Tycho Router

Security Assessment 2

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

2025-08-05

Disclaimer

This security assessment was timeboxed to 7 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
- Scope
- User checklist
- Executor checklist
- Findings
    - 1. UniswapV4Executor Obtains Own Address Using Risky Method
    - 2 Function Selector of DELEGATECALL Is Not Constrained
    - 3. Transient Storage Slot Remains Set to Executor Until Following Swap
    - 4. Magic Number Used as Transient Storage Slot
    - 5. Anyone Can Withdraw ETH That Is Owned by The Router
    - 6. Supporting New Callback Requires Redeployment
    - 7. Executors Give Infinite Allowances
    - 8. Balance Delta check Ignored For Arbitrage Swaps
    - 9. Control Over Encoded Swaps Allows Transfer of Caller's Allowances
    - 10. Power to Add Executors Grants Power to Steal Allowances
    - 11. Transfer Forwards Just 2300 Gas
    - 12. Default Way of Using Tycho Trusts it Entirely
    - 13. EkuboExecutor Largely Uses Assembly And Is Hard to Understand

Summary

PropellerHeads engaged Maximilian Krüger to assess the security of the `TychoRouter` contract over 7 person-days between 2025-04-22 and 2025-05-22. The code in scope consists of around 1450 lines.

The assessment focused primarily on the following:

1. Malicious or accidental loss of assets that are being swapped, the largest pool of assets at risk.
2. Loss of allowances, the second largest pool of assets at risk.
3. Denial of Service i.e. disrupting `TychoRouter`'s normal function.
4. Loss of assets that are owned by the router. Low priority since such assets are considered lost.

The assessment resulted in 13 findings with the following severities:

Severity	Count
High	4
Medium	1
Low	3
Undetermined	2
Informational	3

The most severe finding is 9, which allowed entities that are in control of the encoded swaps - such as the `tycho-execution` Rust crate - control over the caller's allowances.

PropellerHeads accepted the risk of findings 7 and 13. Findings 10 and 12 were partially fixed. All other findings were fixed.

A security assessment can not guarantee that the assessed system is secure. It is possible that further vulnerabilities exist that could not be determined within the allocated time frame.

Even if it were possible to guarantee that no further vulnerabilities exist, the addition of a new executor on-chain, which can happen anytime, can introduce

additional vulnerabilities. We recommend that developers of new executors follow the Executor Checklist.

Even if it were possible to guarantee that no further vulnerabilities exist, users could still loose assets by using `TychoRouter` incorrectly. We recommend that users of `TychoRouter` follow the User Checklist.

Scope

The assessment focused on commit `6b3a26f3920edfccba6c665702d73bb163a9f005` of the tycho-execution repository.

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

- `TychoRouter`
- `Dispatcher`
- `executors`
  - `BalancerV2Executor`
  - `CurveExecutor`
  - `EkuboExecutor`
  - `TokenTransfer`
  - `UniswapV2Executor`
  - `UniswapV3Executor`
  - `UniswapV4Executor`

The repository also contains a `tycho-execution` Rust crate, which was not in scope.

During the assessment one more executor called `MaverickV2Executor` was added. Its version at commit `bcef8f69f628c3a37a7cb5462e26200ad5aab1ee` was added to the scope.

User Checklist

Follow this checklist when using `TychoRouter`. This checklist is not exhaustive. It is necessary for security but not sufficient.

> **Always** set the `minAmountOut` parameter of `TychoRouter`'s `swap...` functions to the smallest amount you accept to receive.
>
> For example, if you expect to receive `1000` USDC and accept 5% slippage, set `minAmountOut` to `950 * 10**6`.
>
> If you set `minAmountOut` to `1`, it's possible that you'll receive `1` due to faulty swap sequences or slippage.
>
> Check the price information that is used to determine `minAmountOut` against at least one other independent price source. Incorrect price information can result in an unacceptably small `minAmountOut` that can cause losses. See finding 12 for more information.

