Security Audit Report

Selfchain

v1.0

June 18, 2025

Table of Contents

License

THIS WORK IS LICENSED UNDER A CREATIVE COMMONS ATTRIBUTION-NODERIVATIVES 4.0 INTERNATIONAL 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.

COPYRIGHT OF THIS REPORT REMAINS WITH THE AUTHOR.

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 Ohana Labs Ltd to perform a security audit of the Selfchain Cosmos SDK Implementation.

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/selfchainxyz/selfchain
Commit	`0dddb59bc6fc4c7033a1eb0e2abef0292a6dfb8c`
Scope	All modules were in scope.
Fixes verified at commit	`d7f1f566b965b0950c8ca65e5c966572f42df8c1` 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

The scope of the audit includes the migration of Selfchain to utilize the latest version of the Cosmos SDK, integration of CosmWasm contracts, postponing vesting schedules, and migrating a predefined number of addresses to use new addresses.

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	Medium	-
Level of documentation	Medium	-
Test coverage	Low	Test coverage cannot be determined due to compilation errors.

Summary of Findings

No	Description	Severity	Status
1	Users will not receive vested funds due to insufficient module balances	Critical	Acknowledged
2	New addresses can be forcefully created by sending funds, halting the migration process	Major	Resolved
3	Permanent token freeze due to `cliff > duration`	Major	Acknowledged
4	Unchecked error in the `msg_server_migrate` token transfer operation	Major	Acknowledged
5	Vesting schedules could be incorrectly extended	Major	Resolved
6	Vulnerabilities in outdated dependencies	Major	Resolved
7	Excess tokens remain in the module	Minor	Acknowledged
8	No mechanism to update ACL admin	Minor	Acknowledged
9	Potential division by zero during vesting calculations	Minor	Acknowledged
10	Unhandled errors in the codebase	Minor	Resolved
11	Modules hold unused and excessive permissions	Minor	Resolved
12	Hardcoded `GetParams` ignores stored parameters	Minor	Acknowledged
13	Missing configuration validation in the `UpdateConfig` function	Minor	Acknowledged
14	Missing event emissions	Minor	Acknowledged
15	Genesis accounts can be created with invalid vesting times	Minor	Resolved
16	Centralized admin control over token minting	Minor	Acknowledged
17	Duplicate vesting module import	Informational	Resolved
18	Missing address validation in CLI commands	Informational	Acknowledged
19	Unmodified fee pool distribution	Informational	Resolved
20	Panic usage instead of proper error handling	Informational	Acknowledged

21	Miscellaneous comments	Informational	Resolved

Detailed Findings

1. Users will not receive vested funds due to insufficient module balances

Severity: Critical

In `x/selfvesting/keeper/msg_server_release.go:93`, vested funds will be transferred from the `x/selfvesting` module to the recipient.

However, since the `x/selfvesting` module does not have the actual vested funds, and the `SendCoinsFromModuleToAccount` operation does not implement any error handling, the transfer will fail silently due to insufficient balance.

Consequently, users will not receive their vested funds, even though the vesting schedule indicates that they have already received them, resulting in a loss of funds.

Recommendation

We recommend minting the `vestedCoins` in the `x/selfvesting` module and returning an error if the `SendCoinsFromModuleToAccount` operation fails.

Status: Acknowledged

The client states that these issues relate to deprecated modules that have already served their purpose on the mainnet. They do not plan to fix them at this point.

2. New addresses can be forcefully created by sending funds, halting the migration process

Severity: Major

One of the migration processes involves replacing old accounts with new addresses, where the addresses are defined in `upgrades/v2/handler.go:44-57`. If the new address has an existing account, the upgrade will be halted in lines `190-192` to prevent conflicts.

The issue is that anyone can create accounts for recipients forcefully by sending funds directly to them with `MsgSend`, as seen here. This would cause the validation to fail, thereby returning an error and halting the chain, causing a denial of service attack.

Recommendation

We recommend modifying the implementation so that accounts can still be migrated, even if they hold existing balances.

Status: Resolved

3. Permanent token freeze due to `cliff > duration`

Severity: Major

In `x/selfvesting/keeper/add_beneficiary.go:36`, when creating a vesting position, the cliff time is calculated as `startTime + req.Cliff`. However, the relationship between cliff and duration parameters is not validated, allowing creation of vesting positions where `req.Cliff > req.Duration`.

This is problematic because in `x/selfvesting/keeper/msg_server_release.go:44`, the `getTokenReleaseInfo` function immediately returns `ErrCliffViolation` if `now < vestingInfo.Cliff`. If the cliff period exceeds the vesting duration, it becomes mathematically impossible for a beneficiary to claim their tokens. Specifically, the beneficiary requires the `now >= Cliff` validation to pass the cliff check, but by that time `elapsedPeriod = now - startTime` would exceed `Duration`, meaning the vesting period has already ended before the cliff period completes.

Consequently, this results in tokens being frozen. Users affected by this issue are unable to access their vested tokens through any means, resulting in the loss of their funds.

Recommendation

We recommend validating the cliff period during vesting position creation.

Status: Acknowledged

The client states that these issues relate to deprecated modules that have already served their purpose on the mainnet. They do not plan to fix them at this point.

