Tycho Router

Security Assessment

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

2025-03-03

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 assessment resulted in 7 findings.

Severity	Count
High	3
Medium	1
Low	3

The most severe is finding 4, which enabled anyone to freeze `TychoRouter`s `swap` function.

Recommendations for Users of `TychoRouter`

The single most important thing you can do to protect your assets is to always set the **`minAmountOut`** parameter of the `swap` function to an amount that reflects acceptable slippage.

For example, if you expect to receive `1000` USDC and accept 5% slippage, set `minAmountOut` to `950 * 10**6`.

If `TychoRouter` can't send `minAmountOut` of `tokenOut` to `receiver` at the end of the `swap` function, the transaction reverts.

If `minAmountOut` is set to `0` you risk loosing assets. See finding 6 for more information.

Avoid infinite Permit2. Set Permit2 allowance and deadline as low as possible.

Before interacting with `TychoRouter`, check the `ExecutorSet(address executor)` events emitted by the contract for executors that were not included in this assessment. New executors can alter the behavior and security of the system.

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

`TychoRouter` executes sequences of swaps that interact with various AMMs. The swaps are generated off-chain, encoded and sent via the calldata 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 allowlisted by accounts with the `EXECUTOR_SETTER_ROLE` before they can be used.

`TychoRouter` gives the caller a lot of freedom over the interactions with AMMs. The caller can freely assemble and execute complex swap sequences.

This makes it difficult to protect assets owned by `TychoRouter` between transactions. For more information see the section "Assets at Risk", which follows the section you're reading.

Four executors were present at time of review:

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

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 `TychoRouter` 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

`TychoRouter` uses Permit2. Allowances given with Permit2 can remain after a swap, if they are larger than the sold amount - including infinity - and have a long enough deadline.

At time of review 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.

The receiver of each swap is encoded in the calldata and can be chosen freely. Incorrect swap sequences can result in a loss of funds. Swaps can interact with low liquidity pools and incur high slippage.

It is easy to create a series of swaps that results in a loss. One simple example is a single swap that sends its output to an address the caller doesn't control. A bug in the code that generates the series of swaps can produce such a series.

If you're a user of `TychoRouter`, 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`.

For more information see Finding 6 and the section "Recommendations for Users of `TychoRouter`".

ETH and ERC20 Balances Between Transactions

PropellerHeads communicated, that `TychoRouter` is not intended 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 `TychoRouter`, via the `withdraw` and `withdrawNative` functions, they can likely be withdrawn by other parties before recovery is possible. See findings 1 and 7.

`TychoRouter` can end up owning assets as the result 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 fully secure assets which are owned by `TychoRouter`. The reviewer made an effort to recommend mitigations, that make withdrawing such assets more difficult.

Fees

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

It is possible, to create series of swaps that avoid this fee. See finding 3.

Findings

1. Anyone Can Call Arbitrary Functions on Registered Executors

Severity: High
Difficulty of Exploitation: Medium

Description

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

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

Exploit Scenario

Alice has accidentally sent 10000 USDC to `TychoRouter`.

Bob deploys a `target` contract that acts like a `UniswapV2Pair` to make the calls to it 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 choses `tokenIn == USDC` and `givenAmount == 10000`, to transfer the 10000 USDC owned by `TychoRouter` 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 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 they can call
3. Which function selector they can call

Fix review

Fixed in 59eb2195b60280bfba9f07b55bf0d7bc973ad23f

PropellerHeads hardcoded 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.

Furthermore, PropellerHeads added checks, that the contracts executors interact with belong to their respective protocol.

For example, `UniswapV2Executor.swap`'s `target` now can only be a UniswapV2Pair.

2. Empty receive Function Provides no Protection Against Accidental ETH Transfers

Severity: Low (Loss of accidentally transfered assets)
Difficulty of Exploitation: Medium (Requires accidental transfer)

Description

The `receive` function is empty:

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

Source

This allows anyone to send ETH to `TychoRouter`.

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

Exploit scenario

Bob accidentally sends ETH to `TychoRouter`.

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

Bob has no way to 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 allowlisting addresses, which must send ETH to `TychoRouter`.

Fix Review

Fixed in 59eb2195b60280bfba9f07b55bf0d7bc973ad23f.

EOAs can no longer accidentally send assets to `TychoRouter`:

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

3. Fees Can Be Avoided

Severity: To be determined by PropellerHeads
Difficulty of Exploitation: Medium (Requires construction of a specific swap sequence)

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 `TychoRouter` without paying fees.

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

The last swap, which swaps a miniscule amount, chooses `TychoRouter` 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)`. As a result, the only way to get assets out of the router is the `amountOut`, on which the fee is paid.

Fix Review

Risk accepted by PropellerHeads

4. Swaps Fail if any Amount of Input Token is Owned by the Router

Severity: High (Prevents anyone from using `TychoRouter`)
Difficulty of Exploitation: Low

Description

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

```
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 `TychoRouter`. They send the smallest unit of the most common tokens to `TychoRouter`. As a result, all swaps for those tokens fail.

Recommendation

Instead of reverting, if balance isn't zero, revert, if 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

Not yet fully fixed.

5. Send only forwards 2300 gas

Severity: Low
Difficulty of Exploitation: High (Unlikely scenario where assets are rescued to contract address)

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

Fixed in 59eb2195b60280bfba9f07b55bf0d7bc973ad23f

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

6. Slippage Protection Is Optional

Severity: High (Loss of swapped funds)
Difficulty of Exploitation: High (Requires faulty sequence of swaps and dangerously low `minAmountOut`)

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 uses Tycho-Execution programmatically. 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.

`TychoRouter` receives the 100000 USDC, the bug is triggered and only a fraction of ETH is sent to Charlie. The majority of assets remain on `TychoRouter`, where they are stolen by Charlie, using one of the methods laid out in findings 1 and 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`. It is their responsibility to set `minAmountOut` to a value that represents acceptable slippage. Otherwise they risk loosing assets.

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

Fix Review

Waiting for PropellerHeads

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)

Description

The caller can choose the receiver of the assets received for any executor.

`_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 calldata.

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

Exploit Scenario 1

`TychoRouter` 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`, `TychoRouter` sends him the additional 10000 DAI.

Exploit Scenario 2

Bob accidentally sent 10000 DAI to `TychoRouter`.

Bob has contacted the PropellerHeads team to help them with the recovery of the funds.

Alice sends a sequence of two swaps.

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

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, `TychoRouter` 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 `TychoRouter` at the very beginning of `swap`. Don't transfer any of that amount to `receiver`.

Fix Review

Waiting for PropellerHeads