- Ensure all addresses that have `EXECUTOR_SETTER_ROLE` or `DEFAULT_ADMIN_ROLE` are multi-sigs and none are EOAs. See finding 10 for more information.
- When manually transfering assets to `TychoRouter` i.e. when not relying on `transferFrom`, always set the `swap...` functions' `isTransferFromAllowed` parameter to `false`. See finding 9 for more information.
- Never approve infinite allowances.
- Never approve infinite Permit2.
- Set Permit2 allowance and deadline as low as possible
- Check the `ExecutorSet(address executor)` events emitted by the contract. Check that all executors are included in this or future assessments. New executors can alter the behavior and security of the system.
- Avoid sending assets to `TychoRouter` without swapping those assets in the same transaction. Assets that are left on `TychoRouter` at the end of a transaction are not considered recoverable.

Executor Checklist

Follow this checklist when developing a new executor. This checklist is not exhaustive. It is necessary for security but not sufficient.

> An executor's code must not contain any call to `ERC20.transferFrom` or `Permit2.transferFrom`.
>
> Any transfer of allowances must use `RestrictTransferFrom._transfer` or else the executor puts allowances at risk.
>
> See finding 9 for more information.

- Review every executor's security before it is added.
- An executor should not execute any `DELEGATECALL`s. If `DELEGATECALL`s are unavoidable, ensure that the caller can't control the `DELEGATECALL`'s target and function selector.
- Introduce a 24 hour delay between addition and activation of new executors. This allows users to do their own due diligence of new executors. See finding 10 for more information.
- Require that callbacks, which are handled via the `handleCallback` function, originate from the correct protocol by checking `msg.sender`.
- Note that an executor's `handleCallback` function's **`data`** argument contains the raw ABI-encoded call data, which must be decoded manually. See finding 6 for more information.
- Note that an executor's `handleCallback` function's return data contains the raw ABI-encoded return data, which must be encoded manually.

Findings

1. `UniswapV4Executor` Obtains Own Address Using Risky Method

> **Severity**: Undetermined

> **Difficulty of Exploitation**: High (Requires another weakness)
>
> **Fix Status**: Fixed

Description

`UniswapV4Executor` executes a `DELEGATECALL` to itself. This requires `UniswapV4Executor` to know its own address. Since `UniswapV4Executor` itself is called via a `DELEGATECALL`, getting its address via `address(this)` is not an option.

Instead, `UniswapV4Executor` reads its own address via `tload(0)`, expecting that `TychoRouter` has previously written the current executor's address to transient storage slot `0`.

```
address executor;
// slither-disable-next-line assembly
assembly {
    executor := tload(0)
}

if (executor == address(0)) {
    executor = address(this);
}
// here we expect to call either `swapExactInputSingle`
or `swapExactInput`. See `swap` to see how we encode the
selector and the calldata
// slither-disable-next-line low-level-calls
(bool success, bytes memory returnData) =
executor.delegatecall(data);
```

UniswapV4Executor

Exploit Scenario

Due to an unrelated issue, the transient storage slot `0` is set to the address of another executor.

As a result, `UniswapV4Executor` `DELEGATECALL`s another executor, with undetermined consequences.

Recommendation

Store `UniswapV4Executor`'s address in immutable state variable when its constructor is run. Since immutable state variables are compiled into the bytecode, the state variable will always contain `UniswapV4Executor`'s address, even during `DELEGATECALL`s.

```
address private immutable _self;

constructor() {
    _self = address(this);
}
```

Fix Review

Confirmed fixed at pull request 180.

2. Function Selector of `DELEGATECALL` Is Not Constrained

Severity: Undetermined

Difficulty of Exploitation: High (Requires another weakness)

Fix Status: Fixed

Description

`UniswapV4Executor` executes a `DELEGATECALL` to itself in the callback it is supposed to receive from Uniswap V4's `PoolManager`. The `DELEGATECALL` is only supposed to call one of two functions, yet a call to any function is permitted.

Exploit Scenario

A `DELEGATECALL` to one of `UniswapV4Executor`'s functions that is outside the set of permitted functions is executed, with potentially harmful effects.

Recommendation

Revert if the function selector is not in the allowed set.

Fix Review

Confirmed fixed at commit `4de1d104062d4aefbde22031d9f31884be5d49ad`.

