Security Audit Report

Paxi Network

v1.0

November 1, 2025

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 Mach One Technology Limited to perform a security audit of Security audit of the Paxi network custom staking and minting Cosmos SDK modules.

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/paxi-web3/paxi
Commit	`fa0651faa8d9d22961720fd6ddaa86548473c809`
Scope	The scope is restricted to the following custom Cosmos SDK modules: • `x/custommint` • `x/customstaking`
Fixes verified at commit	`236627a4a4d8d9873c0d2c25fb730bcf9e965c6d`

	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

PAXI is a next-gen Layer 1 Cosmos SDK-based blockchain.

This audit covers Paxi Network's `custommint` and `customstaking` Cosmos SDK modules, which introduce custom logic for token supply management and validator selection. Key features include a dynamic inflation and fee-burning system, alongside a hybrid validator selection model that combines top stake with weighted randomness to enhance decentralization.

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	Low-Medium	-
Code readability and clarity	Medium-High	-
Level of documentation	Medium	-
Test coverage	Low	No tests have been provided.

Summary of Findings

No	Description	Severity	Status
1	Rejection sampling may cause unbounded looping under skewed validator weights	Major	Resolved
2	`MinBondedToken` check bypass allows under-bonded validators to remain active	Minor	Resolved
3	Panics and unhandled errors in the custom minting module's `BeginBlock` may halt the chain	Minor	Resolved
4	CLI queries do not use the command context	Minor	Resolved
5	Invalid burn threshold parameter is silently set to zero when updating custom minting module parameters	Minor	Resolved
6	Inconsistent enforcement of minimum delegation amounts	Minor	Acknowledged
7	Validator selection mechanism may reduce economic security	Minor	Acknowledged
8	Tokens are not minted at block height 1 due to the early return condition	Informational	Acknowledged
9	Wrong error message when fetching a new validator set	Informational	Resolved
10	Staking parameters are not queriable	Informational	Acknowledged
11	Redundant `found` check adds unnecessary complexity in the validator power update logic	Informational	Resolved
12	Skip minting logic if the mint amount is zero	Informational	Resolved
13	Misleading comment and error message for minted coin destination	Informational	Resolved
14	Parameter update logic is duplicated and bypasses the keeper API	Informational	Resolved
15	Updating module parameters lacks event emission	Informational	Resolved
16	JSON serialization is used for module parameters instead of the application codec	Informational	Acknowledged
17	Misleading error message in `BurnThreshold`	Informational	Resolved

	validation		
18	Deterministic burn behavior is misrepresented as probabilistic	**Informational**	**Resolved**

Detailed Findings

1. Rejection sampling may cause unbounded looping under skewed validator weights

Severity: Major

In `x/customstaking/keeper.go:461-472`, the selection loop repeatedly draws a random weight, maps it to a validator via the prefix-sum distribution, and skips any validator already chosen. When the stake is heavily concentrated among a few validators, the random draw will keep landing in the large ranges that belong to those top-stake validators. Because those validators are often already marked selected, the loop continues without advancing and may run indefinitely.

The attack surface is aggravated by the interaction between the existing (last) validator set, the candidate pool limits, and `MaxValidators`. If the last validator set and the new candidate set are both small relative to `MaxValidators` initially, the algorithm must fill out a much larger validator set up to `MaxValidators` as

```
(maxValidators = sdkmath.Min(maxValidators, totalCandidates)).
```

With `MaxCandidates` as high as 2000, an attacker can spawn many low-delegation candidate validators. In a realistic distribution where the top ~50% of the last set holds most of the stake (for example: 30 validators controlling ~60% of supply) and the candidate pool is similar to last validator set, an attacker who adds dozens of low-stake candidates forces the sampler to hit the same high-stake ranges repeatedly (which are already selected) while trying to discover the many low-weight, unselected entries. This causes severe delays or stalls when building the new validator set.

Recommendation

We recommend replacing rejection sampling with a weighted sampling without replacement algorithm (e.g., Efraimidis–Spirakis method). These algorithms ensure a unique selection proportional to weights in a single pass without repeated sampling attempts

Status: Resolved

2. `MinBondedToken` check bypass allows under-bonded validators to remain active

Severity: Minor

In `x/customstaking/keeper.go:294-298`, the keeper builds `pendingUpdates` when a validator's consensus power changes or when a previously untracked validator appears.

