th resolution round with further code changes, we recommend refraining from making any more changes to the code, as every resolution round will decrease the overall audit confidence and increases the risk of vulnerabilities.
Comments / Resolution	Fixed following recommendations.

CamelotYakRouter

The CamelotYakRouter contract is the main router contract, which calculates paths and carries out swaps. It implements 2 main functions, findBestPath and swapNoSplit. The user is supposed to call findBestPath with their desired input and output tokens. The output of this function can be fed into the swapNoSplit function to carry out the calculated swap.

The contract also implements a fee on the swap, generating revenue for the protocol. A list of adapters is stored in the contract, which is iterated over when calculating the best swap path.

Appendix: findBestPath

The findBestPath function is used to calculate the best possible swap path starting from a specified tokenIn to a specified tokenOut. For any chosen token pair, the contract invokes the queryNoSplit function, which iterates over all the present adapters and calls their individual quote functions, and chooses the best swap out of them.

The contract supports both input and output-constrained swaps. It stores a list of trusted tokens and implements a recursive function, which recursively chooses trusted intermediate tokens until it hits the maximum allowed number of swaps.

The output of each possible swap route is compared, and the best is chosen from amongst them.

Appendix: swapNoSplit

The swapNoSplit carries out the calculated swaps. It takes in a Trade struct, which contains the amounts of input and output tokens, the token hop path, a list of adapters, and a list of recipients. The list of adapters is then iterated over, and each adapter is called to carry out a swap. The output of the swap is routed to the specified recipient, who will be handling the next swap. After the final swap, the output tokens are routed to the specified final recipient.

The contract also implements a few special functions like swapNoSplitFromETH and swapNoSplitToETH, to handle native eth-based inputs and outputs. The actual pools only support WETH, so the router contract handles the intermediate wrapping/unwrapping required to facilitate the swaps. It also supports permit-based tokens, allowing users to pass in permit signatures, skipping the approval transaction process.

Core Invariants:

INV 1: queryNoSplit chooses the best possible quoteAmount, the maximum for exactIn swaps, and the minimum for exactOut swaps
INV 2: findBestPath allows a max depth of 4 swaps
INV 3: Only user-specified and router-owner-specified trusted tokens can be intermediate tokens
INV 4: _findBestPath selects the maximum available quotes for exactIn swaps and the minimum available quotes for exactOut swaps
INV 5: swapNoSplit always ensures the final recipient receives at least amountOut amount of tokens.

Privileged Functions

- setAllowanceForWrapping
- setTrustedTokens
- setAdapters
- setMinFee
- setFeeClaimer

Issue_10	Baseline is never set when exactIn is false
Severity	**High**
Description	In the case where we call queryNoSplit with _exactIn == false, the baseline price will never be set if the first query fails. The result is that the baseline will remain 0. *if (* *i == 0 || _exactIn* *? quoteAmount > bestQuery.amount* *: quoteAmount < bestQuery.amount* *) bestQuery = Query(_adapter, _recipient, _tokenIn, _tokenOut, quoteAmount);* In the case where _exactIn == true, this is not an issue since a baseline of value 0 will get overwritten in the next iteration because any value > 0 will set the bestQuery. However, in the false case, there is no amount that can set the bestQuery because the baseline will be 0. The result will cause the query to incorrectly return 0 for bestAmount, and in the case where this would have been the bestPath, this will cause findBestPath to not return the best path in some cases.
Recommendations	Consider covering the case where the first iteration fails in the loop, and set the baseline even if we are not in the first iteration.
Comments / Resolution	Fixed by setting the bestQuery amount if it is 0 while the returned quote is non 0.

