yAudit Qantura Rebalance Review

Review Resources:

- smart contracts

Auditors:

- Panda
- HHK

Table of Contents

1 Review Summary
2 Scope
3 Code Evaluation Matrix
4 Findings Explanation
5 Critical Findings
6 High Findings
7 Medium Findings
   a 1. Medium - The owner transferring coordinator ownership will not be able to recreate one
8 Low Findings
   a 1. Low - `approve` will always revert if the `IERC20` interface mismatch
   b 2. Low - Missing `feeAmount == 0` check
   c 3. Low - `removeAccount()` function may not execute as expected and can be improved
9 Gas Saving Findings
   a 1. Gas - `onlyCoordinator()` modifier can be inlined

b 2. Gas - Useless `recipientToken == address(0)` check in `rebalanceTrigger()`
c 3. Gas - State variables only set in the constructor should be declared `immutable`
d 4. Gas - Not needed `address(0)` check in `_buildQueue()`
e 5. Gas - Reduce storage access throughout the contracts
10 Informational Findings
a 1. Informational - Withdraw and sweep functions may leave dust for rebasing tokens
b 2. Informational - Unused errors present
c 3. Informational - Unused import
d 4. Informational - Useless `withAccount` parameter in `CoordinatorFactory`
e 5. Informational - Non-assembly method available
f 6. Informational - Add an execute function to `Account` and `Coordinator`
11 Final remarks

Review Summary

Qantura Rebalance

Qantura Rebalance is an integration built on top of Gnosis Pay (Gnosis Card) and Cow protocol. It allows users to earn yield and automatically top up their card balance when needed.

The contracts of the Qantura Rebalance Repository were reviewed over four days. The code review was performed by two auditors between 6th September and 11th September 2024. The repository was under active development during the review, but the review was limited to commit 9dc082eb4e4f6f9e3f001b4738af95e6c70f0157 for the Qantura Rebalance repo.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
src
├── Account.sol
├── AccountFactory.sol
├── Coordinator.sol
├── CoordinatorFactory.sol
├── Factory.sol
├── FactoryErrors.sol
├── GPv2Order.sol
├── IntermediaryDeployer.sol
└── SafeTransfer.sol
```

After the findings were presented to the Qantura Rebalance team, fixes were made and their respective commit hashes are documented below.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Qantura Rebalance and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Appropriate use of access control modifiers like `onlyOwner` and `onlyCoordinator`. No major access control issues were identified.
Mathematics	Good	No significant mathematical errors found. The calculations appear correct.

Category	Mark	Description
Complexity	Good	Overall, code complexity is reasonable. Some functions like `removeAccount()` could be simplified.
Libraries	Good	Proper use of libraries like SafeERC20. No issues with library usage were identified.
Decentralization	Good	The system allows for decentralized management of accounts and funds. No centralization risks were identified.
Code stability	Good	Code appears stable overall. Some minor optimizations and improvements were suggested.
Documentation	Good	Code is generally well-documented, with comments explaining the functionality.
Monitoring	Good	The contracts include appropriate event emissions for key actions.
Testing and verification	Average	The auditors acknowledge that full testing is challenging due to the Cow protocol integration. However, some conditions outside of COW protocol lacked coverage.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas savings
  - Findings that can improve the gas efficiency of the contracts.
- Informational
  - Findings including recommendations and best practices.

Critical Findings

None.

High Findings

None.

Medium Findings

1. Medium - The owner transferring coordinator ownership will not be able to recreate one

When an owner transfers the ownership of a coordinator, they cannot recreate a new coordinator. This issue arises from the deployment process that uses a deterministic address. As a result, once ownership is transferred, the deterministic address remains tied to the previous deployment, preventing the creation of a new coordinator by the same owner.

Technical Details

The deployment of a coordinator is using a deterministic address derived from the owner's address, specifically through the following mechanism:

