Audit Report

Magma Core

v1.0

December 4, 2024

Table of Contents

License

Disclaimer

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

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

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

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

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by Osmosis Grants Company to perform a security audit of Magma Core.

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 target:

Repository	https://github.com/magma-vaults/magma-core
Commit	`6e7de4451d5959777601f719d2b407a5b7c7b649`
Scope	All contracts were in scope.
Fixes verified at commit	`bc601a846c2f78627e65c3ceda206b418e23d600` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

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

Magma Core is a decentralized finance platform allowing users to provide liquidity on vaults that will be used to create liquidity positions on Osmosis pools. These positions are rebalanced periodically to ensure they fall between the ideal price ranges.

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**, **Partially Resolved**, or **Resolved**.

Note that audits are an important step to improving 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 in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Low-Medium → Medium-High	The codebase is under development as there are multiple instances where values must be initialized before mainnet deployment. Multiple `TODO` and `FIXME` comments are also present in the codebase. After the audit, the client made improvements to resolve the above comments, increasing the code readability and clarity.
Level of documentation	Medium → Medium-High	After the audit, the client provided a whitepaper documenting the project's architecture.
Test coverage	None	The codebase implements tests in Osmosis's `test-tube`, which does not support coverage reports at the time of preparing this audit report.

Summary of Findings

No	Description	Severity	Status
1	Protocol fees cannot be updated due to unexposed entry point	Major	Resolved
2	Rebalancing can be triggered on manipulated pools	Minor	Resolved
3	Hardcoded protocol address	Minor	Partially Resolved
4	Admin fee field is ignored when updating vault parameters	Minor	Resolved
5	Overflow checks not enabled for release profile	Minor	Resolved
6	Liquidity tokens cannot be transacted like standard tokens	Informational	Resolved
7	Contracts should implement a two-step ownership transfer	Informational	Resolved
8	Lack of input validation for `seconds_before_rebalance`	Informational	Acknowledged
9	Usage of magic numbers decreases maintainability	Informational	Resolved
10	Code quality improvement	Informational	Resolved
11	Additional funds sent to the contract are lost	Informational	Resolved
12	Missing event emission when withdrawing protocol fees	Informational	Acknowledged

Detailed Findings

1. Protocol fees cannot be updated due to unexposed entry point

Severity: Major

In `src/execute.rs:861`, the `change_protocol_fee` function implements functionality for the protocol to update the fee amount. However, since no entry point exposes it, the function cannot be called after the contract is deployed.

Recommendation

We recommend implementing a message that calls the `change_protocol_fee` function.

Status: Resolved

2. Rebalancing can be triggered on manipulated pools

Severity: Minor

In `src/execute.rs:134`, the `rebalance` function queries the pool's spot price to compute the balanced ratio of tokens for providing liquidity. If the rebalance transaction were executed when the pool is in an imbalanced state (either maliciously or by coincidence), the dispatched `MsgCreatePosition` message would be suboptimal because some tokens would be left out instead of being used to provide liquidity.

Consequently, users will receive fewer rewards when the rebalancing mechanism fails to utilize funds during liquidity provision.

Recommendation

We recommend using a time-weighted average price (TWAP) to determine whether rebalancing can be triggered, similar to Charm Finance's implementation.

Status: Resolved

3. Hardcoded protocol address

Severity: Minor

In `src/constants.rs:9`, the `PROTOCOL` constant value with delayed initialization is declared. The value used as the initializer is `Addr::unchecked("TODO")`, which is an invalid address. Since the protocol is intended to receive protocol fees, any transactions that attempt to send funds to the protocol will fail, including the vault creation cost paid in `src/state.rs:509`.

Additionally, in the event of a compromised protocol, the hardcoded address cannot be updated to minimize the attack surface.

Both scenarios mentioned above can be resolved by a contract migration transaction if the admin is defined before the contract is instantiated. However, contract migrations are usually reserved as a last resort for unexpected or emergency events, as implementing the state migrations is time-consuming and error-prone.

Recommendation

We recommend using the valid address of the protocol fees beneficiary by supplying it via instantiation parameters. Additionally, we recommend allowing the vault admin to update the protocol address.

Status: Partially Resolved

The issue is partially resolved because the protocol address is still a hardcoded address. The client states that the protocol address should have a higher authority than the vault admin address. As vaults are permissionless, anyone could be a vault admin.

4. Admin fee field is ignored when updating vault parameters

Severity: Minor

In `src/execute.rs:814-818`, the `change_vault_parameters` function accepts a parameter of type `VaultParametersInstantiateMsg`. Among other fields, this structure contains the field `admin_fee`.

The function is called to update parameters stored in the `FEES_INFO` state. However, it does not update the `admin_fee` parameter, which is contained in the message but is ignored by the `VaultParameters::new` function called in line `830`. This can be unexpected for the user, who explicitly specified a new value for the admin fee.

Recommendation

We recommend restructuring the message to remove the `admin_fee` field if unnecessary.