3. Transient Storage Slot Remains Set to Executor Until Following Swap

Severity: Medium (Execution of callback at time callbacks should not be executed

Difficulty of Exploitation: High (Requires another weakness)

Fix Status: Fixed

Description

`Dispatcher` sets transient storage slot `0` to the address of an executor and then `DELEGATECALL` s that executor:

```
assembly {
    tstore(0, executor)
}

// slither-disable-next-line controlled-delegatecall,low-level-calls,calls-loop
(bool success, bytes memory result) = executor.delegatecall(
    abi.encodeWithSelector(IExecutor.swap.selector, amount, data)
);
```

Dispatcher.sol

This enables callbacks from protocols to execute on the current executor.

The transient storage slot remains set until the next `DELEGATECALL` to an executor.

Recommendation

To prevent callbacks from getting directed to an executor that has finished executing, unset the transient storage slot right after the call to the executor:

```
assembly {
    tstore(0, executor)
}

(bool success, bytes memory result) =
executor.delegatecall(
    abi.encodeWithSelector(IExecutor.swap.selector,
amount, data)
);

assembly {
    tstore(0, 0)
}
```

Fix Review

Confirmed fixed at commit `1b003dc483d0e6cc01c5fd6d46ea9f28fc1c0aa6`.

4. Magic Number Used as Transient Storage Slot

Severity: Informational

Fix Status: Fixed

Description

`Dispatcher` stores the current executor's address in transient storage slot `0`:

```
assembly {
    tstore(0, executor)
}
```

Dispatcher.sol

This risks collisions with other contracts that use transient storage slot `0`.

The number `0` provides no meaning to readers of the code. To infer its meaning other occurrences of `tstore(0, x)` and `tload(0)` must be investigated.

Exploit Scenario

A new executor stores data in transient storage slot `0`.

Reads and writes of storage slot `0` collide, with undetermined consequences.

Recommendation

Avoid transient storage slot collisions by storing the slot in a constant variable that is set to the hash of a string containing the contract name and state variable:

```
uint256 private constant
_CURRENTLY_SWAPPING_EXECUTOR_SLOT =
keccak("TychoRouter#_CURRENTLY_SWAPPING_EXECUTOR_SLOT");
// ...
assembly {
    tstore(_CURRENTLY_SWAPPING_EXECUTOR_SLOT, executor)
}
```

Avoid magic numbers. Magic numbers are values that don't have any description of what they are or why they exist.

For example, almost every value in the following pseudocode is a magic number:

```
tstore(0, false);
sstore(100, "cancelled");
updatePermissions("0xe592427a0aece92de3edee1f18e0157c0586156
4, true);
```

- What is meant by `0` in the first line?
- What is meant by `100` in the second line?
- What's at address `0xe592427a0aece92de3edee1f18e0157c05861564`?
- What is meant by `4` in the third line?
- What is meant by `true` in the third line?

It's not possible to understand those 3 lines without additional information.

Even though the code is extremely short, the reader will have to spend several minutes looking up additional information, including `0xe592427a0aece92de3edee1f18e0157c05861564`, other uses of transient storage slot `0` in the codebase, other uses of storage slot `100` in the codebase, and `setActor`'s implementation.

Code that uses magic numbers presents a puzzle to reviewers and other developers, which will have to hunt for clues elsewhere. Avoid magic numbers and strive to make code understandable in isolation.

Here's the example above rewritten to be understandable in isolation:

```
IS_PAUSED_TRANSIENT_STORAGE_SLOT = 0;
tstore(IS_PAUSED_TRANSIENT_STORAGE_SLOT, false);

AUCTION_STATUS_STORAGE_SLOT = 100;
sstore(AUCTION_STATUS_STORAGE_SLOT, "cancelled");

uniswap_v3_router =
"0xe592427a0aece92de3edee1f18e0157c05861564";
CO_OWNER_ROLE = 4;
can_access_vault = true;

updatePermissions(uniswap_v3_router, CO_OWNER_ROLE,
can_access_vault);
```

In general, there are many advantages to assigning a value, if it is not self explanatory, to a well-named variable and then using the variable instead of the value:

- The variable name contains information that helps with understanding.
- The variable name connects all uses of the value. This is valuable information that doesn't exist otherwise.
- All uses of the value can be found easily by searching for the variable name.
- The value can be changed in one place.
- There's less risk that the value is changed in one place but not in another.

