Tycho Router

Security Assessment

Prepared for **Tycho** by **Maximilian Krüger** (snd.github.io)

2025-03-07

Disclaimer

This security assessment was timeboxed to 3.5 person-days.

Further time investment might result in additional findings.

The findings must not be considered an exhaustive list of security issues in the codebase.

Table of Contents

Summary

PropellerHeads engaged Maximilian Krüger to assess the security of the `TychoRouter` contract over 3.5 person-days between 2025-02-18 and 2025-02-28. The code in scope consists of around 700 lines.

The assessment resulted in 7 findings with the following severities:

Severity	Count
High	3
Medium	1
Low	2
Undetermined	1

The most severe is Finding 4, which enabled anyone to freeze Tycho Router's `swap` function.

PropellerHeads accepted the risk of Finding 3 and Finding 7. All other findings were fixed.

> ## Recommendations for Users of Tycho Router
>
> If you're planning to use Tycho Router, **always** set the `minAmountOut` parameter of the `swap` function to the smallest amount you accept to receive.
>
> If you set `minAmountOut` to `0`, it's possible that you'll receive `0`, due to bad swap sequences or slippage. See Finding 6 for more information.
>
> For example, if you expect to receive `1000` USDC and accept 5% slippage, set `minAmountOut` to `950 * 10**6`.
>
> Avoid infinite Permit2. Set both Permit2 allowance and deadline as low as possible.
>
> Before interacting with Tycho Router, check the `ExecutorSet(address executor)` events emitted by the contract. Identify executors that were not included in this assessment. New executors can alter the behavior and security of the system.
>
> Assets, which are accidentally sent to Tycho Router, should not be considered recoverable.

Scope

The assessment focused on commit `c940d8b5366f463116b64dc24ddde56e87decace` of the tycho-execution repository.

Only files in the `foundry/src` directory were in scope:

- `TychoRouter`
- `Dispatcher`
- `executors`
  - `BalancerV2Executor`
  - `UniswapV2Executor`
  - `UniswapV3Executor`
  - `UniswapV4Executor`

Target System Description

Tycho Router executes sequences of swaps that interact with various AMMs. The swap sequences are generated off-chain, encoded and sent to the `swap` function of the `TychoRouter` contract.

Each individual swap is executed via a `DELEGATECALL` to an executor contract. An executor contract handles the interaction with a specific AMM, like Balancer or Uniswap.

Executor contracts must be registered by accounts with the `EXECUTOR_SETTER_ROLE` before they can be used.

Four executors were present at time of review:

- `BalancerV2Executor`
- `UniswapV2Executor`
- `UniswapV3Executor`
- `UniswapV4Executor`

Tycho Router gives the caller a lot of freedom over interactions with AMMs. The caller can freely assemble and execute complex swap sequences. The caller can also choose the receiver of all but the last swap.

This makes it difficult to protect assets owned by Tycho Router between transactions. For more information see Assets at Risk, which follows this section.

No issues were found with access control.

Events are emitted for all security relevant events.

The contract can be paused in case of emergencies.

Assets at Risk

A vulnerability in Tycho Router can put assets at risk. These can be divided into four categories:

1. Allowances
2. ETH and ERC20 balances during swaps
3. ETH and ERC20 balances between transactions
4. Fees

Allowances

Tycho Router uses Permit2. Allowances given with Permit2 can remain after a swap, if they are larger than the sold amount and have a long enough deadline.

At the time of this assessment, none of the four executors called `ERC20.transferFrom`.

If an executor were added that called `ERC20.transferFrom`, this could put allowances at risk.

ETH and ERC20 Balances During Swaps

In other words, the assets being swapped.

It is easy to create swap sequences, whose execution results in a loss.

For example a sequence containing a swap, that sends its output to an address the caller doesn't control. A bug in the code that generates the swap sequences can produce such a swap.

If you're a user of Tycho Router, always set `minAmountOut` to the minimum amount you accept to receive. If you set `minAmountOut` to `0`, it's possible that you'll receive `0`, due to bad swap sequences or slippage.

For more information see Recommendations for Users of Tycho Router and Finding 6.

ETH and ERC20 Balances Between Transactions

PropellerHeads communicated, that Tycho Router is not supposed to own any ETH or ERC20 tokens between transactions. Such assets should be considered lost.

While there exists a mechanism to recover assets owned by Tycho Router, via the `withdraw` and `withdrawNative` functions, they can likely be withdrawn by other parties before recovery is possible. See Finding 1 and Finding 7.

Tycho Router can end up owning assets as the result of faulty swap sequences. For example, the second swap in a sequence could use only half of the output of the first swap, leaving the rest on the router after the transaction.

Due to the control given to the caller via the swap sequences, it is difficult to secure assets which are owned by Tycho Router.

The reviewer made an effort to recommend mitigations, that make withdrawing such assets more difficult.

Fees

If enabled, Tycho Router takes a fee and sends it to a specific `feeReceiver`.

It is possible to create swap sequences, that avoid this fee. See Finding 3.