```
File: CoordinatorFactory.sol
27:         bytes32 salt = keccak256(abi.encode(owner));
28:
29:         address proxy = address(_deployContract(COORDINATOR, salt));
```

CoordinatorFactory.sol#L27-L29

The deployment logic relies on a fixed salt derived from the owner's address. Once a coordinator is deployed and ownership is transferred, subsequent attempts by the same owner to deploy a new coordinator will result in a conflict because the existing coordinator already uses the address derived from the salt.

Impact

Medium

Recommendation

Consider disabling ownership transfer if possible.

Developer Response

After discussing this issue, we decided to kill the transferability of ownership. Fixed in commit 0b0fa8c93dcddc040c59219f1ae0ccb560bc89ef.

Low Findings

1. Low - `approve` will always revert if the `IERC20` interface mismatch

Some tokens, such as `USDT`, have a different implementation for the approve function. When the address is cast to a compliant IERC20 interface, and the approve function is used, it will always revert due to the interface mismatch.

The USDT approve function:

```
function approve(address spender, uint value) public;
```

The IERC20 approve function:

```
function approve(address spender, uint256 amount) external returns (bool);
```

Technical Details

```
File: src/Account.sol

86: IERC20(investmentToken).approve(IAccountFactory(factory).vaultRelayer(),
type(uint256).max);
```

src/Account.sol#L86

Impact

Low.

Recommendation

Use the `forceApprove()` function from the SafeERC20 library.

Developer Response

Fixed in commit 4184dbe454232d262f3dbc1285306bd39ad0bf88.

2. Low - Missing `feeAmount == 0` check

The account doesn't ensure that the order fee is set to 0. While COW protocol does not allow orders with non-zero fees to be posted in the orderbook, a whitelisted solver could still submit orders with nonzero fees at the users' expense.

Technical Details

The function `isValidSignature()` is in charge of ensuring the order executed is valid. It will be called on every order execution by the COW protocol.

Inside this function, it checks the amount of the order, the receiver, and other sensitive parameters of the order, but it doesn't check the `order.feeAmount`, which is expected to be 0.

Because the API rules can be subject to changes and because there are some exceptions (Just-In-Time - JIT - orders), there could be a possibility of a malicious bot submitting orders with non zero fees at the expense of the users.

Additionally, adding a check wouldn't cost a lot of gas as it would be a simple `if` condition on a `memory` variable.

Impact

Low.

Recommendation

Add a check `if (_order.feeAmount != 0) revert()` and revert if the fee differs from 0.

Developer Response

Fixed in commit 29d46be2e86e166afedd9241b5e1055b2d0e80e5.

3. Low - `removeAccount()` function may not execute as expected and can be improved

Technical Details

The function `removeAccount()` can be called to remove a previously created account. It will delete the account from the `accounts` storage array, reorder it, and then reduce the `accountCount` by one.

However, multiple elements in this function are useless or may lead to unexpected results:

- The function loops over the `accounts.length` and not `accountCount`, which means it will keep looping even if the accounts left to loop through are `address(0)`. One result is that if the function is called with `address(0)`, it will think there was indeed an account removed and will lower the `accountCount` by one even though no account was removed. On line 323, if we find the account to be removed but it's the last one in the array (`i = 7`), then it will try to read an unallocated index of the array, which will result in an `out of bound` revert.
- The check on line 329 will never be reached since `j < _accounts.length` and `_accounts.length == MAX_ACCOUNTS`, so `j` can never be equal to `MAX_ACCOUNTS`.
- When the account is found inside its `status` is set to `false` but after the loop the function calls `_setAccountStatus()` on line 343 which will set the account `status` to `false` again.

Impact

Low.

Recommendation

Consider fixing several issues with this function.

Developer Response

Fixed in commit 87af961135edea1e9e29279e14ef661c726fe12d.

Gas Saving Findings

1. Gas - `onlyCoordinator()` modifier can be inlined

The `onlyCoordinator()` modifier is only used once, it can be inlined to reduce gas usage.

Technical Details

