Aconcagua Audit

June 2024

By CoinFabrik

v202311

Executive Summary

CoinFabrik was asked to audit the contracts for the Aconcagua project.

During this audit we found one critical issue, one high issue and several minor issues. Also, several enhancements were proposed.

All the issues were resolved. All the enhancements were implemented.

Scope

The audited files are from the git repository located at https://github.com/Aconcagua-CTO/Aconcagua-API-CONTRACTS-POLYGON.git. The audit is based on the commit `f2b3912150c242219869d9d8cb1dfd2f04a353a8`. Fixes were checked on commit `b71af797f1e3208c4c2a4a29a8e7c367a0e025e9`.

The scope for this audit includes and is limited to the following files:

- `contracts/ColateralContract2.sol`: Contract that handles exchanges between different tokens.
- `contracts/ColateralProxy.sol`: Proxy contract used to upgrade the `ColateralContract2` contract.
- `contracts/ColateralProxyAdmin.sol`: Admin contract used to trigger upgrades.
- `contracts/IColateralContract2.sol`: Interface definitions for the `ColateralContract2` contract.
- `contracts/IWETH.sol`: Interface definition for the WETH token, with functions to convert to and from ETH.

In order to resolve the issues reported, the following files were added to the repository:

- `contracts/IPriceConsumerV3.sol`: Interface definition for the `PriceConsumerV3` helper contract.
- `contracts/IValidatorContract.sol`: Interface definition for the `ValidatorContract` helper contract.
- `contracts/PriceConsumerV3.sol`: Helper contract that fetches prices from the Chainlink price-feed contracts.
- `contracts/ValidatorContract.sol`: Helper contract that makes several checks on initialization and token swap.

No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

Methodology

CoinFabrik was provided with the source code. Our auditors spent one week auditing the source code provided, which includes understanding the context of use, analyzing the boundaries of the expected behavior of each contract and function, understanding the implementation by the development team (including dependencies beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. Without being limited to them, the audit process included the following analyses.

- Arithmetic errors
- Outdated version of Solidity compiler
- Race conditions
- Reentrancy attacks
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

After delivering a report with our findings, the development team had the opportunity to comment on every finding and fix the issues they considered convenient. Once fixed and/or commented, our team ran a second review process to verify that the changes to the code effectively solve the issues found and do not unintentionally add new ones. This report includes the final status after the second review.

Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below. This table does not include the enhancements we suggest to implement, which are described in a specific section after the security issues.

ID	Title	Severity	Status
CR-01	Credentials in Git Repository	Critical	Resolved

ID	Title	Severity	Status
HI-01	tokenIn and tokenOut not Used to Exchange Tokens	High	Resolved
MI-01	Initialize Validations	Minor	Resolved
MI-02	Roles Granted to Zero Addresses	Minor	Resolved

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. Blocking bugs are also included in this category. They must be fixed **immediately**.
- **High:** These refer to a vulnerability that, if exploited, could have a substantial impact, but requires a more extensive setup or effort compared to critical issues. These pose a significant risk and **demand immediate attention**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of, but might be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit has one of the following statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code, but is a result of an intentional decision. The reported risk is accepted by the development team.
- **Resolved**: Adjusted program implementation to eliminate the risk.

- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.

- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk.

Critical Severity Issues

CR-01 Credentials in Git Repository

Location:
- `.env.rsk`
- `.env.template`

Classification:
- CWE-200: Exposure of Sensitive Information to an Unauthorized Actor[1]

The `.env.rsk` and `.env.template` files, which are committed in the git repository, have several API keys and private keys. Those keys should be considered compromised.

Recommendation

The following actions should be taken as soon as possible:

1. If any of the accounts of its private key has admin rights (or similar) on any contract, these rights need to be removed. This step may include giving those rights to a different account.
2. All the funds managed by these accounts need to be transferred to new accounts. This includes the native token of any blockchain, ERC20 tokens and funds being held by other contracts that can be accessed with these credentials.
3. Revoke all the API keys in the different servers.

