Otim Protocol Actions Review

I conducted a targeted three day review on several action contracts from the Otim Protocol's 7702 wallet infrastructure. In particular the actions reviewed in this audit were `SweepDepositAccountAction` , `SweepDepositAccountERC20Action` , and `UniswapV3ExactInputAction` . The actions in question enable sweeping funds into an otim account and setting a recurring trade order which invokes Uniswap v3. These actions are naturally limited in impact by the design of the Otim protocol which display good security architecture. The commit hash for the reviewed code was b1adc76.

Disclaimer: Security review is performed on a best effort basis and some bugs may remain.

About The Team

Aleph_v is a former mathematician and current cryptographer and security researcher with 7 years of experience preforming security reviews and finding critical bugs for clients across the blockchain and cryptography industries.

Contact: Twitter - https://twitter.com/alpeh_v (https://twitter.com/alpeh_v) or Telegram - aleph_v

Findings

Summary: Critical: 0, High: 1, Medium: 1, Low: 1

Findings are presented chronologically:

- Eth/ERC20 Sweep actions allow gas griefing when misconfigured: In the ETH swap action the action can only be triggered when a create2 deposit address has a balance which is greater than a threshold. In the case where the threshold is set to 0 or another very low value, a greifer can expend ETH to burn ETH or other fee tokens from a users wallet. They can do so by executing the sweep action every block till the user's account is out of the fee paying token which will have all been paid to the Otim treasury. In the case where the threshold is set to a non zero value the recipient of the sweep can trigger this gas griefing vector despite a high threshold by using a flashloan (or personal funds) to fund the address, calling claim, then repaying the flash loan. Since the griefed fees are paid to the Otim treasury the team may be able to repay them to the user, lowering the impact.
    - Severity Rating: Low [Likelihood: Low, Impact: Medium]

  - Remediation Recommendation: Enforce a set minimum threshold via a constant check on deployment.
  - Status: Acknowledged

- Unlimited priority fee configurations allows for otim to drain accounts or for gas griefing: In the fee module contract used by all actions, the fee module reverts if `(tx.gasprice - block.basefee > fee.maxPriorityFeePerGas && fee.maxPriorityFeePerGas != 0)` and so setting the `fee.maxPriorityFeePerGas` to 0 is an intentional configuration setting for an unlimited priority fee. However unlike the base gas fee there is no safe way to set an unlimited priority fee limit. Since the priority fee is submitter controlled it can always be set to be as high as allows the draining of the account of all fee tokens in one action. Block building services pick the blocks selected by the proposer based on the profit to the proposer and so if an Otim account pays all of its balance to high priority fees this means that the executor who includes that transaction in their proposed block can lower the extraction from other MEV to profit directly by the same amount. So therefore a user can be gas griefed by any MEV bot who pays only the base cost of transaction inclusion at cost of potentially many times more value destroyed than cost. Moreover if the MEV bot is run by the controller of the treasury contract they can use this to extract the whole balance of user accounts because they are reinbursed for the increased priority fee costs. Unlike other gas griefing exploits this can take place in even a single transaction.
  - Severity Rating: Medium [Likelihood: Low, Impact: High]
  - Remediation Recommendation: Enforce that the user must set a priority fee limit by removing the 0 case of the check. Optionally, add a priority fee constant based on historic data.
  - Status: Remediated: Feature flag for unlimited priority fee removed

- Very high susceptibility to MEV because of long standing DCA orders with the same `amountOutMinimum` : In the `UniswapV3ExactInputAction` action the user sets up an action which because of the way the Otim protocol works can potentially be executed multiple times on a schedule. In this action when approved the user sets all of the swap including a fixed amount of eth, or erc20 to swap in and a fixed `amountOutMinimum` . The `amountOutMinimum` parameter is normally used as a slippage control parameter and should ideally be within 1% of the assets price at swap time (depending on asset liquidity), but because the Otim system is designed to re-execute orders in a dollar cost averaging strategy, the minimum price of the swap quickly becomes out of date. This exposes the user to a MEV loss equal to the total change in price of the asset since the creation of the order, potentially equal to a very high percent of the users value because of the volatility of the cryptocurrency markets. Even for relatively small swaps in the case of large divergences MEV bots will still profit from forcing the user to take large losses as a percent of assets as the cost of the MEV bot is only the fees taken when displacing the uniswap v3 price. Consider for instance a DCA user making once weekly buys for a month for a total of 10k, in an example recent period to writing ETH went from roughly 2,000$ to roughly 1,400$ in a period of a month, meaning the final swap would expose the user to 750 dollars of MEV extractable losses in this case. In the contrapositive this fixed `amountOutMinimum` also causes quite a large percentage of swaps, which are intended to regularly execute, to fail as prices move against them.

  - Severity Rating: High [Likelihood: Medium, Impact: High]
  - Remediation Recommendation: The `amountOutMinimum` should be set based on a trusted oracle price with a reasonable level of slippage which is user indicated.
  - Status: Remediated - amount out minimum for the Uniswap swap is now set via a max deviance from a user set time period average.

Notes

Very low severity, performance notes, or other non-issue comments

- If the submitter uses an access control list on any of the state variables or contract addresses inside of the metered area of the action, the user will not be charged for the gas usage appropriate to the execution because the cost of warming storage slots is assessed before the metered block begins.
- The code uses you `execute(bytes calldata commands, bytes[] calldata inputs, uint256 deadline)` on the universal router with deadline = block.timestamp, which has the same behavior as just doing `execute(bytes calldata commands, bytes[] calldata inputs)`