Issue_11	The same pool may be used twice, leading to incorrect prices
Severity	**High**
Description	The path-finding algorithm iterates over all possible paths with a certain maximum depth, and chooses the best one. This can sometimes lead to the same pool being operated on multiple times within the same transaction. Let's say the task is to find the best path for an A->C swap, and it happens to be A->B->C. Now, assume there are two A-B pools, with different prices leading to an arbitrage opportunity. Thus, the path A->B->A can actually be profitable when using the 2 different pools. So the most optimal route will be A->B->A->B->C. A-pool1->B-pool2->A-pool1->B-pool3->C In such a scenario, the pool1 gets utilized twice. The issue is that the quoter function is read-only, so it will simulate both the pool1 swaps, assuming the same constituent tokens. However, if we try to actually execute the swap, it will fail because the first pool1 swap will change the outcome of the second pool1 swap. Thus, the quoter will create paths with arbitrage loops that might not actually exist, since it treats each swap as an isolated event. However, if the same pool is used twice, the swaps are not isolated anymore and affect each other.
Recommendations	The same pool must not be used more than once in a path. Otherwise, the quote for the second usage will be incorrect.
Comments / Resolution	Fixed by storing the poolHash, which is a hash of adapter, tokens, and deployer. Before adding to the path, the logic now checks that the poolHash of the pool to be added does not already exist in the path, If so the logic continues and does not consider this as a potential hop.

Issue_12	Missing pools in an adapter can lead to rejection of valid paths
Severity	High
Description	The queryNoSplit function iterates over each adapter and checks the query value for a given token pair. If this value is better than the current best, it is saved as the best query. *try IAdapter(_adapter).query(_amount, _tokenIn, _tokenOut, _exactIn) returns (* *uint256 quoteAmount, address _recipient* *) {* *if (* *i == 0 || _exactIn* *? quoteAmount > bestQuery.amount* *: quoteAmount < bestQuery.amount* *) bestQuery = Query(_adapter, _recipient, _tokenIn, _tokenOut, quoteAmount);* The issue is that if any of the adapters do not have that tokenPair, it will return 0 instead of reverting. For exactOut swaps, a 0 quote is the best possible quote, so the bestQuery will save that 0 quote, which will cause issues in the path-finding algorithm. Say the Camelot adapter does not have an A-B pool, when query is called on it, it will enquire the factory, which returns address(0). *if (pair != address(0)) quoteAmount = IPair(pair).getAmountIn(_amount, _tokenOut);* The line above in the adapter contract will then refuse to populate the quoteAmount value, returning the default value of 0. This will be saved as the bestQuote for exactOut swaps, since 0 is the best possible quote for exactOut swaps. Then, in the path-finding loop, 0 quotes are rejected. *if (swapAmount == 0) continue;*

	So even though the A-B pair existed in Algebra V2, since it didn't exist in Camelot and the 0 quote slipped in, it will skip that whole path altogether. So even if A-B swap through Algebra V2 was the best possible path, since one of the adapters reported 0, the whole pair is ditched. This will lead to a lot of paths getting skipped if any adapter does not contain that token pair for exactOut swaps.
Recommendations	In the queryNoSplit function, consider rejecting quotes if the returned quote is 0. Also, consider reverting inside the adapters instead of returning default values in case of failure. Since the adapter calls are in try-catch blocks, there will be no impact on the router. Default returns can be misinterpreted in the system as legitimate values.
Comments / Resolution	Fixed by only updating the bestQuery if the quote amount is non zero, thus a failure which returns 0 will no longer cause issues for exactOut swaps.

Issue_13	Quoter failures can lead to rejection of valid paths
Severity	High
Description	The getQuoteSafe function in the AlgebraAdapter and AlgebraV2Adapter contracts is used to simulate swaps and generate quotes for the swap path. This is done via a static call to the uni-quoter contract. The results are then sent back to the quoter function. *(bool success, bytes memory data) = staticCallQuoterRaw(calldata_);* *int256 amount0;* *int256 amount1;* *if (success) (amount0, amount1) = abi.decode(data, (int256, int256));* *return (amount0, amount1);* The issue is that if the quoter ever fails, it will instead return the default value of 0. So the adapter will actually read a value of 0, even though the swap is not possible. In the queryNoSplit function, if an adapter returns a quote of 0, it is automatically treated as the best swap for an exactOut swap. Since no other quotes can beat this value, it will pass this value to the parent findBestPath function to build the final path. The findBestPath function in return checks the quotes from the queryNoSplit function, and rejects it if the quote is 0 by continuing the loop. So assume the function is trying to quote an A-B swap, through adapters Camelot, Algebra, and AlgebraV2. Say Camelot and Algebra give legitimate results, but AlgebraV2 returned a failed swap, and thus a value of 0. The queryNoSplit will then reject the correct quotes and only pass in the 0 quote since it is the best for an exactOut swap. The pathfinder will then drop the whole A-B swap route, since it's a 0 quote, completely rejecting the legitimate swap paths offered by Camelot and Algebra adapters.

