Audit Report

Lum Millions Module

v1.0

May 15, 2023

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 KAIROS SAS to perform a security audit of the Lum Millions Cosmo SDK Module.

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/lum-network/chain-private
Commit	`5ce8852ad47550e8493695978bc9f1d408e8c2da`
Scope	The scope of this audit covered the millions module

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

Lum Network's Millions module implements a Prize-Linked Savings Account (PLSA) for the Cosmos Ecosystem. Users deposit native tokens to participate in prize draws.

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 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	The millions module is complex as it implements ICA and ICA callbacks.
Code readability and clarity	Medium	The readability and clarity of the codebase could be improved with more consistent validation practices as discussed in the findings.
Level of documentation	Medium-High	The Millions module included detailed documentation that accurately described its functionality.
Test coverage	Medium	58.7% code coverage for the Millions module.

Summary of Findings

No	Description	Severity	Status
1	Lack of accounting for slashing could lead to a bank run, resulting in loss of funds for slow depositors	Major	Acknowledged
2	Updating the validator set of a pool does not trigger a redelegation	Major	Acknowledged
3	Delegated funds cannot be redelegated	Major	Acknowledged
4	`DrawRetry` will always fail	Major	Resolved
5	`MinDepositAmount` is not strictly enforced	Minor	Resolved
6	The chain could be spammed with deposits of negligible amount	Minor	Resolved
7	An updated `InitialDrawAt` cannot be enforced	Minor	Resolved
8	Missing validation checks in the codebase	Minor	Partially Resolved
9	Inactive validators may receive delegations, reducing protocol yield	Minor	Acknowledged
10	`PrizeExpirationDelta` parameter can be set to a value that prevents prize claims	Minor	Resolved
11	`FeesStakers` parameter can be set unreasonably high	Informational	Resolved
12	Redundant checks are inefficient	Informational	Partially Resolved
13	Outstanding TODO comments in codebase	Informational	Acknowledged
14	Inaccurate messages may confuse or mislead users	Informational	Resolved
15	Missing usage description for all transaction and query CLI commands	Informational	Resolved
16	Code inefficiencies	Informational	Resolved
17	Code quality could be improved	Informational	Resolved
18	Unused pool states	Informational	Resolved

19	Additional rewards are ignored	Informational	Acknowledged

Detailed Findings

1. Lack of accounting for slashing could lead to a bank run, resulting in loss of funds for slow depositors

Severity: Major

The codebase does not handle any reduction in a pool's total value locked, which could happen for example due to slashing. The logic in the `TransferWithdrawalToLocalChain` function is implemented with the assumption that the amount returned by the unbonding callback will be equal to the amount specified at the beginning of the unbonding request. In the event that slashing occurs, the module will have a lower than expected balance which will cause later withdrawal requests to fail. This will create a situation where early withdrawals can be fully filled while later withdrawals will fail once the module's balance drops below the requested withdrawal amount. These withdrawals will fail until the module's balance is re-capitalized. The missing amount would need to be fully refunded to the module's balance for all users to be able to successfully withdraw their initial amounts.

Recommendation

We recommend implementing a proportional reduction of deposits of all users in the case of slashing. Alternatively, withdrawals and draws could be paused until the yield generated is sufficient to cover the total nominal value of deposits.

Status: Acknowledged

The client acknowledges the current limitation of the Millions module regarding slashing. They intend to implement features to account for slashing in upcoming releases. In the initial release of the Millions module, the client will only select validators that can offer refunds in the case of a slashing event.

2. Updating the validator set of a pool does not trigger a redelegation

Severity: Major

The parameters of a pool can be updated through an update-pool proposal, including any changes to the validator set. If the proposal passes, the pool will be updated in `x/millions/keeper/keeper_pool.go` through the `UpdatePool` function. However, a change in the validator set does not trigger a redelegation to the new set of validators, instead only the flag `isEnabled` is updated. Any new deposits will be delegated to the new set of validators but for existing delegations to be updated, depositors need to withdraw, wait for the three-week unbonding period and redeposit.

Recommendation

We recommend handling the update of the validator set by redelegating the coins appropriately.

Status: Acknowledged

The client acknowledges this issue and intends to introduce this feature in one of the first major upgrades of the Millions module.