Findings

1. Anyone Can Call Arbitrary Functions on Registered Executors

Severity: High (Execution of arbitrary code out of a set of choices)

Difficulty of Exploitation: Medium

Fix Status: Fixed

Description

`unlockCallback` allows anyone to execute a `DELEGATECALL` to any function on any registered executor.

This can be exploited in a variety of ways, including the following exploit scenario.

Exploit Scenario

Alice accidentally sends 10000 USDC to Tycho Router.

Bob deploys a `target` contract that acts like a `UniswapV2Pair`. `target`'s purpose is to make certain calls to itself succeed.

Bob calls `unlockCallback` choosing to `DELEGATECALL` the `swap` function on `UniswapV2Executor`.

`UniswapV2Executor.swap` does a transfer, whose parameters are fully in Bob's control:

```
tokenIn.safeTransfer(target, givenAmount);
```

Source

Bob chooses `tokenIn == USDC` and `givenAmount == 10000`, to transfer the 10000 USDC owned by Tycho Router to the `target` contract controlled by them.

As a result, Bob now owns Alice's 10000 USDC. Alice and the PropellerHeads team had no chance to recover them.

Recommendation

Don't allow the caller to choose the function selector that is called on an executor.

Ensure that current and future executors can only interact with contracts belonging to the executor's protocol. Especially, that executors can only send assets to contracts belonging to the executor's protocol.

Long term, avoid `DELEGATECALL`s if possible. Otherwise restrict the following properties of `DELEGATECALL`s as much as possible:

1. Who can call
2. Which contract can be called
3. Which function selector can be called

Fix Review

Confirmed fixed in the codebase at commit `59eb2195b60280bfba9f07b55bf0d7bc973ad23f`.

PropellerHeads hard coded the functions, which are called on executors. For example, a swap now always calls the `swap` function on the executor and a callback always calls the `handleCallback` function.

Additionally, PropellerHeads added checks, that executors only interact with contracts belonging to their respective protocol.

For example, `UniswapV2Executor.swap`'s `target` can now only be a pair of Uniswap V2.

2. Empty `receive` Function Provides No Protection Against Accidental ETH Transfers

Severity: Low (Loss of assets accidentally sent to Tycho Router)

Difficulty of Exploitation: Medium (Requires accidental transfer)

Fix Status: Fixed

Description

Because the `receive` function's body is empty, anyone can send ETH to `TychoRouter`:

```
receive() external payable {}
```

Source

It's worth making this more difficult, since accidentally sent ETH is likely lost, due to Finding 1 and Finding 7.

Exploit scenario

Bob accidentally sends ETH to Tycho Router.

Charlie uses a method described in Finding 1's or Finding 7's exploit scenario to transfer that ETH to themselves.

Bob can't recover their ETH.

Recommendation

Prevent a subset of accidental transfers by checking that the sender is a contract.

If possible, prevent a larger subset of accidental transfers by allowing only those addresses, which must send ETH to Tycho Router.

Fix Review

Confirmed fixed in the codebase at commit `59eb2195b60280bfba9f07b55bf0d7bc973ad23f`

EOAs can no longer accidentally send assets to Tycho Router:

```
receive() external payable {
    require(msg.sender.code.length != 0);
}
```

3. Fees Can Be Avoided

Severity: Undetermined (Trade-off by PropellerHeads, the only affected party)

Difficulty of Exploitation: Medium (Requires construction of a specific swap sequence)

Fix Status: Risk accepted by PropellerHeads

Description

Fees are only taken from the index in the `amounts` array that was incremented by the last swap.

The caller can choose the receiver of any swap.

The combination of both can be utilized to avoid fees.

Exploit Scenario

Alice wants to use Tycho Router without paying fees.

They create a swap sequence that outputs all but a tiny amount to addresses, that Alice controls.

The last swap, which swaps a tiny amount, chooses Tycho Router as its recipient.

Alice only pays fees on the output amount of the last swap.

As a result, Alice pays a dust amount in fees.

Recommendation

Don’t allow the caller to choose the receiver of individual swaps. Instead always send swap outputs to `address(this)`.

Once this change is made, the only way to get assets out of the router would be `amountOut`, on which the fee is paid.

4. Swaps Fail If Any Amount of Input Token Is Owned by The Router

Severity: High (Prevents anyone from using Tycho Router)

Difficulty of Exploitation: Low

Fix Status: Fixed

Description

`swap` reverts, if any balance of `tokenIn` remains on Tycho Router:

```
uint256 leftoverAmountIn =
IERC20(tokenIn).balanceOf(address(this));
if (leftoverAmountIn > 0) {
    revert
TychoRouter__AmountInNotFullySpent(leftoverAmountIn);
}
```

Source

Since executors only consume `amountIn`, any `tokenIn` balance before a `swap` causes the `swap` to fail.

Exploit Scenario

Charlie wants to prevent anyone from using Tycho Router. They send the smallest unit of the most common tokens to Tycho Router. As a result, all swaps for those tokens fail.

Recommendation

Instead of reverting, if Tycho Router's balance isn't zero, revert, if the balance hasn't decreased by `amountIn`,

Long term, avoid using absolute values of `address(this).balance` and `IERC20(token).balanceOf(address(this))`. Balances can be increased by anyone.

