Cryptonics

Anchor Protocol Smart Contracts - Audit Report

March 8, 2021

This audit has been performed by

Philip Stanislaus and **Stefan Beyer**

Cryptonics Consulting S.L.
Ramiro de Maeztu 7
46022 Valencia
SPAIN

https://cryptonics.consulting/

Table of Contents

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHORS AND THEIR EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT IS NOT A SECURITY WARRANTY, INVESTMENT ADVICE, OR AN ENDORSEMENT OF THE CLIENT OR ITS PRODUCTS. THIS AUDIT DOES NOT PROVIDE A SECURITY OR CORRECTNESS GUARANTEE OF THE AUDITED SOFTWARE.

Introduction

Purpose of this Report

Cryptonics Consulting has been engaged by Terraform Labs to perform a security audit of the Anchor Protocol smart contracts. This current audit report covers the implementation of the CosmWasm smart contract and the Ethereum smart contracts of the Anchor protocol architecture.

The objectives of the audit are as follows:

1. Determine the correct functioning of the system, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the code submitted in the following GitHub repositories:

https://github.com/Anchor-Protocol/anchor-eth-contracts/tree/development

Commit no: `91f0c0326c510da40853f3ae1d1349ba8d224b7f`

https://github.com/Anchor-Protocol/anchor-bAsset-contracts

Commit no: `75f0e9dd856858679a00d7ae090975caf6d1d4c9`

https://github.com/Anchor-Protocol/money-market-contracts

Commit no: `04b685a80c68548b03ce15c62ff719032950d99a`

Methodology

The audit has been performed by a mixed team of smart contract and full-stack auditors.

The following steps were performed:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under- / overflow issues
    c. Key management vulnerabilities
    d. Permissioning issues
    e. Logic errors
4. Report preparation

The results were then discussed between the auditors in a consensus meeting and integrated into this joint report.

Functionality Overview

The submitted code implements the smart contracts for the Anchor protocol, a DeFi savings protocol on the Terra blockchain.

The protocol consists of a liquidity-pool based lending protocol, that allows liquidity providers to earn yields on a number of assets.

How to read this Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged,** or **Resolved**. Informational notes do not have a status, since we consider them optional recommendations.

Note, that audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentatio**n, and **test coverage**. We include a table with these criteria for each module, in the corresponding findings section.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Summary of Findings

The Anchor smart contracts were found to contain 2 critical issues, 2 major issues, 7 minor issues and 6 informational notes:

No	Description	Severity	Status
Ethereum Smart Contracts			
1	`AnchorEthFactory.sol`: Storage Initialization breaks Upgradability Pattern	Critical	Resolved
2	`AnchorEthFactory.sol`: Migration to new contract will work only once and fail, if too many `AnchorAccount` contracts deployed	Major	Resolved
CosmWasm anchor-bAsset-contracts Smart Contracts			
3	Coins other than the configured stable coin might be lost	Minor	Acknowledged
4	`query_get_finished_amount` may include not yet released amounts from undelegation	Minor	Acknowledged
5	`handle_deregister_validator` can panic if the last validator is deregistered	Minor	Resolved
6	`handle_deregister_validator` will revert if `total_balance` is zero	Minor	Resolved
7	Changing the underlying coin denom of the hub will result in unbondable tokens	Minor	Resolved
8	Overflow checks not set for profile release in all packages	Informational	Resolved
9	`history.withdraw_rate` is used inconsistently	Informational	Resolved
10	Unnecessary duplicate `slashing` function call	Informational	Resolved

CosmWasm money-market-contracts Smart Contracts			
11	`sub` and `add` functions for tokens don't handle multiple tokens with the same address properly	Critical	Resolved
12	Undercollateralized loans cannot be liquidated	Major	Resolved
13	Coins other than the configured stable coin might be lost	Minor	Acknowledged
14	Interest not computed before reserve factor is updated	Minor	Resolved
15	After market contract initialization, anyone can set the overseer contract	Informational	Acknowledged
16	Interest not compounding between messages	Informational	Acknowledged
17	Overflow checks not set for profile release in `packages/moneymarket/Cargo.toml`	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium	-
Level of Documentation	Medium-high	-
Test Coverage	Medium-high	-

Detailed Findings

Ethereum Smart Contracts

1. `AnchorEthFactory.sol`: Storage Initialization breaks Upgradability Pattern

Severity: Critical

The function `migrate()` suggests that this contract should be upgradable and deployed through Open Zeppelin's implementation of the proxy pattern. However, this framework requires storage not to be initialized through static assignment or through the constructor. Instead, the contract should inherit from OZ's `Initializable` contract and implement an `initialize()` function to set global storage variables in the proxy's context. The current implementation fails to initialize the `terrausd` and `achorust` variables.