Fix Review

Confirmed fixed at pull request 182.

5. Anyone Can Withdraw ETH That Is Owned by The Router

Severity: Low (Assets owned by router are considered lost)

Difficulty of Exploitation: Low

Fix Status: Fixed

Description

If `msg.value == 0`, `TychoRouter._wrapETH` deposits all ETH that is owned by `TychoRouter` into WETH. The caller can use that WETH in swaps and withdraw it.

This is due to `_wrapETH` succeeding, if `msg.value == 0`:

```
function _wrapETH(uint256 amount) internal {
    if (msg.value > 0 && msg.value != amount) {
        revert
TychoRouter__MessageValueMismatch(msg.value, amount);
    }
    _weth.deposit{value: amount}();
}
```

TychoRouter

Exploit Scenario

`TychoRouter` owns `n` ETH.

Attacker Alice calls `TychoRouter.singleSwap` with `msg.value == 0`, `wrapEth == true` and `amountIn == n`. The `n` ETH is deposited into WETH. Alice uses the WETH in swaps and ultimately withdraws its value to an address they control.

Recommendation

Make `TychoRouter._wrapETH` revert if `msg.value != amount` regardless of the value of `msg.value`:

```
function _wrapETH(uint256 amount) internal {
    if (msg.value != amount) {
        revert
TychoRouter__MessageValueMismatch(msg.value, amount);
    }
    _weth.deposit{value: amount}();
}
```

Fix Review

Confirmed fixed at commit `04000059e5e7940f209639c900557b437a766913`.

6. Supporting New Callback Requires Redeployment

Severity: Informational

Fix Status: Fixed

Description

Even though it looks almost identical to `fallback`, PropellerHeads communicated that the `TychoRouter.unlockCallback` function is necessary. Removing `unlockCallback` and handling it via `fallback` made `UniswapV4Executor`'s tests fail for reasons that were not known to the PropellerHeads team.

```
function unlockCallback(bytes calldata data)
    external
    returns (bytes memory)
{
    if (data.length < 24) revert
TychoRouter__InvalidDataLength();
    bytes memory result = _handleCallback(data);
    return result;
}
```

TychoRouter

Furthermore, the `fallback` function only returned the raw first 16 bytes of the `bytes memory` returned by `_handleCallback`:

```
fallback() external {
    bytes memory result = _handleCallback(msg.data);
    // slither-disable-next-line assembly
    assembly ("memory-safe") {
        // Propagate the calculatedAmount
        return(add(result, 32), 16)
    }
}
```

TychoRouter

As a result, `fallback` is unable to handle callbacks of executors other than `EkuboExecutor`.

Scenario

A new executor is added whose protocol requires a callback that returns more than 16 bytes.

To handle the callback a special method must be added to `TychoRouter`. Finally, `TychoRouter` must be redeployed.

Recommendation

Use an architecture where modification and redeployment of `TychoRouter` is not required every time a new callback needs handling.

The failing tests were investigated and the following was found to work:

```
contract TychoRouter {
    // ...
    fallback(bytes calldata data) external returns (bytes
memory) {
        return _handleCallback(data);
    }
    // ...
    // remove unlockCallback
}
```

```
contract UniswapV4Executor {
    // ...
    function handleCallback(bytes calldata data)
        external
        returns (bytes memory)
    {
        // remove 4-byte function selector
        // as well as 32-byte offset
        // and 32-byte size of ABI-encoded `bytes data`
        bytes calldata stripped = data[68:];
        verifyCallback(stripped);
        return abi.encode(_unlockCallback(stripped));
    }
    // ...
}
```

Note that an executor's `handleCallback` function's **`data`** argument contains the raw ABI-encoded call data, which must be decoded manually. An executor's `handleCallback` function's return data contains the raw ABI-encoded return data, which must be encoded manually.