```
File: Account.sol

65:     modifier onlyCoordinator() {
66:         if (msg.sender != coordinator) revert AccountError.OnlyCoordinator();
67:         _;
68:         /// Continue executing the function code here
69:     }

110:     function withdrawInvestment(uint256 _amount) external onlyCoordinator {
111:         IERC20(investmentToken).safeTransfer(owner(), _amount);
112:     }
```

Account.sol#L65 Account.sol#L110

Impact

Gas savings.

Recommendation

Inline the modifier.

Developer Response

Fixed in commit a107352ad1051569b5a0de3286ce4651559115d3.

2. Gas - Useless `recipientToken == address(0)` check in `rebalanceTrigger()`

Technical Details

The function `rebalanceTrigger()` checks that the `recipient` and `recipientToken` are different than 0 otherwise return early with `address(0)`.

But it is impossible that the `recipientToken` is set to 0 if the `recipient` is different than `address(0)`. This is because the only two functions where they can be set are `setRecipient()` and `_initialize()` and both have a check that makes sure that if the `recipient` is set then the `recipientToken` has to be set.

Even though this function is usually called off-chain, consider simplifying the check for better readability and gas efficiency.

Impact

Gas.

Recommendation

Consider only doing `if (recipient == address(0)) return address(0);`.

Developer Response

Fixed in commit 12da75a21b4a6e8f2fdcc159057b1c3061d583ba.

3. Gas - State variables only set in the constructor should be declared `immutable`

The following variables are only set in the constructor.

Technical Details

```
File: src/IntermediaryDeployer.sol

43: coordinatorFactory = ICoordinatorFactory(_coordinatorFactory);

44: accountFactory = IAccountFactory(_accountFactory);
```

IntermediaryDeployer.sol#L43, IntermediaryDeployer.sol#L44

Impact

Gas savings.

Recommendation

Use the immutable function modifier.

Developer Response

Fixed in commit d6419111dad3344d772e0d47b3cfe19d2da5416a.

4. Gas - Not needed `address(0)` check in `_buildQueue()`

Technical Details

The function `_buildQueue()` put the accounts passed at the beginning of the array then add the missing ones at the end. During the function, it checks that the accounts passed as parameters are not `address(0)`. This function is always followed by a call to `checkAccount()` which also checks that no account is equal to `address(0)`.

This makes the check inside `_buildQueue()` no needed as it will be checked twice.

Impact

Gas.

Recommendation

Consider removing the `address(0)` check on line 255.

Developer Response

Fixed in commit 70419589fc4693211b4c51c8897888c14435f3b7.

5. Gas - Reduce storage access throughout the contracts

Technical Details

Throughout the code, storage variables are accessed multiple times. While the contracts will be deployed on Gnosis Chain and various functions will be called off-chain, it could be worth limiting storage access to reduce the RPC load and gas price when the function is called on-chain.

- In `initialize()` the `investmentToken` and `factory` are read from storage instead of using the function parameters.
- In `getAccounts()`, `getAccountInfo()`, `getAccountTokens()` and `getAccountAssets()` the storage variable `accountCount` is read twice.
- In `amountToRebalance()` the storage variables `threshold`, `recipientToken` and `recipient` are read twice.
- In `rebalanceTrigger()` the storage variables `recipient` and `recipientToken` are read twice.
- In `removeAccount()` the storage array `accounts` is read multiple time in a loop and `accountCount` is read twice.
- In `sweep()` the storage variable `owner` will be read multiple times throughout the loop.

Impact

Gas.

Recommendation

Consider reducing storage access.

Developer Response

Partially fixed in commit 8cfb864e42ce10bf61af044c8ddb252c6b76fe42.

Informational Findings

1. Informational - Withdraw and sweep functions may leave dust for rebasing tokens

Technical Details

The `withdraw()` and `sweep()` functions in the `Coordinator` contract currently accept specific amounts to be transferred:

