Scanned Code Report

AUDITAGENT

Powered by NETHERMIND SECURITY

Code Info

Developer Scan

Scan ID
9

Date
February 25, 2026

Organization
RigoBlock

Repository
v3-contracts

Branch
fix/uni-wrap

Commit Hash
d4a871ec...42faaee5

Contracts in scope

contracts/protocol/extensions/adapters/AUniswapDecoder.sol

Code Statistics

Findings
0

Contracts Scanned
1

Lines of Code
437

Findings Summary

Total Findings

High Risk (0)

Medium Risk (0)

Low Risk (0)

Info (0)

Best Practices (1)

Code Summary

The `AUniswapDecoder` is an abstract contract designed to function as a sophisticated calldata decoder for the Uniswap ecosystem. Its primary role is to parse and interpret complex, often batched, transactions intended for the Uniswap Universal Router and the Uniswap V4 Position Manager. The contract provides a mechanism to understand the effects of a transaction before or during its execution by extracting key parameters.

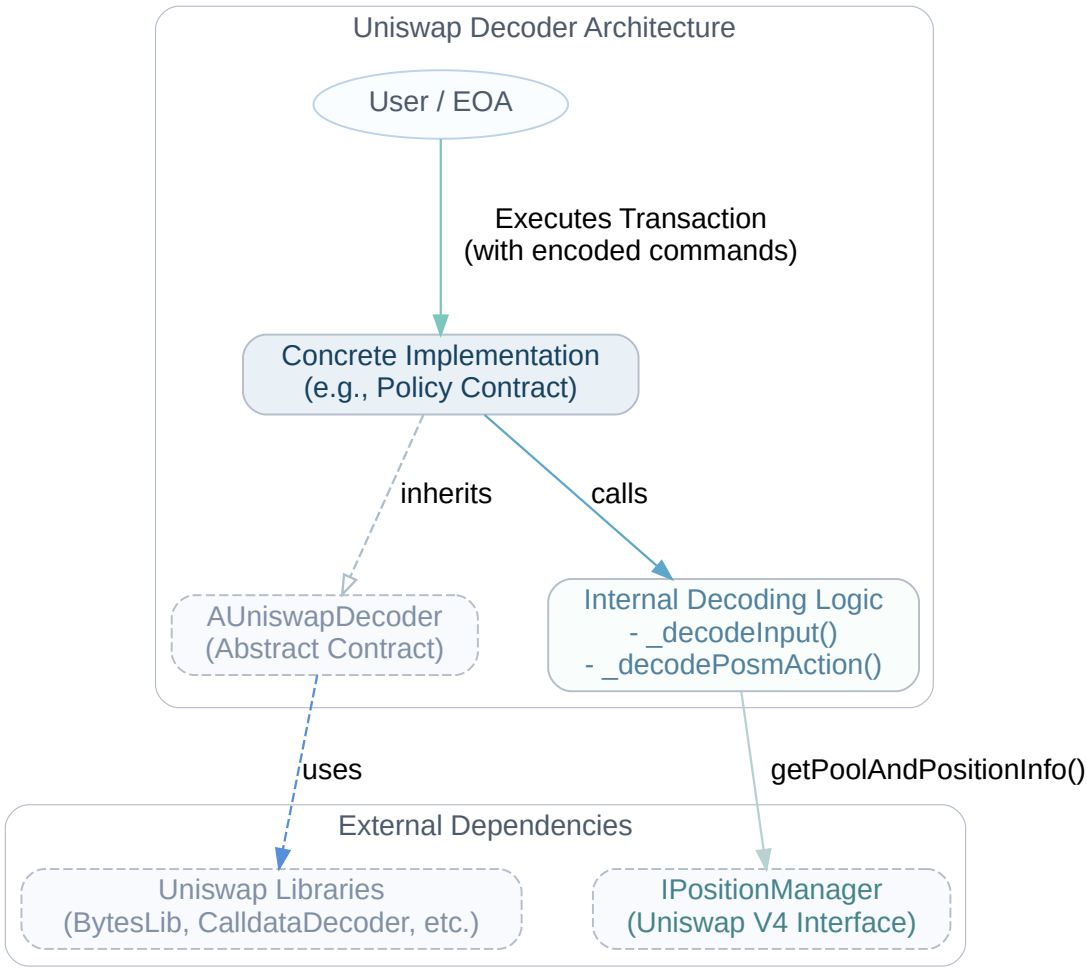
The contract's logic is split into two main internal decoding functions:
- One function is responsible for decoding commands sent to the Universal Router. It supports a wide range of commands across different Uniswap versions, including `V2_SWAP_EXACT_IN`, `V3_SWAP_EXACT_IN`, `V4_SWAP`, token transfers (`TRANSFER`, `SWEEP`), and native currency management (`WRAP_ETH`, `UNWRAP_WETH`). It can also recursively decode nested commands within a sub-plan.
- The second function focuses on decoding actions directed at the V4 Position Manager. This includes liquidity management operations such as minting new positions (`MINT_POSITION`), increasing (`INCREASE_LIQUIDITY`), decreasing (`DECREASE_LIQUIDITY`), and burning liquidity (`BURN_POSITION`), as well as various token settlement and withdrawal actions.