It's important to remember, that the special `fallback` function handles call data and return data differently than a normal function:

```
function handleCallback(bytes calldata data) external
returns (bytes memory)
```

means that `data` is expected to be ABI-encoded. `handleCallback` will automatically ABI-decode `data` and ABI-encode the returned data.

```
fallback(bytes calldata data) external returns (bytes
memory)
```

looks almost identical but will do no decoding or encoding. Both must be done manually.

Fix Review

Confirmed fixed at commit `f14c8ee29ba67ca4cd8cfe8b00e8b907a0352f9a`.

7. Executors Give Infinite Allowances

Severity: Low (Assets owned by the router after transaction are considered lost)

Difficulty of Exploitation: High (Requires weakness in external protocol)

Fix Status: Risk accepted by PropellerHeads

Description

`BalancerV2Executor` and `CurveExecutor` allow Balancer vault and Curve pool to infinitely spend a specific token `TychoRouter` owns.

PropellerHeads communicated that this prevents repeated calls to approve. If a token is already approved, the approval is skipped (`needsApproval == false`) saving gas.

```
if (needsApproval) {
    // slither-disable-next-line unused-return
    tokenIn.forceApprove(VAULT, type(uint256).max);
}
```

BalancerV2Executor

Exploit Scenario

A vulnerability in Balancer or Curve allows an attacker to spend allowances given to Balancer vault or a Curve pool, including assets owned by `TychoRouter` .

Recommendation

Never give infinite allowances. Always approve the smallest feasible amount.

Fix Review

Risk Accepted by PropellerHeads.

8. Balance Delta Check Ignored For Arbitrage Swaps

Severity: High (Loss of swapped funds)

Difficulty of Exploitation: High (Requires another weakness)

Fix Status: Fixed

Description

`TychoRouter`'s `swap...` functions check that `amountOut` returned from the executor matches the increase of the `receiver`'s balance of `tokenOut`.

```
if (tokenIn != tokenOut) {
    uint256 currentBalanceTokenOut = _balanceOf(tokenOut,
receiver);
    uint256 userAmount = currentBalanceTokenOut -
initialBalanceTokenOut;
    if (userAmount != amountOut) {
        revert TychoRouter__AmountOutNotFullyReceived(
            userAmount, amountOut
        );
    }
}
```

TychoRouter

This check is ignored for arbitrage swaps, where `tokenIn == tokenOut`.

Exploit Scenario

Alice does an arbitrage swap. There is slippage that is not expected by the executor. Alice looses funds.

Recommendation

Don't ignore the check for arbitrage swaps.

Fix Review

Confirmed fixed at commit `70230bf05f4bdfdd54f62799b72bd998d351983e`.

9. Control Over Encoded Swaps Allows Transfer of Caller's Allowances

Severity: High (Loss of allowances caller has given router)

Difficulty of Exploitation: High (Requires control over encoded swaps)

Fix Status: Fixed

Description

Encoded swaps can contain instructions to transfer allowances the caller has given `TychoRouter`.

This much increases the trust placed in the encoded swaps and by extension in the components that generate them.

Previously, no trust had to be placed in encoded swaps, provided `minAmountOut` was set to an acceptable slippage. As long as `minAmountOut` was received, the exact sequence of swaps to achieve this didn't matter. This was only the case because a swap could only access assets that were directly owned by `TychoRouter`.

Now swaps can also access the caller's allowances.

As a result, malicious or faulty swaps can lead to loss of a caller's allowances.

Exploit Scenario A

Attacker Charlie manages to attack the supply chain of component involved in producing and encoding the swaps. Charlie makes the component produce swaps that transfer some of the caller's allowances to an address Charlie controls.

Bob uses the component to produce swaps and sends them to `TychoRouter`. Some of Bob's allowances have been stolen by Charlie.

Exploit Scenario B

A new version of a component that is involved in generating and encoding the swaps is released. It contains a bug that causes certain swaps to transfer more of

the caller's allowances than necessary into `TychoRouter`.

Alice uses the new version of the component to produce swaps and sends them to `TychoRouter`. More of Alice's allowances than necessary are transfered into `TychoRouter`, where they remain after the transaction.