After following these steps, the file needs to be removed from the git repository, but it must be taken into account that it will still be in the history of the repository.

Status

Resolved. The files were removed from the repository.

High Severity Issues

HI-01 tokenIn and tokenOut not Used to Exchange Tokens

Location:
- `contracts/ColateralContract2.sol: 179-241`

[1] See https://cwe.mitre.org/data/definitions/200.html.

The `swapExactInputs` function receives an array of `SwapParams` to indicate the wanted token swap operations. `SwapParams` is defined in `contracts/IColateralContract2.sol:34-38` as

```
struct SwapParams {
 ISwapRouter.ExactInputParams params;
 address tokenIn;
 address tokenOut;
}
```

`ISwapRouter.ExactInputParams` is defined in the `ISwapRouter.sol` file of the `@uniswap/v3-periphery` library as

```
struct ExactInputParams {
   bytes path;
   address recipient;
   uint256 deadline;
   uint256 amountIn;
   uint256 amountOutMinimum;
}
```

The `path` parameter is used in the calls in line 208

```
quoter.quoteExactInput(swapParams.params.path, swapParams.params.amountIn)
```

and lines 233-234[2]

```
inputs[0] = abi.encode(swapParams.params.recipient,
swapParams.params.amountIn, swapParams.params.amountOutMinimum,
swapParams.params.path, false);
     try IUniversalRouter(contractAddresses['router']).execute(commands,
inputs, swapParams.params.deadline) {
```

But all the events emitted in lines 210, 212, 236 and 248 use `swapParams.tokenIn` and `swapParams.tokenOut`.

There is no place in the code of the `swapExactInputs` function that ensures that `swapParams.params.path` is consistent with `swapParams.tokenIn` and `swapParams.tokenOut`.

`tokenIn` is also used to transfer the tokens to the router in line 228.

[2] Lines are wrapped to fit in this document.

It also must be noted that the only way to withdraw funds from the contract is by calling either the `withdraw` or `rescue` functions, that only allow to withdraw funds of tokens in the `tokenTable` mapping.

This has several implications:

1. There is no warranty that the events correspond to the exchanges made.
2. A call with the wrong path may leave funds locked in the `ColateralContract2` contract that can only be retrieved by upgrading the code.

Recommendation

1. Change the `SwapParams` struct in the `contracts/IColateralContract2.sol` file so it does not use `ISwapRouter.ExactInputParams`.
2. Add all the fields in `ISwapRouter.ExactInputParams` except for the `path` field to the `SwapParams` struct. While doing so, take note that the recipient field will always be `address(this)`, so it is not necessary[3].
3. Add a new field named `intermediateSteps` that will contain the addresses of the intermediate tokens in the exchange paths.
4. Change the calls in lines 208 and 233 to construct the full path to be used in the token exchange, based on `tokenIn`, `tokenOut` and `intermediateSteps`.

If changing the interface of the `swapExactInputs` is not possible, you have to make sure in the code of the contract that `swapParams.params.path` is consistent with `swapParams.tokenIn` and `swapParams.tokenOut`.

Status

Resolved. The development team added a check to make sure that the `tokenIn` and `tokenOut` fields of each element of the `swapParams` array are consistent with the `params.path` field in the same element. The implementation of this check can be seen in the `validatePath` function defined in `contracts/ValidatorContract.sol:212-228`.

Medium Severity Issues

No issues found.

Minor Severity Issues

MI-01 Initialize Validations

Location:

- `contracts/ColateralContract2.sol: 179-241`

[3] See `contracts/ColateralContract2.sol:192`

There are several problems with the validation of the parameters passed to the `initialize` function in the `ColateralContract2` contract.