	This is especially important in Algebra V2, since it can contain custom pools whose swaps can be frozen. Thus, if such a pool is frozen by its hooks and the quoter reverts, this will lead to rejected paths. Returning 0 as the default value is not a good choice when exactOut swaps are also involved.
Recommendations	In the queryNoSplit function, consider rejecting the quote if the returned quote is 0. Also, consider reverting inside the adapters instead of returning default values in case of failure. Since the adapter calls are in try-catch blocks, there will be no impact on the router. Default returns can be misinterpreted in the system as legitimate values.
Comments / Resolution	Fixed by only updating the bestQuery if the quote amount is non zero, thus a failure which returns 0 will no longer cause issues for exactOut swaps.

Issue_14	Incomplete swaps can lead to erroneous paths, leading to token losses
Severity	**Medium**
Description	In uniswap V3-like pools, there can be partial swaps if the pool runs out of liquidity. The pool continues iterating over the various ticks until it hits the max/min limit and then simply stops, swapping only that amount. The rest of the input tokens are not charged from the caller. If such a low liquidity v3 style pool is the only possible option, then the swap function can actually lose tokens. This is because the contract assumes that in any swap, the entire amount of input tokens is used. However, in such a partial-swap scenario, the unutilized input tokens will sit in the adapter contract, and will result in a loss for the caller.
Recommendations	Either refund unused input tokens after a swap, or revert in case there are any unused tokens, since it is not efficient and causes a loss to the caller.
Comments / Resolution	Fixed, unutilized input tokens are sent back to the router in order to allow for refunds to users.

Issue_15	A malicious user can bypass fees by using untrusted input tokens
Severity	**Medium**
Description	Currently, in function _swapNoSplit, there is no verification on the path of tokens provided by the user. Since the fees are taken on the input token in _swapNoSplit, the malicious user can: • Use an untrusted low-value token as the input token. An attacker can create a pool where 1 meme token can be swapped for 1000 USDC, for example. This will basically be funded by the attacker themselves. • MIN_FEE percent of 1 meme token is sent to the fee claimer • Code uses one of the adapters (example, Camelot or Uniswap pair) to swap it to USDC. • Path continues swapping as intended. • Attacker-controlled meme token can then burn the tokens held by the FEE_CLAIMER and mint them to themselves. Through this, the attacker is able to avail free swaps and cause loss to the team. While this is not a big issue, it is recommended to implement verification.
Recommendations	Either only allow input tokens to be one of the trusted tokens, or consider acknowledging the issue, accepting an occasional fee loss.
Comments / Resolution	Acknowledged. No aggregator fee will be used for now.

Issue_16	findBestPath function does not calculate router fee
Severity	Medium
Description	The findBestPath function of the router is used to calculate the best route for a swap. The output is then formatted in a way that it can be fed directly into the swapNoSplit function to carry out the actual swap. The issue is that the swapNoSplit function actually charges a fee, which is not taken into account in the findBestPath function. So the output cannot be directly plugged into the swap function, the fee amount also needs to be added on top. The output quote isn't exactly compatible with the swap input.
Recommendations	Consider implementing another quote function that calculates the best path, including the fee. This will allow contracts to call the swap with the exact output from the quote function.
Comments / Resolution	Acknowledged. The aggregator fee will not be turned on for now.