```
File: Coordinator.sol
372:     function withdraw(address[] memory _accounts, uint256[] memory _amounts)
external onlyOwner {
373:         if (_accounts.length != _amounts.length) {
374:             revert CoordinatorErrors.InvalidWithdrawArguments();
375:         }
376:
377:         _checkAccounts(_accounts);
378:         for (uint256 i = 0; i < _accounts.length; i++) {
379:             IAccount(_accounts[i]).withdrawInvestment(_amounts[i]);
380:         }
381:     }
382:
383:     /// @inheritdoc ICoordinator
384:     function sweep(address[] calldata _tokens, uint256[] calldata _amounts)
external onlyOwner {
385:         if (_tokens.length != _amounts.length) {
386:             revert CoordinatorErrors.InvalidWithdrawArguments();
387:         }
388:
389:         for (uint256 i = 0; i < _tokens.length; i++) {
390:             if (_tokens[i] == address(0)) {
391:                 SafeTransfer._safeTransferETH(owner(), _amounts[i]);
392:             } else {
393:                 SafeTransfer._safeTransfer(_tokens[i], owner(), _amounts[i]);
394:             }
395:         }
396:     }
```

Coordinator.sol#L372-L396

Impact

Informational

Recommendation

Use a uint256 max value to transfer a token's entire `balanceOf()`.

Developer Response

Acknowledged.

2. Informational - Unused errors present

Unused error is defined and can be removed.

Technical Details

```
File: src/Account.sol


23: error WrongSellAmount();
```

src/Account.sol#L23

Impact

Informational

Recommendation

Remove the unused error.

Developer Response

Fixed in commit fb699ec000c606bb867847c6462bef473dc80d3a.

3. Informational - Unused import

The identifier is imported but never used within the file.

Technical Details

```
File: src/CoordinatorFactory.sol


6: import {Coordinator} from "./Coordinator.sol";
```

src/CoordinatorFactory.sol#L6

Impact

Informational

Recommendation

Remove unused imports.

Developer Response

Fixed in commit 1e80deb742c981da07c459a14ea39d2e73ccc06b and 9dc082eb4e4f6f9e3f001b4738af95e6c70f0157.

4. Informational - Useless `withAccount` parameter in `CoordinatorFactory`

Technical Details

The function `deployContract()` has a parameter `withAccount` that is used inside a condition. However, inside the condition, the function doesn't do anything useful. It just saves the parameters into memory and then doesn't use them.

Additionally, the function `initialize()` in the `Coordinator` contract will always deploy an account.

Impact

Informational.

Recommendation

Remove the parameter and check.

Developer Response

Fixed in commit 6f0e8e1dc45a4b5f8892240b50074595e8b3c77d.

5. Informational - Non-assembly method available

`assembly { size := extcodesize() }` can be replaced with `uint256 size = address().code.length`

Technical Details

```
File: src/Factory.sol


20: size := extcodesize(account)
```

src/Factory.sol#L20

Impact

Informational.

Recommendation

Reduce code complexity using the solidity version.

Developer Response

Fixed in commit d904cabd17acc16fc10b3a3b8c88a2b3d64aee91.

6. Informational - Add an execute function to `Account` and `Coordinator`

While the `Account` and `Coordinator` contracts have a sweep function, adding an execute function is required to handle more complex scenarios with the tokens.

Technical Details

Add an execute function to both Account and Coordinator contracts to allow the owner to perform more complex actions beyond sweeping tokens.

```
    function execute(
        address _to,
        uint256 value,
        bytes memory _data
    ) external payable onlyOwner returns (bytes memory) {
        (bool success, bytes memory result) = _to.call{value: value}(_data);
        return result;
    }
```

Impact

Informational

Recommendation

Add an execute method controlled by the owner.

Developer Response

Acknowledged.

Final remarks

The Qantura Rebalance smart contracts demonstrate a solid foundation with well-implemented access control mechanisms and effective use of established libraries. The architecture supports decentralized management of accounts and funds, aligning well with blockchain principles. While no critical vulnerabilities were found, there are opportunities for improvement in gas optimization and function robustness.