Status: Resolved

5. Overflow checks not enabled for release profile

Severity: Minor

The `Cargo.toml` file does not include the `overflow-checks` directive with the value set to `true`. This is problematic because the protocol does not enforce additional overflow protection, which may lead to more severe issues in the future if an overflow occurs.

Recommendation

We recommend setting the value of `overflow-checks` to `true`.

Status: Resolved

6. Liquidity tokens cannot be transacted like standard tokens

Severity: Informational

When users deposit tokens into the vault, their liquidity token balance is recorded in the `BALANCES` state, as seen in `src/execute.rs:116`. However, the user cannot transact the tokens normally because no entry point has been implemented.

For example, a user may want to send the tokens to another DeFi protocol for leverage or transfer some tokens to a recipient with a specific allowance. This limits composability and integration with other protocols, reducing the user experience.

Recommendation

We recommend implementing the necessary entry points to allow composability with other protocols. This includes, but is not limited to the `Transfer`, `Send`, `IncreaseAllowance`, `DecreaseAllowance`, `TransferFrom`, and `SendFrom` messages, which can be adopted from the CW20 contract.

Status: Resolved

7. Contracts should implement a two-step ownership transfer

Severity: Informational

The contracts within the scope of this audit allow the current owner to execute a one-step ownership transfer in `src/execute.rs:794-810`. While this is common practice, it presents a risk for the ownership of the contract to become lost if the owner transfers ownership to the incorrect address. A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address validated with `addr_validate`.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Resolved

8. Lack of input validation for `seconds_before_rebalance`

Severity: Informational

In `src/state.rs:408`, the `seconds_before_rebalance` parameter enforces the minimum time to elapse before rebalancing the vault. However, no validation ensures the parameter is set to a reasonable range.

This allows the vault admin to set extremely large values, potentially leading to significant delays in the rebalancing process. While this might not directly compromise the system's security, it could result in unexpected behavior and hinder operational efficiency.

Recommendation

We recommend introducing a maximum value for the `seconds_before_rebalance` parameter to ensure timely rebalancing operations. This maximum value should be carefully chosen based on the intended rebalancing frequency and system dynamics.

Status: Acknowledged

The client states that admins can always change `seconds_before_rebalance`, and vault users can always just leave the vault, so they do not think allowing long rebalancing times is problematic.

9. Usage of magic numbers decreases maintainability

Severity: Informational

Throughout the codebase, hard-coded number literals without context or a description are used. Using such “magic numbers” goes against best practices as they reduce code readability and maintenance because developers are unable to easily understand their use and may make inconsistent changes across the codebase.

Instances of magic numbers are listed below:

- `src/execute.rs:218-219`, `243-244` and `545`
- `src/state.rs:160` and `181`

Recommendation

We recommend defining magic numbers as constants with descriptive variable names and comments, where necessary.

Status: Resolved

10. Code quality improvement

Severity: Informational

The following instances illustrate code paths where readability can be improved:

- In `src/utils.rs:80-96`, the lambda function `compute_price_inverse` is defined but used only once on a single argument.
- In `src/state.rs:265-266`, `pool_id` is defined using the `info.pool_id` value. However, line `268` asserts that `pool_id.0` equals `info.pool_id`, which is impractical because the result will always be `true`.
- In `src/msg.rs:51-52`, the `amount0` and `amount1` parameters are defined, which are always checked to be equal to actual funds attached in `src/execute.rs:48`. However, both values are part of the user's input. Hence, the parameters and checks are redundant, as the values can be retrieved from user input instead.
- In `src/execute.rs:30`, the validation checks if any funds are sent by the user, including non-protocol coins. Similarly, in `src/execute.rs:64`, the same parameters `amount0` and `amount1` are checked against `MIN_LIQUIDITY`. The former validation can be removed because the latter validation checks for the same and is more strict than the former validation.
- Typos in `src/execute.rs:248` and `263`, "overflow" should be changed to "underflow" when commenting on subtraction operations.
- Numerous usages of `Decimal::from_str` and `SignedDecimal256::from_str` are noticed in the codebase, which defines numerical constants. This approach bypasses static type checking and induces some performance overhead in runtime. Consider using decimal constructors like `Decimal::permille` instead of parsing numbers from strings.

Recommendation

We recommend applying the recommendations mentioned above.

Status: Resolved

11. Additional funds sent to the contract are lost

Severity: Informational

The `deposit` function in `src/execute.rs:11` allows users to deposit funds into the vault. However, there is no validation if only vault assets were sent, and in turn, more funds than needed may be accidentally sent by users to the vault contract.

While blockchains generally do not protect users from sending funds to the wrong accounts, reverting extra funds increases the user experience.

Recommendation

We recommend checking that the transaction contains only the expected `Coins` and no additional native tokens.

Status: Resolved

12. Missing event emission when withdrawing protocol fees

Severity: Informational

In `src/execute.rs:706`, the `withdraw_protocol_fees` function lacks event emission for fee withdrawal amounts. This is problematic because it reduces on-chain transparency and complicates historical fee data reconstruction.

Recommendation

We recommend emitting an event containing values of the fees that were withdrawn.

Status: Acknowledged