3. Delegated funds cannot be redelegated

Severity: Major

The codebase does not send redelegate packets, thus the function `RedelegateCallback` in `x/millions/keeper/callbacks_redelegate.go` is never used. In the situation where delegations need to flow from one validator to another (e.g. if a validator stops validating blocks) after the validator set got updated through the update-pool proposal, every depositor would need to manually withdraw, wait for the three-week unbonding period, and redeposit. This will require coordination, manual actions by users and impact the overall staking returns, as some depositors may not follow those steps and others have to wait for the unbonding period.

Recommendation

We recommend implementing redelegation.

Status: Acknowledged

The client acknowledges this issue and aims to introduce a feature to manage redelegations.

4. `DrawRetry` will always fail

Severity: Major

The function `DrawRetry` in `x/millions/keeper/msg_server_draw.go:13` does not behave correctly, and as a result, the draw will never be able to be re-executed. The `ClaimRewardsOnNativeChain`, `TransferRewardsToLocalChain`, and `ExecuteDraw` functions are all incorrectly called with the parameter `draw.PoolId` instead of the `DrawID`. This will cause the `TransferRewardsToLocalChain` and `ExecuteDraw` functions to fail due to an erroneous draw state. The `ClaimRewardsOnNativeChain` function will be executed successfully, claiming pool rewards but returning a wrong `draw`, unless `draw.DrawId` is equal to `draw.PoolId`. This means that any draw that fails its initial execution, will be stuck in an erroneous state.

Recommendation

We recommend calling the functions with the correct parameters, namely `(ctx, draw.PoolId, draw.DrawId)`.

Status: Resolved

5. `MinDepositAmount` is not strictly enforced

Severity: Minor

The function `SetParams` in `x/millions/types/params.go:21` does not explicitly validate that the value of `MinDepositAmount` is not zero. Although a proposal to update the parameters would prevent a minimum deposit amount of zero, as it is caught in the `update_params` message's `ValidateBasic` function, `MinDepositAmount` could be set to 0 during genesis without causing an error.

Recommendation

We recommend enforcing a strictly positive `MinDepositAmount` in `SetParams`.

Status: Resolved

6. The chain could be spammed with deposits of negligible amount

Severity: Minor

The function `SetParams` in `x/millions/types/params.go:36` does not enforce a reasonable minimum acceptable amount for `MinDepositAmount`. Thus, the parameter can theoretically be set close to zero, allowing for spam deposits with negligible amounts. The `LaunchNewDraw` function is triggered by the `BeginBlocker` function and iterates over an unbounded loop of deposits. The ability to add a high amount of negligible deposits can allow an attacker to slow down or even halt block production. The minimum deposit needs to be reasonably high to provide enough financial disincentive against such an attack.

We classify this issue as minor since governance controls this value.

Recommendation

We recommend enforcing a minimum acceptable deposit.

Status: Resolved

7. An updated `InitialDrawAt` cannot be enforced

Severity: Minor

The `ShouldDraw` function in `x/millions/types/draw_schedule.go` returns `true` if a draw should be executed. For the first draw, it will only return `true` if it is passed the `InitialDrawAt` parameter of `DrawSchedule`. However, if the draw schedule is updated through a proposal and a new `InitialDrawAt` is set to a time in the future, this will not be enforced since the condition in line `46` will not be met. Thus, draws will continue to be drawn every `DrawDelta`, even if `InitialDrawAt` has not been met.

Recommendation

We recommend updating the `lastDrawAt` to `nil` if a new `DrawSchedule` with a valid `InitialDrawAt` in the future is proposed and passed.

Status: Resolved

8. Missing validation checks in the codebase

Severity: Minor

There are missing or insufficient validation checks in the codebase:

- The `PrizeBatch` validation in `x/millions/types/prize_batch.go:19` allows for a `DrawProbability` of 0. A prize strategy that has 0 draw probability for all batches would make it impossible for the pool to have a winner.
- The `FeesStakers` parameter is not validated in `x/millions/types/proposal_update_params.go`.
- The `DrawSchedule` validation in `x/millions/types/draw_schedule.go:10` does not check if `DrawDelta` is less than the `MaxDrawScheduleDelta`. Use the `ValidateNew` function for validation of `DrawSchedule` instead.
- The `PoolId`, `DepositId`, and `ToAddress` are not properly validated in `MsgWithdrawDeposit` in `x/millions/types/message_withdraw_deposit.go:36`.
- In the `MsgDeposit`'s `ValidateBasic` function in `x/millions/types/message_deposit.go:49`, the `msg.Amount` is being checked to ensure the amount deposited is not negative, but it does not check if the amount is `0`.