The ones we could find were:

1. The token names should always include `'WETH'`, `'WBTC'`, `'USDC'` and `'USDT'`
2. The contract keys should always include `'router'`, `'quoter'`, and `'swapper'`.
3. Lack of zero-address checks on addresses passed via arrays.

Recommendation

Either add the checks or refactor the code so the checks are not needed.

Status

Resolved. The missing checks were added in the new `contracts/ValidatorContract.sol` file.

MI-02 Roles Granted to Zero Addresses

Location:

- `contracts/ColateralContract2.sol: 128-130, 135`

In the `initialize` function of the `ColateralContract2` contract there exists the possibility of giving the zero address the `ACONCAGUA_ROLE` role (lines 128-130) or the `SWAPPER_ROLE` role (line 135).

While this is not an exploitable issue given that the private key for the zero-address is not known and there is no known way to deploy a contract to the zero address, if someone discovers how to to that in the future or the blockchain is altered to signal some different state with `msg.sender == 0x0` it would leave the code open to a critical bug.

This issue is related to MI-01 Initialize Validations.

Recommendation

Only add non-zero addresses to roles.

Status

Resolved. Zero-address checks were added in the new `contracts/ValidatorContract.sol` file.

Enhancements

These items do not represent a security risk. They are best practices that we suggest implementing.

ID	Title	Status
EN-01	No Tests	Implemented
EN-02	No Quoter Required	Implemented
EN-03	Not Needed Instance Variables	Implemented

EN-01 No Tests

No automated tests were provided by the development team for the audited contracts.

Recommendation

Implement the automated tests.

Status

Implemented. Now the tests can be run by executing `npm install`, then `npx hardhat compile` and finally `npm run test`.

EN-02 No Quoter Required

Location:

- `contracts/ColateralContract2.sol: 208, 216`

The `quoter` address is only used in the `swapExactInputs` function logic of the `ColateralContract2` to calculate the `quotedAmountOut`, which in turn is only used in the following check in line 216

```
swapParams.params.amountOutMinimum < (quotedAmountOut * 98) / 100)
```

And also, in the `IQuoter` documentation states that its functions should not be used on-chain because they are too expensive[4].

Recommendation

Calculate the actual amount of `tokenOut` tokens obtained by calculating the difference in balances before and after calling `execute` on line 234 and use this amount to do the check in line 216.

Status

Implemented. A new check was implemented by consulting the price with an oracle instead.

[4] See https://docs.uniswap.org/contracts/v3/reference/periphery/interfaces/IQuoter

EN-03 Not Needed Instance Variables

Location:

- `contracts/ColateralContract2.sol`

The instance variables `_rolesSet`, and `swapRouter` are not needed in the code of the `ColateralContract2` contract. The `_rolesSet` variable never changes so all its uses in the `getRoleCount` and `getRoleByIndex` functions can be resolved in a different manner. The `swapRouter` variable is not used at all.

Recommendation

Remove the instance variables and adjust the code accordingly.

Status

Implemented. The `swapRouter` instance variable and the `getRoleCount` and `getRoleByIndex` functions were removed. The `_rolesSet` variable was not removed to keep the contract's instance variables layout compatible given that it is an upgradeable contract.

Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest to other stakeholders of the project, including users of the audited contracts, token holders or project investors.

Centralization

The code upgrades can be triggered by the admin of the `ColateralProxy` contract. So if this account is compromised all the system is compromised.

Besides that, all the state-changing operations in the `CollateralContract2` contract can only be executed by a role that has a whitelist of approved accounts. It must also be noted that funds can be retrieved from the contract by 2 different roles (`LENDER_LIQ_ROLE` and `RESCUER_ROLE`) by two different functions (`withdraw` and `rescue`).

Upgrades

The `ColateralContract2` contract is intended to be upgraded via the `ColateralProxy` contract. The `ColateralProxy` contract itself is expected to be administered by an instance of the `ColateralProxyAdmin` contract. So in effect, the owner of the

`ColateralProxyAdmin` is the account responsible for upgrading the code. This schema is consistent with the suggestions on the OpenZeppelin documentation[5].

