Aconcagua Audit

Collateral Contract

September 2023

By CoinFabrik

v202307

Executive Summary

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

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

Scope

The audited files are from the git repository located at https://github.com/aconcagua-finance/Aconcagua-API-CONTRACTS-POLYGON. The audit is based on the commit `cf54a197c380cc667a19724b902295e7574606c0`. Fixes reviewed on commit `119c0d88afba89b3bbe73c090f14cea8b4b101ee`.

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

- `contracts/ColateralContract.sol`: Contract for managing collateral.
- `contracts/IColateralContract.sol`: Interface for ColateralContract.
- `contracts/ColateralProxy.sol`: Upgradeable Proxy for ColateralContract.
- `contracts/ColateralProxyAdmin.sol`: Admin for Upgradeable Proxy.

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.

Contracts Descriptions

- `ColateralContract.sol`:
    1. `initialize()`: Initializes the contract with initial token addresses, roles, wallet addresses, and other parameters. Only callable once due to the `initializer` modifier.
    2. `version()`: Returns the version of the contract as a string, which is "`1.0.0`".
    3. `setWithdrawWalletAddress()`: Allows setting a new withdrawal wallet address. Can only be called by an account with the `ACONCAGUA_ROLE`.
    4. `setRescueWalletAddress()`: Allows setting a new rescue wallet address. Can only be called by an account with the `ACONCAGUA_ROLE`.
    5. `setStartTimePeriod()`: Sets the start time of the period and adjusts the end time accordingly. Restricted to `ACONCAGUA_ROLE`.
    6. `setTokenAddress()`: Updates the address of a specific token and emits a `TokenAddressChange` event. Restricted to `ACONCAGUA_ROLE`.
    7. `setWithdrawalLimitPerPeriod()`: Sets the withdrawal limit for a given token per period and emits a `WithdrawalLimitChange` event. Restricted to `ACONCAGUA_ROLE`.

8. `swapExactInputs()`: Performs a series of token swaps using a Uniswap-equivalent router. Restricted to `SWAPPER_ROLE`.
9. `withdraw()`: Allows withdrawal of a specified amount of a token, checking against period limits. Restricted to `LENDER_LIQ_ROLE`.
10. `balanceOf()`: Returns the balance of a specific token held by the contract.
11. `getBalances()`: Returns the balances of `ETH`, `USDC`, `USDT`, `WBTC`, and `WETH` held by the contract.
12. `rescue()`: Transfers a specified amount of a token to the rescue wallet address. Restricted to `RESCUER_ROLE`.
13. `getRoleCount()`: Returns the total number of unique roles in the contract.
14. `getRoleByIndex()`: Returns a role by its index in the enumerable set of roles.
15. `_checkPeriodLimits()`: Internal function to update the withdrawal limits for tokens if the current time has exceeded the end of the period.

The contract also includes various state variables, mappings, and constants to manage tokens, roles, withdrawal limits, and period timings.

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

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	Incorrect Usage of Initializer Modifier in Upgradable Contract	Critical	Resolved
ME-01	Denial Of Service Because of Invalid Token Addresses	Medium	Resolved
ME-02	Role Revocation	Medium	Mitigated
MI-01	Floating Solidity Pragma	Minor	Resolved
MI-02	Manipulation of Start Time Period by Aconcagua Role	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**.
- **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 Incorrect Usage of Initializer Modifier in Upgradable Contract

Location:

- `contracts/ColateralContract.sol:53`

The `CollateralContract` utilizes an `initializer` modifier in both the constructor and an `initialize()` function. The `initializer` modifier is designed to ensure that initialization logic in upgradeable contracts is only run once. In the current implementation, the contract is initialized at the time of deployment due to the presence of the `initializer` modifier in the constructor. This means that any subsequent calls to the `initialize()` function, or any other function using the `initializer` modifier, will be reverted, leading to a loss of functionality and issues in managing the contract state.

Steps to Replicate

1. Deploy the `CollateralContract`.
2. After deployment, attempt to call the `initialize()` function.
3. Observe that the transaction is reverted.