Issue_17	Path calculation not valid for fee-on-transfer tokens
Severity	**Medium**
Description	The protocol makes an effort to support fee-on-transfer tokens by always calculating amounts as the pre and post balance differences, and also using the swapSupportingFeeOnInputTokens function, which supports FOT tokens. However, the quote function actually does not support FOT tokens, since it does not take into account the fees incurred during the various token transfers. Thus, the quote generated by the findBestPath function will actually be incorrect if FOT tokens are involved, and will result in an amountOut estimate that will be higher than any possible actual swap, which will then result in a revert if the same amountOut value is passed in the swapNoSplit function. Thus, even though the swap function supports FOT tokens, the quote will always be wrong for such tokens.
Recommendations	If FOT token support is desired, the quote function must also calculate the fees paid for all the token transfers involved. Otherwise, it should be made clear that the protocol queries do not support FOT tokens.
Comments / Resolution	Acknowledged. Slippage setting is expected to take this into account. Certain whitelisted FOT tokens will have custom slippage settings.

Issue_18	Longer paths may be chosen over shorter paths of equal value
Severity	**Medium**
Description	The logic does not handle the case where a long path is found first, but a shorter path of equal value is found after. This is because the logic will only update the bestOption if the path in question is a better value, the length of the path is ignored. *bool isBetter = (bestAmount == 0)* *|| (_exactIn ? pathAmount > bestAmount : pathAmount < bestAmount);* *if (isBetter) {* *bestAmount = pathAmount;* *bestOption = newOffer;* This is a problem because longer paths will incur more gas costs, and thus the shorter path of equal value is actually the best path once we factor in the gas costs.
Recommendations	Consider checking the number of steps when a path is of equal value, and updating the bestOffer if a path of equal value has fewer steps.
Comments / Resolution	Acknowledged. Gas considerations have been removed from the aggregator.

Issue_19	Baseline can be incorrect if query from non-first adapter fails
Severity	**Low**
Description	In the queryNoSplit function, for the exactOut swap case, the quote amount is first checked before being set as the best quote. *(quoteAmount < bestQuery.amount) || (bestQuery.amount == 0 && quoteAmount != 0)* The _queryNoSplitWithDeployer function implements a similar thing, but with a slightly different setup. *quoteAmount != 0 && (bestQuery.amount == 0 || quoteAmount < bestQuery.amount)* The issue is that the first instance doesn't always work if any adapter in the middle returns 0. The adapters are configured with a gas limit, and if this gas limit is exceeded, or if a pool doesn't exist, it will return a 0. In the _queryNoSplitWithDeployer function, a 0 return is always rejected. But in the first code instance, this is not the case. If the query returns 100, 0, 200 in 3 adapters, the code will return 200 instead of 100. This is because in the first iteration, the bestQuery will be set to 100. Then, when a query fails and we get 0, 0<100 is true, so the bestQuery will be replaced by 0 instead of rejecting this value. Then, in the third iteration, it will be overwritten by 200, which is worse than the 100 quote.
Recommendations	A solution would be to use the second code instance linked above to replace the first code instance. However, since that would result in a 4th resolution round with further code changes, we recommend refraining from making any more changes to the code, as every resolution round will decrease the overall audit confidence and increases the risk of vulnerabilities.
Comments / Resolution	Fixed following recommendations.

Issue_20	Multiple user errors possible, may break implicit invariants and cause loss of funds
Severity	**Low**
Description	The _swapNoSplit function requires that the trade path contains one more element than the adapter array, which must be of the same size as the recipient's array. However, this check is never enforced, and thus, if users miss some of the elements of the array, it can lead to fund loss.
Recommendations	Validate that _trade.path.length - 1 == _trade.adapters.length. Also validate the length of the recipients array.
Comments / Resolution	Fixed following recommendations.

Issue_21	_swapNoSplit does not validate adapter whitelist prior to execution
Severity	**Low**
Description	Adapter whitelist is critical for system integrity, since each adapter has the ability to control and steal user funds as well as manipulate future hop `recipientBalanceBefore` values, all of which could cause loss of funds or swap denial of service. However, the _swapNoSplit trusts user input directly when constructing which adapter is being used.
Recommendations	Consider making sure the adapter provided by the user is actually supported/whitelisted by the router.
Comments / Resolution	Acknowledged. Aggregator is to be used by the dapp directly. External users are expected to pass in the correct paths.