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

ColateralContract2

All the roles in this contract are handled by the `AccessControl` API provided by the OpenZeppelin library.

ACONCAGUA_ROLE

An address with this role can:

1. Grant and revoke the ACONCAGUA_ROLE role for any account using the `AccessControl` API.
2. Grant and revoke the SWAPPER_ROLE role for any account using the `AccessControl` API.
3. Set the address where funds are transferred to when the `withdraw` function is invoked via the `setWithdrawWalletAddress` function.
4. Set the address where funds are transferred to when the `rescue` function is invoked via the `setRescueWalletAddress` function.

After initialization, the 3 addresses passed to the `initialize` function in the `_aconcagua` array are the addresses with this role.

LENDER_LIQ_ROLE

An address with this role can:

1. Grant and revoke the LENDER_LIQ_ROLE role for any account using the `AccessControl` API.
2. Transfer any amount of tokens registered in the contract to the `withdrawWalletAddress` via the `withdraw` function.

After initialization, the addresses passed to the `initialize` function in the `_firstLenderLiq` and `_secondLenderLiq` parameters are the addresses with this role.

RESCUER_ROLE

An address with this role can:

1. Grant and revoke the RESCUER_ROLE role for any account using the `AccessControl` API.

[5] See https://docs.openzeppelin.com/contracts/4.x/api/proxy#TransparentUpgradeableProxy.

2. Transfer any amount of tokens registered in the contract to the `rescueWalletAddress` via the `rescue` function.

After initialization, the addresses passed to the `initialize` function in the `_firstLenderLiq` and `_secondLenderLiq` parameters are the addresses with this role.

SWAPPER_ROLE

An address with this role can swap tokens using the `swapExactInputs` function.

After initialization the address in `contractAddresses['swapper']` is the only address with this role.

ColateralProxy

Admin

The initial address with the admin role is passed as a parameter in the constructor. The only functionality available is provided by the `TransparentUpgradeableProxy` base contract implemented by the OpenZeppelin library.

It is expected that this address will be set pointing to a `ColateralProxyAdmin` deployed contract.

ColateralProxyAdmin

Owner

The initial address with the owner role is passed as a parameter in the constructor. The only functionality available is provided by the `ProxyAdmin` base contract implemented by the OpenZeppelin library.

Funds Outflow Analysis

The only analyzed contract that transfers funds away is `CollateralContract2`.

The `swapExactInputs` function transfers funds away to the router on line 228.

The `withdraw` function transfers funds away to the `withdrawWalletAddress` on line 257.

The `rescue` function transfers funds away to the `rescueWalletAddress` on line 281.

The `receive` function deposits the received ether in the WETH contract, if the contract deployed to the RSK network, on line 300.

Audit Assumptions

For the context of this audit we assume that the quoter used by the `ColateralContract2` contract, and set in the initialize function, corresponds to the smart contract with the same interface implemented by `UniswapV3`. In particular, but not exclusively, we assume that no reentrancy attacks can be performed through it[6].

We also assume that the `contractAddresses['router']` address points to the contract in the `UniswapV3` project that implements the `IUniversalRouter` interface. In particular, but not exclusively, we assume that no reentrancy attacks can be performed through it[7].

When we checked the corrections, we assumed that the `validator` points to a `ValidatorContract` contract and that its `priceConsumerV3` points to a `PriceConsumerV3` contract.

Changelog

- 2024-06-28 – Initial report based on commit `f2b3912150c242219869d9d8cb1dfd2f04a353a8`.
- 2024-09-09 – Check fixes based on commit `b71af797f1e3208c4c2a4a29a8e7c367a0e025e9`

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the Aconcagua project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.

[6] See `contracts/ColateralContract2.sol:208`.
[7] See `contracts/ColateralContract2.sol:233`.