Attacker Charlie withdraws the assets that remain on `TychoRouter` after the transaction using a specially crafted sequence of swaps.

Some of Alices's allowances have been stolen by Charlie.

Recommendation

Ensure that only allowances the caller intends to use in the swap can be transfered during the swap.

Ideas on how this can be achieved:

Allow and require the caller of any swap function to specify whether transfers of allowances can take place at all and how much of `tokenIn` can be transferred from their allowances. Since this is a mechanism to reduce trust in the encoded swaps, this specification has to be passed in parameters that are separate from the encoded swaps.

Remember this information in transient storage.

Only permit executors to transfer allowances via one specific function that checks whether transfers are allowed and whether the amount can be transfered.

Fix Review

Confirmed fixed at pull request 198.

10. Power to Add Executors Grants Power to Steal Allowances

Severity: High (Execution of arbitrary code that can steal allowances)

Difficulty of Exploitation: High (Requires `EXECUTOR_SETTER_ROLE`)

Fix Status: Partially Fixed

Description

Executors' code is executed via `DELEGATECALL` in the context of `TychoRouter`. This gives an executor the power to transfer `TychoRouter`'s allowances.

Anyone can execute any registered executor's `swap` function by calling `TychoRouter.singleSwap` with the right calldata.

Any account with the `EXECUTOR_SETTER_ROLE` can register arbitrary executors, execute arbitrary code, and steal any of `TychoRouter`'s allowances.

Exploit Scenario

The value of allowances users have approved `TychoRouter` to spend exceeds 1 million USD.

PropellerHeads developer Charlie stores the private key of an account that has `EXECUTOR_SETTER_ROLE` on their computer.

Attacker Bob tricks Charlie into cloning a Rust repository from Github. The `build.rs` script of the Rust repository executes malicious code that manages to steal the account's private key. Similar attacks are frequent recently.

Bob uses Charlie's account to add a malicious executor.

The malicious executor's `swap` function executes `token.transferFrom(spender, bob, amount)`, where `bob` is an account Bob controls and `token`, `spender` and `amount` are taken from the protocol data, which the caller, in this case Bob, also controls.

Bob calls the executor repeatedly via `TychoRouter.singleSwap` to transfer `TychoRouter`'s allowances to themselves.

As a result, users that have approved `TychoRouter` to spend their assets loose those assets.

Recommendation

Increase the difficulty of adding executors, by only giving `EXECUTOR_SETTER_ROLE` to a multi-sig. The `DEFAULT_ADMIN` role can give `EXECUTOR_SETTER_ROLE` to other accounts and should also be a multi-sig.

Introduce a 24 hour delay between addition and activation of new executors. This gives PropellerHeads and `TychoRouter`'s users time to react to the exploit scenario above. Set up alerts for the addition of new executors so the PropellerHeads team is notified and can react.

Fix Review

The following reflects the state at mainnet block 23054604 (2025-08-02). Since privileged accounts can be added and removed at any time, it might not reflect the most recent state.

`TychoRouter` is deployed to mainnet address `0xfD0b31d2E955fA55e3fa641Fe90e08b677188d35`. The most recent address can be found here.

`EXECUTOR_SETTER_ROLE` is granted to the following mainnet addresses:

- `0x06e580b872a37402764f909fccab0eb5bb38fe23` (multi-sig)

`DEFAULT_ADMIN_ROLE`, which is a superset of `EXECUTOR_SETTER_ROLE`, is granted to the following addresses:

- `0xa30afdd6dec77eba4d0057110d8f7aa5d9c96f29` (multi-sig)

At block 23054604 (2025-08-02) adding an executor requires multiple signatures.

Propellerheads communicated, that this is also the case for the the Unichain and Base deployments.

The recommended time delay was not implemented.

11. `transfer` Forwards Just 2300 Gas

Severity: Low (Loss of transaction fees)

Difficulty of Exploitation: Medium (Requires receiver to be contract)

Fix Status: Fixed

Description

The lowlevel solidity `transfer` function forwards just 2300 gas.

It will fail for most receivers that are contracts.

Exploit Scenario

Alice uses `TychoRouter` to swap tokens into ETH and send that ETH to her contract.