The validator set itself, however, is only recomputed periodically via `getValidatorsAboveThreshold()`, approximately every 200 blocks.

Consequently, if a validator is included in the active set and later undelegates (reducing its `ConsensusPower` below `MinBondedTokens`), the code path that runs between recompute intervals does not re-evaluate the `MinBondedTokens` threshold. As a result, an under-bonded validator can remain active and continue signing blocks until the next full rotation.

Recommendation

We recommend handling power updates (the branch that appends to `pendingUpdates` and explicitly testing `newPower` against the `MinBondedTokens` threshold.

Status: Resolved

3. Panics and unhandled errors in the custom minting module's `BeginBlock` may halt the chain

Severity: Minor

In `x/custommint/module.go:125-129`, the `BeginBlock` function orchestrates token burning and minting operations for each block. It calls the `BurnExcessTokens` and `BlockProvision` functions from the minting keeper.

The `BurnExcessTokens` function in `x/custommint/keeper/keeper.go:144` will panic if burning coins fails. Additionally, the `BeginBlock` function propagates any errors returned from `BlockProvision`. In the Cosmos SDK, a panic or a returned error from the `BeginBlock` execution phase will halt the chain, leading to a denial of service.

Recommendation

We recommend preventing panics and handling errors gracefully to avoid chain halts. The `BurnExcessTokens` function should be modified to return an error instead of panicking. In `BeginBlock`, errors from both `BurnExcessTokens` and `BlockProvision` should be logged rather than being returned, as a failure in the minting process for a single block should not compromise the liveness of the entire network.

Status: Resolved

4. CLI queries do not use the command context

Severity: Minor

In the `custommint` module's CLI query commands, `context.Background()` is used to create a context for the query instead of using the context from the command itself, i.e., `cmd.Context()`.

`context.Background()` returns an empty, non-cancellable context that never has a deadline. This means that user-initiated cancellations, for example, by pressing Ctrl+C, and command timeouts are ignored by the query. This can lead to a poor user experience with seemingly unresponsive CLI commands and unnecessary resource consumption on the queried node, as the queries will continue to run.

The affected instances are in `x/custommint/client/query.go`:

- `CmdQueryTotalMinted` in line `21`
- `CmdQueryTotalBurned` in line `43`
- `CmdQueryParams` in line `65`

Recommendation

We recommend replacing `context.Background()` with `cmd.Context()` in all instances. This ensures that the queries are executed within the command's context, respecting cancellations and timeouts.

Status: Resolved

5. Invalid `BurnThreshold` parameter is silently set to zero when updating custom minting module parameters

Severity: Minor

In `x/custommint/keeper/msg_server.go:30`, the `UpdateParams` message handler does not correctly handle a potential error when parsing the `BurnThreshold` parameter. The `sdkmath.NewIntFromString` function is used to convert the string parameter to an integer, but the boolean return value that indicates parsing success is ignored.

If governance provides an invalid string for the `BurnThreshold` parameter, the parsing will fail, and `sdkmath.NewIntFromString` will return a zero value. Because the error is not checked, the `BurnThreshold` is silently set to `0` rather than rejecting the transaction. This can lead to unintended behavior of the token burn mechanism, as the parameter is not set to the value intended by governance.

Recommendation

We recommend checking the boolean value returned by `sdkmath.NewIntFromString`. If parsing the `BurnThreshold` parameter fails, the function should return an error. This will ensure that governance proposals with invalid parameter values are rejected.

Status: Resolved

6. Inconsistent enforcement of minimum delegation amounts

Severity: Minor

In `x/customstaking/msg_server.go`, the `Delegate` and `Undelegate` functions enforce minimum amount thresholds for token delegations and undelegations. For example, in lines `21-23`, the `Delegate` function validates that the delegation amount is not less than `MinDelegation`. Similarly, in lines `29-31`, the `Undelegate` function checks that the amount is not less than `MinUndelegation`.

However, this enforcement is not consistently applied. Specifically, the `BeginRedelegate`, and `CreateValidator` message handlers also lack a similar check.

Recommendation

We recommend consistently applying the minimum amount checks to all relevant message handlers.