Fix Review

Confirmed fixed in the codebase at commit `f853739a3dafb57dc12c77d97a30703a6d65445a`

`swap` now reverts, if execution of the swap sequence didn't decrease Tycho Router's balance of `tokenIn` by `amountIn`; regardless of the exact value of the previous balance.

5. Send Only Forwards 2300 Gas

> **Severity**: Low (Assets can be recovered to another address)
>
> **Difficulty of Exploitation**: High (Unlikely scenario where assets are rescued to contract address)
>
> **Fix Status**: Fixed

Description

`withdrawNative`, which is used to rescue ETH, uses `send` to transfer ETH:

```
bool success = payable(receiver).send(amount);
```

Source

`send` only forwards 2300 gas.

As a result, the transaction will revert, if `receiver` is a contract, whose execution consumes more than 2300 gas.

Recommendation

Use OpenZeppelin's `Address.sendValue` or equivalent code, that forwards all gas, for ETH transfers.

Long term, avoid `transfer` and `send`, which only forward 2300 gas.

Fix Review

Confirmed fixed in the codebase at commit `59eb2195b60280bfba9f07b55bf0d7bc973ad23f`

All gas is now forwarded :

```
Address.sendValue(payable(receiver), amount);
```

Source

6. Slippage Protection Is Optional

Severity: High (Loss of swapped funds)

Difficulty of Exploitation: Medium (Requires faulty sequence of swaps and zero or low `minAmountOut`)

Fix Status: Fixed

Description

It is possible to call `swap` with parameters, that cause a loss of sold assets.

This can happen by accident or due to a bug in the code that generates the swap sequence.

Exploit scenario

Charlie writes a program that interacts with Tycho Execution and Tycho Router. They are in a hurry and decide to set the `minAmountOut` parameter of the `swap` function to `0`.

The integration works and performs well.

A new version of Tycho-Execution gets released. It contains a bug, where one step only uses a fraction of the assets bought in the previous step.

Charlie uses the new version to generate a solution that swaps 100000 USDC into ETH.

Tycho Router receives the 100000 USDC, the bug is triggered and only a fraction of ETH is sent to Charlie. The majority of assets remain on Tycho Router, where they are stolen by Charlie, using one of the methods laid out in Finding 1 and Finding 7.

As a result, Charlie has lost 90000.

Recommendation

Revert if `minAmountOut` is zero.

Warn users in code and documentation that they are only guaranteed to receive `minAmountOut`. If they set `minAmountOut` to `0`, it's possible that they'll receive `0`.

Long term, make it more difficult to use the protocol in insecure ways by making protections like `minAmountOut` mandatory.

Fix Review

Confirmed fixed in commit `5c28d77f1d92bb7fba5fa7495e77fcbb5e077eb8`

7. Anyone Can Withdraw Assets Owned by The Router

Severity: Medium (Assets owned by the router are considered unrecoverable)

Difficulty of Exploitation: Medium (Requires construction of a specific swap sequence)

Fix Status: Risk accepted by PropellerHeads

Description

The caller can choose the receiver for any executor swap.

`_swap`, which is called by `swap`, keeps track of input and output amounts in the `amounts` and `remainingAmounts` arrays.

The caller can freely index into both arrays because they control `tokenInIndex` and `tokenOutIndex`, which are taken directly from the call data.

This opens up at least two ways through which assets, which are owned by Tycho Router, can be withdrawn by anyone.

Exploit Scenario 1

Tycho Router owns 10000 DAI due to a faulty swap sequence.

Charlie sends a single swap that sells 10000 USDT and buys equivalent USDC, which are sent to a receiver address Charlie controls. The swap is net neutral for Charlie.

The swap writes `10000` into `amounts[tokenOutIndex]`. `amounts[tokenOutIndex]` is returned from `_swap` and assigned to `amountOut` in `swap`.

Since Charlie chose `tokenOut == DAI` and `amountOut` is `10000`, Tycho Router sends them the additional 10000 DAI.

Exploit Scenario 2

Bob accidentally sent 10000 DAI to Tycho Router.

Bob has contacted the PropellerHeads team to help with the recovery of their assets.

Alice sends a sequence of two swaps.

The first swap sells 10000 USDT and buys equivalent USDC, which are sent to the Tycho Router.

The swap executes and writes 10000 into `remainingAmounts[tokenOutIndex]`.

Since Alice controls `tokenInIndex`, they make 10000, previously written into `remainingAmounts[tokenOutIndex]`, the second swap's input amount.

The second swap simply swaps 10000 DAI into equivalent USDC at an address Alice controls. However, this time the assets sold are not Alice's, but rather Bob's.

Finally, Tycho Router sends 10000, which is the output amount of the second swap and `amountOut` of `tokenOut`, which Alice chose to be USDC, to `receiver`.

Bob is unable to recover his 10000 DAI.

Recommendation

Don't allow the caller to choose the receiver of individual swaps. Instead always send swap outputs to `address(this)`.

Measure the amount of `tokenOut` owned by Tycho Router at the very beginning of `swap`. Don't transfer any of that amount to `receiver`.