Alice's contract runs out of gas since 2300 gas are not enough to execute it.

As a result, the swap transaction fails and Alice looses the transaction fee.

Recommendation

Use Openzeppelin's `Address.sendValue`, which forwards all gas.

Fix Review

Confirmed fixed at pull request 198.

12. Default Way of Using Tycho Trusts it Entirely

Severity: High (Loss of swapped funds)

Difficulty of Exploitation: High (Requires other weakness)

Fix Status: Partially Fixed

Description

The `tycho-execution` repository contains a `tycho-execution` Rust crate, which was not in scope. The Rust crate is the default way of interacting with `TychoRouter`.

It became evident in communication with PropellerHeads, that the default way of using the Rust crate involves the crate directly generating the calldata of transactions that are sent to the `TychoRouter` contract:

```
let encoder = TychoRouterEncoderBuilder::new()
    .chain(Chain::Ethereum)
    .swapper_pk(swapper_pk)
    .build()
    .expect("Failed to build encoder");
```

main.rs

```
let tx = encoder
    .encode_calldata(vec![solution.clone()])
    .expect("Failed to encode router calldata")[0]
    .clone();
```

main.rs

`TychoRouter`'s `swap` functions revert the transaction, if `receiver` didn't receive at least `minAmountOut` of `tokenOut`.

This reduces trust in the encoded swaps and the Rust crate that generates them.

Since both the encoded swaps and the `minAmountOut` parameter are controlled by the `tycho-execution` Rust crate, the crate becomes a single point of failure.

As a result, bugs in or compromise of the Rust crate can lead to a loss of assets.

Exploit Scenario A

A bug or bad price information makes the `tycho-execution` Rust crate produce a `minAmountOut` that is far below current market prices.

As a result, the swap sequence is vulnerable to sandwich attacks.

Exploit Scenario B

Alice runs a DEX that uses Tycho-Execution. Like the majority of users, Alice assumes the advertised default way is a safe way to use Tycho-Execution. The DEX's code hands the entire construction of transaction calldata to the Rust crate.

Attacker Charlie compromises the `tycho-execution` Rust crate through a supply chain attack. Charlie makes the crate generate calldata that always sets `minAmountOut` to zero and sends a large fraction of the swapped funds to themselves.

Alice runs `cargo update`, which updates to the compromised version, and redeploys.

Bob uses Alice's DEX to swap 20000 USD worth of assets. As a result, Bob looses 10000 USD, which is sent to Charlie.

Recommendation

Have the user be responsible for the ABI-encoding of calldata as well as the sourcing of the `minAmountOut` parameter and other security critical parameters. Make the secure way the default and recommended way of using the `tycho-execution` crate.

A swap can always loose assets due to slippage, unless slippage is explicitely determined and checked against an acceptable value.

Slippage can have many causes including bugs, low liquidity and sandwich attacks.

Making the component that computes swaps also responsible for controlling the slippage makes that component a single point of failure.

Long term, make the default way secure and avoid single points of failure.

Fix Review

PropellerHeads communicated that examples and documentation now have user code be responsible for the construction of calldata and that the `tokenIn`, `tokenOut` and `amountIn` parameters no longer pass through the `tycho-execution` Rust crate.

The `minAmountOut` parameter, however, is still determined by the `tycho-execution` Rust crate, which continues to be a single point of failure, into which a lot of trust must be placed.

13. `EkuboExecutor` Largely Uses Assembly And Is Hard to Understand

Severity: Informational

Fix Status: Risk accepted by PropellerHeads

Description

A large fraction of `EkuboExecutor` is written in assembly instead of Solidity.

Most function names don't convey what the functions are doing. There are no doc strings. To understand the purpose of a function, one has to read its implementation. For example, there exist functions with the similar names `locked`, `_lock` and `_locked`, whose differences can only be found by reading the functions' implementations.

As a result, we consider `EkuboExecutor`'s code to be of a lower quality and present a higher risk than the rest of the codebase.

Recommendation

Avoid assembly when possible. Add a docstring to every function. Give each function a descriptive name that allows the reader to intuit much of the function's behavior.