Status: Acknowledged

The client acknowledges this finding with the note that in `x/customstaking/msg_server.go`, the methods `BeginRedelegate`, `EditValidator`, `CreateValidator`, `CancelUnbondingDelegation`, and `UpdateParams` are thin wrappers that intentionally delegate to the upstream Cosmos SDK staking `MsgServer` without altering semantics.

7. Validator selection mechanism may reduce economic security

Severity: Minor

In `x/customstaking/keeper.go:205-218`, the logic for selecting the active validator set is implemented. The process involves deterministically selecting the top 50% of potential validators, sorted descending by stake, and then choosing the remaining 50% through a weighted random selection from the other candidates.

While this approach introduces randomness, it does not guarantee that the resulting active validator set holds a sufficient amount of the total stake. It is possible for the selected group of validators to collectively control significantly less than a certain threshold required for liveness and security guarantees in a Byzantine Fault Tolerant system.

Recommendation

We recommend modifying the validator selection mechanism to ensure the chosen set consistently holds a significant percentage of the cumulative stake. For instance, the system could first select top validators by stake until a certain threshold (e.g., 67% of total stake) is

met, and then fill the remaining validator slots using a weighted random selection from the rest of the candidates.

Status: Acknowledged

The client acknowledges this finding with the note that the 50/50 deterministic and random validator selection approach is an intentional design choice to ensure decentralization and fairness while maintaining BFT safety within the active set. They further state that an economic coverage threshold is not enforced by design.

8. Tokens are not minted at block height 1 due to the early return condition

Severity: Informational

In `x/custommint/keeper/keeper.go:42-44`, the code checks whether the current `blockHeight` modulo `mintThreshold` equals zero. If not, the function returns early without minting any tokens. This means that at block height 1, token minting is skipped entirely, as the modulo condition fails. This behavior may result in skipped inflation or delayed initial minting.

Recommendation

We recommend reviewing the intended minting schedule and adjusting the condition to ensure tokens are minted at the correct initial block.

Status: Acknowledged

The client acknowledges this finding with the note that this behavior is intentional and has no material impact on the network. While minting was skipped once at block height 1, the schedule has continued normally for over 1.5 million blocks since then. Consequently, the total minted supply is only affected by an insignificant one-block delay in the initial emission, with no impact on inflation consistency or long-term token economics.

9. Wrong error message when fetching a new validator set

Severity: Informational

In `x/customstaking/keeper.go:184-187`, calling the `k.getValidatorsAboveThreshold` function returns an error, which is wrapped and returned as `"failed to get last validator set"`.

However, the operation is actually generating the new candidate validator set rather than retrieving a previous one. This misleading wording can lead to confusion when debugging or analyzing logs.

Recommendation

We recommend changing the error message to describe the operation accurately.

Status: Resolved

10. Staking parameters are not queriable

Severity: Informational

In `x/customstaking/params.go:3-9`, the module defines certain staking-related parameters as plain Go variables rather than as on-chain parameters registered through the Cosmos SDK parameter subspace. As a result, these parameters are not exposed to the chain's query layer and cannot be retrieved via gRPC or CLI queries.

Recommendation

We recommend defining these parameters using the Cosmos SDK `x/params` module and registering them in the module's parameter subspace. Alternatively, create a special query handler.

Status: Acknowledged

The client acknowledges this finding with the note that the parameters defined in `x/customstaking/params.go` are static chain constants and are not meant to be modified through governance or exposed via gRPC queries.

11. Redundant `found` check adds unnecessary complexity in the validator power update logic

Severity: Informational

In `x/customstaking/keeper.go:297-298`, the code checks `if !found || oldPower != newPower` before appending to `pendingUpdates`. However, the `found` variable is derived from the `last` map, and both `last` and `addrStr` are already constructed from the same validator set context. Because every validator in the iteration corresponds to an entry being compared or updated, the `found` condition adds no meaningful distinction.

Recommendation

We recommend removing the redundant `found` check and relying solely on comparing `oldPower` and `newPower` to decide when to append updates.

Status: Resolved

12. Skip minting logic if the mint amount is zero

Severity: Informational

In `x/custommint/keeper/keeper.go:47-54`, the `BlockProvision` function calculates the `mintAmount`, proceeds to mint coins, updates the total minted amount, and transfers the newly minted coins to the fee collector module account.