Recommendation

We recommend revisiting validation checks to enhance security and maintainability of the codebase.

Status: Partially Resolved

All missing validation checks have been resolved except for the first point. The client acknowledges it, stating that it is highly unlikely for governance to create a pool with a `DrawProbability` of 0. Furthermore, users will have no reason to deposit into a pool that cannot have winners.

9. Inactive validators may receive delegations, reducing protocol yield

Severity: Minor

The `DelegateDepositOnNativeChain` function in `x/millions/keeper/keeper_deposit.go:111` delegates deposits to the native chain of the pool. However, it is possible for a validator to have already stopped validating and unbonded at the time the latest pool validator set has been proposed and approved. This would allow delegations of pool deposits to inactive validators that will not produce any yield.

Recommendation

We recommend checking the validator's status before delegating.

Status: Acknowledged

10. `PrizeExpirationDelta` parameter can be set to a value that prevents prize claims

Severity: Minor

The function `ValidateBasics` in `x/millions/types/params.go:36` would allow for a `PrizeExperiationDelta` as short as a second. Very short periods would prevent winners from claiming their prizes. We classify this issue as minor since it can only be caused by governance.

Recommendation

We recommend enforcing a minimum acceptable period for prize expiration.

Status: Resolved

11. `FeesStakers` parameter can be set unreasonably high

Severity: Informational

The function `ValidateBasics` in `x/millions/types/params.go:36` does not prevent an unreasonably high value for the `FeesStakers` parameter. Although this parameter is controlled by governance, it is possible to be set close to 1, which would mean that all draw proceedings will be taken as fees, causing depositors to receive only a minimal amount of rewards.

Recommendation

We recommend enforcing maximum acceptable stakers fees.

Status: Resolved

12. Redundant checks are inefficient

Severity: Informational

The codebase contains redundant checks that lead to inefficiencies:

- The last draw state check in `x/millions/keeper/keeper_draw.go:51` is a duplicate validation as it is invoked already through `x/millions/keeper/keeper_blockers.go:375`, `ListPoolsToDraw`.
- The `ValidateDenom` function in `x/millions/types/pool.go` checks the length of the denomination inside the function. The validation checks in lines `20` and `26` can be removed.
- The validation to check if the sender has sufficient balance in `x/millions/keeper/msg_server_deposit.go:60-62` is duplicate as it is already checked in the `SendCoins` function called in line `79`.
- The `sanitizedDenom` function in `x/millions/keeper/msg_server_deposit.go:34` can be removed if a stateless check is performed in the message `ValidateBasic` function in `x/millions/types/message_deposit.go:37`.
- A length of validators check in `x/millions/keeper/keeper_pool.go:251` is duplicate as it is being checked already in `x/millions/types/proposal_update_pool.go:49`.
- The `strings.TrimSpace` call in `x/millions/types/message_deposit.go:53` can be removed as it is being checked already through the `AccAddressFromBech32` function in line `55`.
- All the transaction commands in `x/million/client/cli/tx.go:25-30` call `msg.ValidateBasic`. This is unnecessary, since the function is already invoked in the `GenerateOrBroadcastTxWithFactory` function.

Recommendation

We recommend removing the aforementioned redundant checks to remove unnecessary computation and increase the readability and maintainability of the codebase.

Status: Partially Resolved

This finding has been marked as partially resolved. The client acknowledges the first point and states that it is necessary to facilitate unit testing.

13. Outstanding TODO comments in codebase

Severity: Informational

There are multiple TODOs in the codebase that may refer to important logic. Instances are:

- `x/millions/keeper/keeper_draw.go:199`
- `x/millions/keeper/keeper_pool.go:85`

Recommendation

We recommend resolving all TODOs in the codebase before the production release, especially those which may relate to important logic.

Status: Acknowledged

The client has decided to keep the TODO comments to accurately depict the future changes that will be implemented for findings 1-3.

