Audit Report

Edge Protocol

v1.0

April 6, 2022

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Edge Protocol to perform a security audit of Edge protocol money market smart contracts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, 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 following GitHub repository:

https://github.com/edge-protocol/edge-contracts

Commit hash: `aa2310215901f45a61b512968e1e487081eee122`

Methodology

The audit has been performed in the following steps:

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
4. Report preparation

Functionality Overview

Edge protocol is a money-market-as-a-service (MMaaS) provider in the Terra ecosystem. It provides a factory that allows anyone to create a pool that offers functionality to create a market for different assets. Assets deposited in a created pool can be borrowed and liquidity providers can earn interest on their position. Prices are retrieved using the TeFi oracle.

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**.

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 documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low 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

No	Description	Severity	Status
1	Insurance funds not used by protocol during credit events	Critical	Resolved
2	Protocol mint fees locked in factory contract for perpetuity	Major	Resolved
3	No validation of collateral factor and liquidation incentives could lead to loss of funds	Major	Resolved
4	Unbounded size of `MARKET_WHITELIST` leads to increasing gas consumption with number of markets, eventually causing gas depletion	Major	Acknowledged
5	No validation of admin and insurance fee rates may lead to underflows	Minor	Resolved
6	No validation of protocol fee rate during config updates	Minor	Resolved
7	Increase of Terra taxes could cause positions to default without an ability to repay or liquidate	Minor	Acknowledged
8	Recurring use of magic numbers throughout codebase	Informational	Resolved
9	Double slope IRM misses kink validation	Informational	Resolved
10	Triple slope IRM misses kink validation	Informational	Resolved
11	Overflow checks not enabled for release profile	Informational	Resolved
12	Unreachable error for eToken limit	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	-
Level of documentation	Medium	The high level business logic is covered by the provided documentation, while the code is could benefit from more low level comments.
Test coverage	Medium-High	-

Detailed Findings

1. Insurance funds not used by protocol during credit events

Severity: Critical

Insurance fees are continually accumulated by the protocol in file `contracts/edge-pool/src/borrow.rs` in lines `500` and `446`. Insurance fees are intended to be used when there are underwater liquidations. However, throughout the protocol, these funds are never utilized. For example, there is no usage of them in the `withdraw` function in `contracts/edge-pool/src/underlying.rs:103`.

The consequence is that insurance funds are permanently locked.

Recommendation

We recommend deducting the shortfall in funds from the `whitelisted_market.total_insurance` value when the entitled withdrawal amount is greater than the true credit limit i.e `whitelisted_market.total_credit - whitelisted_market.total_insurance`.

Status: Resolved

Resolved in [b0dd4cd]

2. Protocol mint fees locked in factory contract for perpetuity

Severity: Major

The Edge pool contract mints eTokens to the factory contract when computing the interest accrued in `contracts/edge-pool/src/borrow.rs:532`. However, there is no function in the factory contract that allows the factory owner to redeem these eToken funds.

The eTokens accrued during interest computations are thus locked in perpetuity.

Recommendation

We recommend adding a function to the factory contract that allows the factory admin to withdraw the tokens accrued by `protocol_mint_fee`.

Status: Resolved

Resolved in [b0dd4cd]

3. No validation of collateral factor and liquidation incentives could lead to loss of funds

Severity: Major

During both the registration of a new market in `contracts/edge-pool/src/gov.rs:21` and in the update of an existing market in `contract/edge-pool/src/gov.rs:148` the collateral factor and liquidation incentives are defined. Both fields define portions of collateral value during valuation and liquidation respectively. Currently they could be set to any `Decimal::256` value which could lead to haircuts of greater value than the underlying or liquidators receiving more collateral than initially deposited. Paying out too much collateral to users implies a loss of funds for other users.

Even though these values can only be set by the admin, we classify this issue as major since any mistake in the config can lead to irrecoverable losses.

Recommendation

We recommend validating that the `collateral_factor` and `liquidation_incentive` are both less than `Decimal::one()` prior to storage of the `TMP_COLLATERAL_INFO` on the line `contracts/edge-pool/src/gov.rs:47`. Similarly, the same validation should occur during the update of `MARKET_CONFIG` in line `contracts/edge-pool/src/gov.rs:171`.

Status: Resolved

Resolved in [ff3c3db]

4. Unbounded size of `MARKET_WHITELIST` leads to increasing gas consumption with number of markets, eventually causing gas depletion

Severity: Major

The map `MARKET_WHITELIST` in `contracts/edge-pool/src/state.rs:49` has an unbounded size.

However, a number of functions, such as `_compute_interest` and `query_markets`, iterate over the `MARKET_WHITELIST`. As the number of markets grows, executing these functions will consume more gas, up to the point where they will no longer be executable within gas limits.

Recommendation

We recommend performing gas analysis and setting an upper bound for the number of markets that can be whitelisted.

Status: Acknowledged

5. No validation of admin and insurance fee rates may lead to underflows

Severity: Minor

Neither during the instantiation of the pool contract in `contracts/edge-pool/src/contract.rs:49`, nor in the configuration update function `set_config` does the protocol verify that `admin_fee_rate < Decimal256::one()`. Likewise, the `insurance_fee_rate`, whilst initially set to `Decimal256::zero()`, can also be set to a value greater than `Decimal256::one()` in the `set_config` function in `269`.