However, the function does not verify if the calculated `mintAmount` is greater than zero. While the `sdk.NewCoins` function sanitizes the input by removing zero coins, the subsequent logic for minting and transferring coins is still executed. This leads to unnecessary operations, even when no actual minting occurs.

Recommendation

We recommend adding a check to ensure `mintAmount` is greater than zero before proceeding with the minting logic to avoid unnecessary operations and improve efficiency.

Status: Resolved

13. Misleading comment and error message for minted coin destination

Severity: Informational

In the `BlockProvision` function in `x/custommint/keeper/keeper.go:64-67`, newly minted coins are transferred to the fee collector account. However, the comment in line `64` and the error message in line `66` incorrectly state that the coins are sent directly to the distribution module.

Recommendation

We recommend updating the comment and the error message to accurately reflect that the minted coins are sent to the fee collector account.

Status: Resolved

14. Parameter update logic is duplicated and bypasses the keeper API

Severity: Informational

In `x/custommint/keeper/msg_server.go:44-51`, the `UpdateParams` message handler directly writes module parameters to the key-value store. This implementation

duplicates the logic already present in the `SetParams` function within `x/custommint/keeper/params.go:10-19`.

By implementing the parameter storage logic directly within the message handler, the designated keeper method, `SetParams`, is bypassed. This code duplication can lead to maintenance challenges and potential inconsistencies.

Recommendation

We recommend refactoring the `UpdateParams` message handler to call the `k.SetParams` function. This will centralize the parameter update logic, reduce code duplication, and ensure that any future modifications to parameter handling are applied consistently.

Status: Resolved

15. Updating module parameters lacks event emission

Severity: Informational

The `UpdateParams` message handler function in `x/custommint/keeper/msg_server.go:23-53` allows governance to update the parameters for the custommint module.

However, the function does not emit an event after successfully updating the parameters. This makes it difficult for off-chain services and other monitoring tools to track changes to these critical module parameters in real-time.

Recommendation

We recommend emitting an event after the parameters have been updated. The event should include the newly set parameter values.

Status: Resolved

16. JSON serialization is used for module parameters instead of the application codec

Severity: Informational

In `x/custommint/keeper/params.go:12` and line `31`, the `SetParams` and `GetParams` functions use the standard `json` library for marshaling and unmarshaling the module's parameters.

This approach is inconsistent with the standard practice in the Cosmos SDK, which utilizes the application codec (`cdc`) for serialization. The `Keeper` struct already contains a `cdc` field intended for this purpose.

Recommendation

We recommend updating the `SetParams` and `GetParams` functions to use the application codec (`k.cdc`) for marshaling and unmarshaling.

Status: Acknowledged

The client acknowledges this finding with the note that JSON serialization is intentionally used for the `custommint` parameters. As these parameters are simple, stable, and infrequently changed, JSON provides a readable and lightweight format for internal tools and off-chain scripts, whereas using the app codec would offer no significant benefit in this specific case.

17. Misleading error message in `BurnThreshold` validation

Severity: Informational

In `x/custommint/types/params.go:56-58`, the `Validate` function for `Params` validates the `BurnThreshold` parameter.

The validation correctly checks if `BurnThreshold` is negative with `p.BurnThreshold.IsNegative()`, which allows `0` as a valid value. However, the error message in line `57` incorrectly states that the `BurnThreshold` "cannot be negative or zero". This can cause confusion as the logic and the error message are inconsistent.

Recommendation

We recommend updating the error message to reflect the validation logic accurately. For example, by changing it to "burn threshold cannot be negative".

Status: Resolved

18. Deterministic burn behavior is misrepresented as probabilistic

Severity: Informational

In `x/custommint/keeper/keeper.go:139-140`, the comment indicates a “50% chance to burn all balance from the fee pool,” yet the actual logic deterministically burns tokens on every even-numbered block, `ctx.BlockHeight()%2 == 0`. This means the burn action is not probabilistic but strictly periodic, making the documentation misleading.

Recommendation

We recommend implementing a probability-based burn mechanism aligned with the whitepaper’s design goals.

Status: Resolved