14. Inaccurate messages may confuse or mislead users

Severity: Informational

Across the codebase, instances of inaccurate messages have been found, that could confuse or mislead users.

Recommendation

The following are some recommendations to improve the user experience and avoid confusion:

- The long message in `x/millions/client/cli/proposal.go:231`, `CmdProposalUpdateParams` should be “Submit an update-pool proposal...”, instead of “Submit an register-pool proposal..”.
- The function `CmdTxWithdrawDepositRetry` in `x/millions/client/cli/tx.go` does not expect any optional flags, however args is set to `cobra.MinimumNArgs(2)`. Consider changing the expected arguments to exactly two.

- The function `CmdTxWithdrawDeposit` in `x/millions/client/cli/tx.go` receives `to_address`, but the value is used for both `DepositAddress` and `ToAddress`.
- The function `CmdTxWithdrawDeposit` in `x/millions/client/cli/tx.go` allows 2 arguments but the example demonstrates usage with 3 arguments.

Status: Resolved

15. Missing usage description for all transaction and query CLI commands

Severity: Informational

All the transaction and query CLI commands for the Millions module in `x/million/client/cli/tx.go:25-30` and `x/million/client/cli/query.go:37-59` are missing a long message that describes their usage, which would be helpful for users and external developers.

Recommendation

We recommend specifying a long message for all transaction and query CLI commands. Each command should provide a description of how to correctly use the command.

Status: Resolved

16. Code inefficiencies

Severity: Informational

There are several parts of the codebase that can be optimized to perform stateful and stateless checks, reduce computational resources and gas consumption:

- Trim whitespace to catch more cases of invalid values instead of just empty strings for the parameters `ChainId`, `Bech32PrefixAccAddr`, and `Bech32PrefixValAddr` in `x/millions/types/pool.go:13`, `ValidateBasic`.
- In `x/millions/types/message_deposit.go:44-51`, using `msg.Amount.Validate` can replace the implemented functions.
- In `x/millions/types/proposal_register_pool.go:59-64`, validation checks with `strings.TrimSpace` can be replaced with the `ValidateDenom` function from Cosmos SDK's `types` package.
- The `ComputeSplitDelegations` function in `x/millions/keeper/keeper_deposit.go:130-133` and `158-161` can be combined.

Recommendation

We recommend implementing the optimizations mentioned above to reduce computational resource and gas consumption. Furthermore, these optimizations also enhance the overall codebase's readability and maintainability.

Status: Resolved

17. Code quality could be improved

Severity: Informational

Across the codebase instances of unused or commented code have been found.

Recommendation

We recommend improving the code quality as follows:

- Remove unused error `ErrInvalidPoolAccountAddress` in `x/millions/types/errors.go:30`.
- Remove unused event `EventTypeRegisterPool` in `x/millions/types/events.go:5`.
- Remove unused file `x/millions/types/prize_ref.go`.

Status: Resolved

18. Unused pool states

Severity: Informational

Currently, the pool states `PoolState_Paused` and `PoolState_Killed` are unused. Additionally, the `Deposit` function in `x/millions/keeper/msg_server_deposit.go:15` allows for deposits to be made to a paused pool. If a paused pool state was implemented in the future without updating this function it could be problematic.

Recommendation

We recommend updating the pool states to reflect those actually in use, or clearly documenting that they will be implemented in the future.

Status: Resolved

19. Additional rewards are ignored

Severity: Informational

Currently, the `OnClaimRewardsOnNativeChainCompleted` ignores additional coins returned in the ICA claim rewards callback if they do not match the module account. With interchain security, it is becoming more common to expect chains to return more than one type of reward coin. Overall this would also increase the rewards users receive.

Additionally, there is an unbounded iteration in `x/millions/keeper/keeper_draw.go:193` that iterates over every coin received. While the likelihood of receiving a large amount of additional coins from a trusted chain is low this edge case should still be handled.

Recommendation

We recommend updating the `OnClaimRewardsOnNativeChainCompleted` function to handle additional coins. Alternatively, such additional coins could also be registered to the pool config as a whitelisted reward denom.

Status: Acknowledged

The client acknowledges this issue and states that they will monitor the amount of additional rewards and assess the priority of creating a mechanism to capture them.