For any given transaction data, the decoder aggregates critical information into a structured format. This includes identifying all unique token recipients, the specific tokens being sent (tokensIn) and received (tokensOut), and the total native currency (e.g., ETH) value involved. It is intended to be inherited by other contracts, serving as a foundational utility for systems that require deep inspection of Uniswap interactions, such as smart contract wallets, aggregators, or security analysis tools.

Entry Points and Actors

As an abstract contract, `AUniswapDecoder` does not expose any public or external state-modifying functions. It is designed to be inherited by concrete implementations which would define the user-facing entry points.

Code Diagram



1 of 1 Findings

contracts/protocol/extensions/adapters/AUniswapDecoder.sol

`_decodePosmAction` Fails to Record Output Tokens for Liquidity Removal Actions

Best Practices

The `_decodePosmAction` function is responsible for parsing calldata for the Uniswap V4 Position Manager. For actions that result in the pool receiving tokens, such as `DECREASE_LIQUIDITY` and `BURN_POSITION`, the function correctly identifies the `tokenId` of the position being modified but fails to add the underlying tokens (`currency0` and `currency1` of the position's pool) to the `params.tokensOut` array.

The protocol's security model relies on the decoder to identify all tokens the pool will receive (`tokensOut`) so that an upstream adapter can verify they have a valid price feed in the oracle. By not populating `params.tokensOut` for these liquidity removal actions, the decoder provides incomplete information to the validator. A pool owner could be tricked into executing a transaction that removes liquidity from a pool containing a malicious or un-priced token. The pool would receive this token, but the price feed check would be bypassed, corrupting the pool's Net Asset Value (NAV). This could be exploited to manipulate share prices.

While the function makes a `view` call to `_uniV4Posm.getPoolAndPositionInfo` for `INCREASE_LIQUIDITY`, it omits this for `DECREASE_LIQUIDITY` and `BURN_POSITION`, likely as a gas-saving measure. However, this optimization compromises a critical security guarantee.

```
// in _decodePosmAction
...
            } else if (action == Actions.DECREASE_LIQUIDITY) {
                // uint256 tokenId, uint256 liquidity, uint128 amount0Min, uint128
amount1Min, bytes calldata hookData
                (uint256 tokenId, , , , ) = actionParams.decodeModifyLiquidityParams();
                // hook address is not relevant for decrease, use ZERO_ADDRESS instead to
save gas
                positions = _addUniquePosition(positions, Position(ZERO_ADDRESS, tokenId,
Actions.DECREASE_LIQUIDITY));
                return (params, positions); // <-- params.tokensOut is not updated
...
            } else if (action == Actions.BURN_POSITION) {
                // uint256 tokenId, uint128 amount0Min, uint128 amount1Min, bytes calldata
hookData
                (uint256 tokenId, , , ) = actionParams.decodeBurnParams();
                // hook address is not relevant for burn, use ZERO_ADDRESS instead to save
gas
                positions = _addUniquePosition(positions, Position(ZERO_ADDRESS, tokenId,
Actions.BURN_POSITION));
                return (params, positions); // <-- params.tokensOut is not updated
            }
...
```

Disclaimer

Kindly note, no guarantee is being given as to the accuracy and/or completeness of any of the outputs the AuditAgent may generate, including without limitation this Report. The results set out in this Report may not be complete nor inclusive of all vulnerabilities. The AuditAgent is provided on an 'as is' basis, without warranties or conditions of any kind, either express or implied, including without limitation as to the outputs of the code scan and the security of any smart contract verified using the AuditAgent.

Blockchain technology remains under development and is subject to unknown risks and flaws. This Report does not indicate the endorsement of any particular project or team, nor guarantee its security. Neither you nor any third party should rely on this Report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset.

To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this Report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, 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, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.