Recommendation

Remove the `initializer` modifier from the constructor and rely solely on the `initialize()` function for initializing the contract. This adjustment ensures that the contract is not automatically initialized at the time of deployment, allowing for appropriate initialization when required. Local testing (as proposed in EN-05) and testnet deployment are advised in order to avoid proxy issues on production.

When applying this fix, the implementation contract should not be left uninitialized. Instead, `_disableInitializers()` needs to be placed in the constructor body in order to avoid an attacker taking ownership of that contract.

Status
Resolved. Fixed according to the recommendation.

Medium Severity Issues

ME-01 Denial Of Service Because of Invalid Token Addresses

Location:
- `contracts/ColateralContract.sol:155`

Classification:
- SWC-113: DoS with Failed Call[1]
- CWE-703: Improper Check or Handling of Exceptional Conditions[2]

The `setTokenAddress()` function in the examined smart contract allows an account with the `ACONCAGUA_ROLE` role to set the address associated with a specific token symbol. The possibility of assigning an invalid or malicious address to a token symbol poses a notable risk to the integrity of contract operations.

While direct loss of funds is not an immediate concern due to the allowance mechanism in place for token transfers, the integrity of the system's operations can be severely compromised. Setting an invalid or non-compliant address can disrupt the functioning of any interactions with the associated token, potentially leading to a denial of service for users and other contract roles relying on this token.

Recommendation
Define critical token addresses as immutable or implement a time-lock mechanism. This approach will either prevent unintended alterations to token addresses or allow ample time for irregularities to be detected and addressed, thereby enhancing the security and integrity of the contract's operations.

Status
Resolved. `setTokenAddress()` function removed and token addresses are set at initialization. The values cannot be changed.

[1] https://swcregistry.io/docs/SWC-113/
[2] https://cwe.mitre.org/data/definitions/703.html

ME-02 Role Revocation

Location:

- `contracts/ColateralContract.sol:111-114`

Classification:

- CWE-284: Improper Access Control[3]

The current implementation of the smart contract allows any member of a role group to revoke role membership of others within the same group. This design poses a risk as it enables any admin to unilaterally grant or revoke access to significant functionalities, potentially leading to unauthorized access or denial of service.

Recommendation

Implement additional safeguards such as multi-signature requirements or time locks for sensitive role assignments and revocations. Otherwise, evaluate the necessity of allowing every member within a role group to modify membership, and restrict this ability if it is not essential for the operation of the contract.

Status

Mitigated. These roles will be assigned to multi signature wallets. Therefore, the likelihood is lower.

Minor Severity Issues

MI-01 Floating Solidity Pragma

Location:

- `contracts/ColateralContract.sol`
- `contracts/IColateralContract.sol`
- `contracts/ColateralProxy.sol`
- `contracts/ColateralProxyAdmin.sol`

Classification:

- SWC-103: Floating Pragma[4]
- CWE-664: Improper Control of a Resource Through its Lifetime[5]

The smart contracts use a floating Solidity pragma. This implies that these contracts are not bound to a specific compiler version. Although this can be advantageous for flexibility and compatibility, especially for libraries, it poses potential risks for contracts that are deployed in live environments.

[3] https://cwe.mitre.org/data/definitions/284.html
[4] https://swcregistry.io/docs/SWC-103/
[5] https://cwe.mitre.org/data/definitions/664.html

If contracts get deployed using an unintended compiler version, it might inadvertently introduce bugs or vulnerabilities that can negatively affect the system's integrity. It is paramount that contracts be deployed with the same compiler version and flags they've been rigorously tested with to ensure predictable and secure behavior.

Recommendation
Lock the pragma version in the smart contracts. This ensures that the contracts are always compiled with the intended compiler version.

Status
Resolved. Fixed according to the recommendation.

MI-02 Manipulation of Start Time Period by Aconcagua Role

Location:

- `contracts/ColateralContract.sol:148`

The `setStartTimePeriod()` function is capable of being called by an admin (with `Aconcagua` role) at any moment, even after the start time of the period has been initiated. While this does not lead to an issue with the contract itself, it does introduce uncertainties around the trustworthiness of period values. Parties interacting with the contract must be aware that the Aconcagua role has the ability to manipulate the start and end times of periods, potentially affecting their interactions or strategies.