Recommendation

Refactor the contract according to the method described in:
https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable

Status: Resolved

2. `AnchorEthFactory.sol`: Migration to new contract will work only once and fail, if too many `AnchorAccount` contracts deployed

Severity: Major

The function `migrate()` allows the privileged owner of the factory contract to migrate ownership of all `AnchorAccount` wallet contracts to a new address. This is meant to be used if the smart contract is upgraded using Open Zeppelin's implementation of the proxy pattern. However, the function introduces two important limitations:

- Migration will only work once, since the `isMigrated` variable will already be true after the first migration. Note, that in the proxy upgradability pattern, the context of storage is the actual proxy contract, meaning that the value of `isMigrated` will be maintained.
- The `migrate()` function iterates over the `ContractsList` array. This means that once this array grows too big the function invocation will hit the block gas limit, causing the transaction to revert.

Recommendation

Consider revising the migration process not to rely on the `isMigrated` variable. Furthermore, it is advisable to refactor the migration process to allow migrating wallets in batches (for example by 100 contracts at a time by index range).

Status: Resolved

CosmWasm `anchor-bAsset-contracts` Smart Contracts

3. Coins other than the configured stable coin might be lost

Severity: Minor

In several places in the codebase, sent funds are filtered by the configured stable denomination, but any other coins sent are not returned to the sender. That leaves those other coins frozen to the contract. Those places are: `contracts/anchor_basset_hub/src/bond.rs:33`, `contracts/anchor_basset_hub/src/contract.rs:38`

Recommendation

We recommend either returning unused coins or reverting the transaction if unexpected coins are sent.

Status: Acknowledged

4. `query_get_finished_amount` may include not yet released amounts from undelegation

Severity: Minor

In `query_get_finished_amount` in `contracts/anchor_basset_hub/src/state.rs:219`, no check is made whether the amount has been released yet. If the target block is after the latest released block, the returned amount will be bigger than what can actually be withdrawn.

Recommendation

As done in `get_finished_amount`, we recommend adding a check for the `released` status of the batch.

Status: Acknowledged

`query_get_finished_amount` accepts a `block_time` argument that might be after currently released batches, the logic would need to calculate the `released` status of all batches at that time.

5. `handle_deregister_validator` can panic if the last validator is deregistered

Severity: Minor

In `contracts/anchor_basset_hub/src/config.rs:258`, no check is made whether the validator set is not empty, in which case `unwrap` leads to a panic.

Recommendation

We recommend to either prevent deregistering the last validator before line `258` or handling delegated funds from that last validator differently.

Status: Resolved

6. `handle_deregister_validator` will revert if `total_balance` is zero

Severity: Minor

In `contracts/anchor_basset_reward/src/global.rs:65`, an error is returned if `total_balance` is zero. That leads `handle_deregister_validator` to revert, implying that validators cannot be deregistered until the balance is getting positive.

Recommendation

We recommend to allow deregistering even if the total balance is zero.

Status: Resolved

7. Changing the underlying coin denom of the hub will result in unbondable tokens

Severity: Minor

The permissioned function `chandle_update_params` in `contracts/anchor_basset_hub/src/config.rs:36`, allows a change of the `underlying_coin_denom`. Since that variable is used during unbonding of tokens, a change of the value will leave any bonded tokens locked without any possibility to unbond.

Recommendation

This issue is relatively minor, since the `handle_update_params` call is permissioned. However, since access to bonded tokens could be lost, we still recommend implementing either a check that fails the function call if there are still bonded tokens or an automatic unbonding.

Status: Resolved

8. Overflow checks not set for profile release in all packages

Severity: Informational

Currently, only the workspace `cargo.toml` enables `overflow-checks` for the release profile, while the individual packages have not enabled release overflow checks.

Recommendation

While this check is implicitly applied to all packages from the workspace `cargo.toml`, we recommend also explicitly enabling overflow checks in every individual package. That helps when the project is refactored to prevent unintended consequences. This recommendation is already implemented in `money-market-contracts`, with one exception (see below).

Status: Resolved

9. `history.withdraw_rate` is used inconsistently

Severity: Informational

`history.withdraw_rate` is used for two purposes: a) to store the historical exchange rate, which is used to convert bAssets to assets and b) to store the slashing rate, which is used to convert burned funds to unbonded funds. While this might make sense for efficiency, it will be confusing to users and the unbonding history loses information about the exchange rate after releasing the unbonded funds.

Recommendation

We recommend separating the exchange rate and slashing rate into distinct fields.

Status: Resolved

A new field applied_exchange_rate has been added to the state, which will allow external users to query the used exchange rate before the slashing.

10. Unnecessary duplicate `slashing` function call

Severity: Informational

In `contracts/anchor_basset_hub/src/bond.rs:41-42`, the function `slashing` is called twice. This is not necessary.