This means the protocol admin could set the `admin_fee_rate` and `insurance_fee_rate` to be greater than the value of a specific transaction, causing potentially an underflow in several places in the codebase.

Recommendation

We recommend validating that the `admin_fee_rate + insurance_fee_rate < Decimal256::one()` in both the instantiation and configuration update function `set_config`.

Status: Resolved

Resolved in [b0dd4cd]

6. No validation of protocol fee rate during config updates

Severity: Minor

During the instantiation of the factory contract (`contracts/edge-factory/src/contract.rs`) the protocol verifies that `protocol_fee_rate < Decimal256::one()`. However in the function `execute_update_config` the user input is not validated and could be set to greater than `Decimal256::one()`. A value bigger than one would cause an underflow in `contracts/edge-pool/src/borrow.rs:515`.

Recommendation

We recommend validating that the `protocol_fee_rate < Decimal256::one()` in the `execute_update_config` function as is performed during the contract instantiation.

Status: Resolved

Resolved in [b0dd4cd]

7. Increase of Terra taxes could cause positions to default without an ability to repay or liquidate

Severity: Minor

The current gateway implementation only allows to `deposit`, `liquidate` and `repay` when the Terra tax is zero, see `contracts/gateway/guard-cap/src/contract.rs:107`, `134`, and `contracts/gateway/guard-cap/src/contract.rs:159`.

This is currently no issue since Terra taxes have been set to zero in a recent governance vote. They could increase again, however, causing those functions to not be executable anymore. This is especially problematic for liquidations and repayments, since positions could go underwater without an ability to recover them.

Recommendation

We recommend removing this check and allowing the gateways to function when there is a non-zero tax.

Status: Acknowledged

8. Recurring use of magic numbers throughout codebase

Severity: Informational

Throughout the protocol codebase numbers are used without context for calculations – so-called “magic numbers”. Usage of magic numbers is problematic since they decrease the maintainability of the codebase. The following lines contain magic numbers without further context:

- `contracts/edge-pool/src/borrow.rs:31`
- `contracts/edge-pool/src/borrow.rs:433-434`
- `contracts/edge-pool/src/gov.rs:64`
- `contracts/edge-pool/src/gov.rs:302`
- `contracts/helpers/wrapped-oracle/src/contract.rs:64`

Recommendation

We recommend replacing magic numbers with constants.

Status: Resolved

Resolved in [8ad8c09]

9. Double slope IRM misses kink validation

Severity: Informational

The instantiation of the double slope interest rate mechanism in `contracts/irm/double-slope/src/contract.rs` defines the key characteristics of the slope. This includes the first and second slope multipliers and the "kink", the point at which the slope changes from the first to the second multiplier.

Currently, the kink can be set as greater than `Decimal::one()` which would prevent the second multiplier from being activated.

Recommendation

We recommend validating that `DoubleSlopeModelInstantiateMsg.kink1 < Decimal::one()` prior to line `contracts/irm/double-slope/src/contract.rs:22`.

Status: Resolved

Resolved in [4d9ca6e]

10. Triple slope IRM misses kink validation

Severity: Informational

The instantiation of the triple slope interest rate mechanism in `contracts/irm/triple-slope/src/contract.rs` defines the key characteristics of the slope. This includes the slope multipliers and the "kinks", the points at which the slope changes from the first to the second and then from the second to the third multiplier.

Currently, both `kink1` and `kink2` can be set to values greater than `Decimal::one()`. Additionally, it is possible to set `kink2 < kink1`, which would remove the mid-section of the triple slope.

Recommendation

We recommend validating that `TripleSlopeModelInstantiateMsg.kink1` and `TripleSlopeModelInstantiateMsg.kink2` are less than `Decimal::one()` prior to line `contracts/irm/double-slope/src/contract.rs:22`. Additionally, the protocol should validate that `kink1 < kink2`.

Status: Resolved

Resolved in [4d9ca6e]

11. Overflow checks not enabled for release profile

Severity: Informational

The following packages and contracts do not enable `overflow-checks` for the release profile:

- `contracts/edge-etoken/Cargo.toml`
- `contracts/edge-factory/Cargo.toml`
- `contracts/edge-pool/Cargo.toml`
- `contracts/gateway/Cargo.toml`
- `contracts/irm/*/Cargo.toml`
- `packages/edge-protocol/Cargo.toml`
- `packages/edge-utils/Cargo.toml`

While enabled implicitly through the workspace manifest, a future refactoring might break this assumption.

Recommendation

We recommend enabling overflow checks in all packages, including those that do not currently perform calculations, to prevent unintended consequences if changes are added in future releases or during refactoring. Note that enabling overflow checks in packages other than the workspace manifest will lead to compiler warnings.

Status: Resolved

Resolved in `5d9c337`

12. Unreachable error for eToken limit

Severity: Informational

The eToken contract verifies that the total supply at instantiation does not exceed the cap defined by `msg.get_cap` in lines `contracts/e-token/src/contract.rs:43-47`. However, the `total_supply` is hardcoded to zero at deployment, and therefore the condition `if total_supply > limit {...}` cannot be triggered.

Recommendation

We recommend removing lines `43-47` from `contracts/e-token/src/contract.rs` to simplify the codebase.

Status: Resolved

Resolved in `02ec188`