Recommendation
`setStartTimePeriod()` function should not be able to set a new start time in the past.

Status
Resolved. Periods were removed from the codebase.

Enhancements

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

ID	Title	Status
EN-01	Typo in Contract Names	Not implemented
EN-02	Utilization of Proper Initialization Functions	Not implemented
EN-03	Unnecessary Type Casting	Not implemented
EN-04	Missing Pausing Functionality	Not implemented
EN-05	Implementation of Automated Testing	Not implemented

EN-01 Typo in Contract Names

Location:

- `contracts/ColateralContract.sol`
- `contracts/IColateralContract.sol`
- `contracts/ColateralProxy.sol`
- `contracts/ColateralProxyAdmin.sol`

File, contract, and interface names have a typo in the word “collateral”, which is currently written as “colateral”.

Recommendation

Rename files and contracts.

Status

Not implemented.

EN-02 Utilization of Proper Initialization Functions

Location:

- `contracts/ColateralContract.sol`

The examined smart contract employs `__AccessControl_init_unchained()` and `__ReentrancyGuard_init_unchained()` for the initialization of Access Control and Reentrancy Guard. However, utilizing the `__init_unchained()` variants directly can be unconventional and prone to human error, especially if additional initializers are added in the future.

The more conventional and recommended approach is to use the `__init()` function, which internally calls the `__init_unchained()` function. This ensures a more structured initialization sequence and is less error-prone, particularly when dealing with multiple initializers.

Recommendation

Modify the initialization sequence to utilize the `__init()` functions for both Access Control and Reentrancy Guard. This change aligns with best practices and provides a safer, more structured, and less error-prone initialization process.

```
Unset
// Current Initialization
__AccessControl_init_unchained();
__ReentrancyGuard_init_unchained();

// Recommended Initialization
__AccessControl_init();
__ReentrancyGuard_init();
```

Status

Not implemented.

EN-03 Unnecessary Type Casting

Location:

- `contracts/ColateralContract.sol:180`

The contract contains an unnecessary type casting where `tokenIn` is explicitly cast to an address, even though `tokenIn` is already of type address.

Recommendation

Remove the unnecessary type casting of `tokenIn` since it is already an address.

Status

Not implemented.

EN-04 Missing Pausing Functionality

Location:

- `contracts/ColateralContract.sol:26`

The `ColateralContract` currently defines a `PAUSER_ROLE`, but it is not associated with any pausing functionality in the contract. This role should either be linked to the actual pausing of contract interactions for security purposes or be removed if deemed unnecessary.

Recommendation

Evaluate the necessity of the `PAUSER_ROLE` in the `ColateralContract`. If essential, implement associated pausing functionality; if not, remove the unused role to maintain code clarity and cleanliness.

Status

Not implemented.

EN-05 Implementation of Automated Testing

The project currently lacks a comprehensive automated testing system, potentially allowing errors and vulnerabilities to go undetected. Implementing a diverse range of automated tests will enhance early issue detection, improve system reliability, and ensure consistent functionality verification.

Recommendation

Implement a comprehensive automated testing framework incorporating unit, integration, and end-to-end tests. Integrate this framework into the CI/CD pipeline for timely issue detection and resolution, thereby improving the overall quality and reliability of the system.

Status

Not implemented.

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 provided `ColateralContract` has various roles such as `ACONCAGUA_ROLE`, `LENDER_LIQ_ROLE`, and others, which are assigned specific permissions, thereby restricting key functions like token swapping, withdrawals, and parameter modifications to designated addresses. These roles have the ability to control fund management, set parameters, and perform rescue operations.

Upgrades

The given smart contract employs OpenZeppelin's upgradeable contracts library, which is the upgrade mechanism. Initializer functions are used instead of constructors to preserve the contract's storage layout across upgrades, thereby allowing modification and addition of functionalities.

Changelog

- 2023-09-26 – Initial report based on commit `cf54a197c380cc667a19724b902295e7574606c0`.
- 2023-11-06 – Final report based on commit `119c0d88afba89b3bbe73c090f14cea8b4b101ee`.

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.