Recommendation

We recommend removing the duplicate function call.

Status: Resolved

CosmWasm `money-market-contracts` Smart Contracts

11. `sub` and `add` functions for tokens don't handle multiple tokens with the same address properly

Severity: Critical

The `add` and `sub` functions defined in `packages/moneymarket/src/tokens.rs:27` and `61` do not handle multiple tokens with the same address in either `self` or the `tokens` argument properly.

In the `add` function, if there are more entries with the same addresses in `self` or `tokens`, the additional entries are skipped and the sum will be too small.

In the `sub` function, if there are more entries with the same addresses in `tokens`, the additional entries will be skipped, i. e. too little is subtracted. If there are more entries with the same addresses in `self`, the additional entries will be skipped, which is not an issue.

An example where this can be exploited is in `unlock_collateral` in `contracts/overseer/src/collateral.rs:78`. An attacker can exploit this issue by submitting multiple entries with the same address in `collaterals_human`. The code only subtracts the value of the first of those entries from the stored collaterals and stores those collaterals again, but proceeds to unlock all requested collaterals from `collaterals_human`.

Likewise, `lock_collateral` in `contracts/overseer/src/collateral.rs:21` is affected by this issue. In this case, collateral will be locked but not added to the user's collateral store, so the user can never unlock those collaterals again.

Recommendation

We recommend adding a check to the `add` and `sub` functions to return an error if multiple entries with the same address are encountered, or alternatively apply the calculation such that the result considers all entries.

Status: Resolved

12. Undercollateralized loans cannot be liquidated

Severity: Major

During the `query_liquidation_amount` function in `contracts/liquidation/src/contract.rs:262`, an error is returned if the expected repay amount from a liquidation is not high enough to fully cover the borrowed amount. While that might make sense in cases where prices temporarily dip, it implies that loans might not even be partially liquidated and might lose even more value in a market collapse.

Recommendation

To prevent a black swan event where lenders lose all their value, we recommend implementing partial liquidation or even a default of loans.

Status: Resolved

13. Coins other than the configured stable coin might be lost

Severity: Minor

In several places in the codebase, sent funds are filtered by the configured stable denomination, but any other coins sent are not returned to the sender. That leaves those other coins frozen to the contract. Those places are:
`contracts/market/src/contract.rs:31`,
`contracts/market/src/borrow.rs:104`,
`contracts/market/src/deposit.rs:20`,
`contracts/liquidation/src/bid.rs:35`

Recommendation

We recommend either returning unused coins or reverting the transaction if unexpected coins are sent.

Status: Acknowledged

14. Interest not computed before reserve factor is updated

Severity: Minor

Before the reserve factor is updated in `contracts/market/src/contract.rs:209`, `compute_interest` is not called. That will lead to the new reserve factor being applied to the interest accumulated until the time of execution, rather than just to future interest.

Recommendation

As done in line `202` when updating the interest model, we recommend calling `compute_interest` before updating the reserve factor.

Status: Resolved

15. After market contract initialization, anyone can set the overseer contract

Severity: Informational

During the `init` function of the market contract in `contracts/market/src/contract.rs:26`, the `overseer_contract` variable is assigned to `CanonicalAddr::default()`. After that initialization, anyone can send the `RegisterOverseer` message, since there is no permission check in the `register_overseer_contract` handler.

Recommendation

Since there is an initial deposit at stake, we recommend adding permissioning to the `register_overseer_contract` handler for the `RegisterOverseer` message.

Status: Acknowledged

Contract deployment is done in a script and can be verified after deployment.

16. Interest not compounding between messages

Severity: Informational

`compute_interest_raw` uses a simple linear interest formula rather than compounding the interest accumulated between function calls. That opens a way for rational lenders to increase their interest rate by regularly sending a borrow/repay/deposit/redeem message which updates the interest rate, which in turn triggers compounding more often.

Recommendation

While not a security concern, the current implementation leads to an undefined annualized percentage yield (APY). That APY depends on the number of calls to the `compute_interest_raw` function. While this linear interest model might be a deliberate business decision, a compounding implementation could be preferable. For reference, see Maker's implementation and docs.

Status: Acknowledged

The current implementation is a conscious design decision by the Anchor team to simplify the design.

17. Overflow checks not set for profile release in `packages/moneymarket/Cargo.toml`

Severity: Informational

While set in all other packages, `packages/moneymarket/Cargo.toml` does not enable `overflow-checks` for the release profile.

Recommendation

While this check is implicitly applied to all packages from the workspace `cargo.toml`, we recommend also explicitly enabling overflow checks in every individual package. That helps when the project is refactored to prevent unintended consequences.

Status: Resolved