Issue_22	Stranded eth can be stolen via swapNoSplitFromETH
Severity	**Low**
Description	The router inherits the recoverable contract, which allows the contract to recover any stranded native token or erc20 tokens. The problem occurs when swapNoSplitFromETH does not enforce that msg.value == trade.amountIn *function swapNoSplitFromETH(Trade calldata _trade, uint256 _fee, uint256 _deadline, address _to)* *external* *payable* *override* *withDeadline(_deadline)* *{* *if (_trade.path[0] != WNATIVE) revert PathDoesNotBeginWithWETH(_trade.path[0]);* *_wrap(_trade.amountIn);* *_swapNoSplit(_trade, address(this), _fee, _to);* *}* A user can specify any amountIn without sending a msg.value in order to use the stranded eth in the contract. Lost eth should not be able to be touched or used by anyone other than the maintainer, as indicated by the various onlyMaintainer recovery functions present in the contract.
Recommendations	Considering enforcing msg.value matches amountIn in swapNoSplitFromETH.
Comments / Resolution	Fixed by ensuring msg.value matches amountIn.

Issue_23	Permit functionality can revert
Severity	**Low**
Description	Functions swapNoSplitWithPermit and swapNoSplitToETHWithPermit allow users to provide parameters to use permits to create token approvals. In this case, the token would be the input tokenIn being swapped. However, once the permit signature is available in the mempool, anyone can frontrun the transaction and use up the permit signature to grant the approval. This will then cause the permit call to revert, reverting the transaction. The user will then need to call the swap function without permit, since the approval has now already been granted. While this will not block any operations, it can lead to bad UX for the user since their initial swap transaction failed.
Recommendations	Consider putting the permit call in a try-catch block. If the permit signature is already used up, it will proceed to do the swap anyway instead of reverting.
Comments / Resolution	Fixed by putting the permit calls in try-catch blocks.

Issue_24	Permit functionality is incompatible with DAI
Severity	**Low**
Description	Functions swapNoSplitWithPermit() and swapNoSplitToETHWithPermit() use the following permit signature: *function permit(* *address,* *address,* *uint256,* *uint256,* *uint8,* *bytes32,* *bytes32* *) external;* However, the DAI token (available in the Camelot DEX UI) uses the following permit signature: *function permit(address holder, address spender, uint256 nonce, uint256 expiry,* *bool allowed, uint8 v, bytes32 r, bytes32 s) external* Due to the incompatible signatures, it is currently not possible to use the permit functionality with the DAI token. Please note that this issue is only valid in case the blockchain is not Arbitrum.
Recommendations	Consider implementing a separate function to support permit for DAI in case the blockchain is not Arbitrium. Otherwise this issue only serves as an informational reminder.
Comments / Resolution	Acknowledged.

Issue_25	swapNoSplit is fundamentally an exactIn swap
Severity	**Low**
Description	While the findBestPath function supports finding paths for both exactIn and exactOut swaps, the actual swap function swapNoSplit is fundamentally an exactIn swap. While the swapNoSplit requires a path to be specified, it charges the user the exact amountIn amount. In every step of the swap, it utilizes all the available funds to swap to the next token. If the prices become more favorable than the passed in Trade route, the user is granted more tokens than tokenOut. This is fundamentally different from an exactOut swap, where if the prices become more favorable, the user gets charged less amountIn and receives the same amountOut. While in this contract, they get charged the same amountIn and get back more amountOut.
Recommendations	Consider implementing a similar swap function with exactOut swaps. This would, however, require getting quotes from each adapter again, in order to do the swaps.
Comments / Resolution	Acknowledged.

Issue_26	Path-finding algorithm can find profitable but gas-intensive loops
Severity	**Low**
Description	The recursion-based path-finding loop finds the most optimal route for swaps. This can lead to some unintuitive paths, since the amountOut is being maximised, not the gas or the path length. Say we want to swap A-C, allowing an intermediate trusted token B within a maximum step of 4. A normal router will quote the shortest, most efficient path, which is, say, an A-C direct swap. However, the recursive algorithm here will keep recursing to maximize amountOut. If there exists an A-B-A arbitrage loop utilizing two different A-B pools, it will also work that into this swap. So the most optimal swap can look like A->B->A->C Simply because the A-B-A loop gives more A tokens than the starting amount, which will lead to a larger C token amount. This can, however, lead to large gas costs, since that is not factored in anywhere in the calculations. So while the algorithm here optimizes for returns, it might actually be worse in terms of total cost (gas + return) than direct/simpler swaps.
Recommendations	Consider acknowledging this if such paths are acceptable and gas costs are not important. Otherwise, put a restriction in the recursion function such that any token can only appear once in the path. This will lead to shorter paths and no loops.
Comments / Resolution	Acknowledged. Output amount is prioritized over gas.

Issue_27	No sanity check in setMinFee
Severity	**Informational**
Description	The setMinFee function does not bound the input _fee value. Thus, any fee can be set by the maintainer, even higher than the FEE_DENOMINATOR value, leading to 100% fees.
Recommendations	Enforce a strict upper bound on setMinFee, which ensures that _fee <= FEE_DENOMINATOR at all times.
Comments / Resolution	Fixed by adding a maximum fee of 10%.

Issue_28	Unnecessary maintainable inheritance
Severity	**Informational**
Description	The CamelotYakRouter inherits Maintainable and Recoverable. However, Recoverable already inherits Maintainable. This creates a superfluous inheritance which C3 linearisation will opt to remove.
Recommendations	Only inherit a contract once.
Comments / Resolution	Fixed by removing the inheritance.

Issue_29	Bad event emission in setFeeClaimer
Severity	**Informational**
Description	The setFeeClaimer emits an event. *FEE_CLAIMER = _claimer;* *emit UpdatedFeeClaimer(FEE_CLAIMER, _claimer);* The issue is that when the event is being emitted, FEE_CLAIMER has already been overwritten. So both the fields of the event have the same value. This is in contrast to the other events, where it emits first the old value and then the new value.
Recommendations	Store the current FEE_CLAIMER in a variable and emit it instead.
Comments / Resolution	Fixed by following the recommendation.

Issue_30	Deadline is checked twice
Severity	**Informational**
Description	The modifier withDeadline ensures that the deadline has not passed. However, the modifier exists in the permit variants swapNoSplitWithPermit and swapNoSplitToETHWithPermit, these functions both have a subcall to a function that also includes the withDeadline modifier, essentially making the same check twice.
Recommendations	Consider removing the deadline check from the permit variants. Otherwise, consider acknowledging this issue to minimise code changes.
Comments / Resolution	Fixed. Removed deadline check in permit functions.

Issue_31	returnTokensTo uses transfer, which restricts gas forwarded
Severity	**Informational**
Description	The use of transfer when sending native tokens may cause problems with interacting contracts that have receive or fallback functions with extended logic. This is because transfer only sends 2300 gas for executing the logic, which can be insufficient. *function _returnTokensTo(address _token, uint256 _amount, address _to) internal {* *if (address(this) != _to) {* *if (_token == NATIVE) payable(_to).transfer(_amount);* *else IERC20(_token).safeTransfer(_to, _amount);*
Recommendations	Consider using call instead to send native.
Comments / Resolution	Fixed by using low-level calls to transfer eth.

Issue_32	Giving allowance for wrapping is not needed
Severity	Informational
Description	*if (_feeClaimer == address(0) || _wrapped_native == address(0)) revert AddressZero();* *setAllowanceForWrapping(_wrapped_native);* *setTrustedTokens(_trustedTokens);* *setFeeClaimer(_feeClaimer);* *setAdapters(_adapters);* *WNATIVE = _wrapped_native;* *}* The constructor will call setAllowanceForWrapping, which will give the weth contract allowance for wrapping. However the contract does not need allowance since the _wrap function will call deposit, which does not need approval.
Recommendations	Consider removing setAllowanceForWrapping.
Comments / Resolution	Fixed by removing allowance for wrapping.