4. Unchecked error in the `msg_server_migrate` token transfer operation

Severity: Major

In `x/migration/keeper/msg_server_migrate.go`, when processing token migrations, the code transfers instantly released tokens to the user without checking if the transfer succeeded.

Specifically, the `SendCoinsFromModuleToAccount` function returns an error that indicates whether the transfer was successful. However, this error is completely ignored. If the transfer fails for any reason, such as insufficient module balance, invalid recipient address, or other bank module constraints, the code continues execution and marks the migration as processed by setting the token migration record.

This creates a critical state inconsistency where:

- The migration is recorded as complete in the `TokenMigration` store,
- The user never receives their instantly released tokens,
- The tokens remain stuck in the module account,
- The user cannot retry the migration because it is marked as processed.

From the system's perspective, the migration succeeded, but from the user's perspective, they never received their funds. This causes a loss of funds,

Recommendation

We recommend always checking and handling the error returned by `SendCoinsFromModuleToAccount`.

Status: Acknowledged

The client states that these issues relate to deprecated modules that have already served their purpose on the mainnet. They do not plan to fix them at this point.

5. Vesting schedules could be incorrectly extended

Severity: Major

In `upgrades/v2/handler.go:221-230`, all vesting periods of the vesting account are iterated to determine which schedule exceeds the `referenceTime`, and its duration is extended by three months in line `249`.

However, if none of the vesting periods exceed the `referenceTime`, the vesting schedule is still extended. This is because line `223` initially sets the `firstUnvestedIdx` to the zero index value, and will be incorrectly used in line `247`.

Consequently, the vesting schedules that will not exceed the `referenceTime` are incorrectly extended by three months, preventing users from withdrawing their funds on time.

Recommendation

We recommend calling the `return` statement directly if the vesting schedule does not need to be extended.

Status: Resolved

6. Vulnerabilities in outdated dependencies

Severity: Major

In the `go.mod` file, vulnerable versions of dependencies are used, which may allow attackers to exploit and impact the chain's availability, integrity, and confidentiality:

- `github.com/CosmWasm/wasmvm/v2 v2.2.1`
    - GHSA-mx2j-7cmv-353c
- `github.com/cosmos/ibc-go/v8 v8.5.1`
    - GHSA-jg6f-48ff-5xrw

Recommendation

We recommend updating the dependencies to their latest versions.

Status: Resolved

7. Excess tokens remain in the module

Severity: Minor

In `x/migration/keeper/msg_server_migrate.go::77-99`, 1 SLF is minted by the `x/selfvesting` module, but there is a possibility that the amount to transfer to the recipient is less than 1 SLF. This occurs when the `migrationAmount` is less than `instantlyReleased`, causing the excess amount to remain in the module and become stuck.

Recommendation

We recommend minting the required amount of funds within the `if` statement, as indicated by the `migrationAmount` variable.

Status: Acknowledged

The client states that these issues relate to deprecated modules that have already served their purpose on the mainnet. They do not plan to fix them at this point.

8. No mechanism to update ACL admin

Severity: Minor

The `x/migration` module implements an ACL (Access Control List) system, where an admin address controls critical functions such as adding or removing migrators and updating configuration. This admin is set during genesis initialization, but the module provides no mechanism to transfer admin rights to a different address.

Consequently, if the admin private key is compromised, lost, or needs to be rotated for security reasons, there is no way to update it. This creates a single point of failure where loss

of the admin key renders the entire migration module unmanageable. Critical operations, such as adding new migrators or updating the configuration, would become impossible, potentially requiring a hard fork or chain restart to resolve.

Recommendation

We recommend implementing a two-step admin transfer process to allow secure rotation of admin privileges via governance.

Status: Acknowledged

The client states that these issues relate to deprecated modules that have already served their purpose on the mainnet. They do not plan to fix them at this point.

9. Potential division by zero during vesting calculations

Severity: Minor

In `x/selfvesting/keeper/msg_server_release.go:54`, the `getTokenReleaseInfo` function performs a division operation for `amountToVest`, which can cause a runtime panic. If `vestingInfo.Duration` equals zero, the `QuoUint64(0)` operation will trigger a Go runtime panic.

This vulnerability exists because the `AddBeneficiary` function in `x/selfvesting/keeper/add_beneficiary.go` does not validate that the duration is non-zero when creating vesting positions.

Consequently, attackers with migrator privileges or a misconfigured migration could create a vesting position with `Duration = 0`. When any user attempts to release tokens from such a position, the transaction will panic, preventing them from claiming their funds.

Recommendation

We recommend adding validation in the `AddBeneficiary` function to ensure the duration is always greater than zero.

Status: Acknowledged

The client states that these issues relate to deprecated modules that have already served their purpose on the mainnet. They do not plan to fix them at this point.

10. Unhandled errors in the codebase

Severity: Minor

Across the codebase, there are multiple instances of unhandled errors:

- `app/app.go:986`

- `upgrades/v2/handler.go:459, 515, 612, 615, 649, 652, 674, 713, 716`
- `x/migration/keeper/msg_server_migrate.go:99, 106, 109`
- `x/selfvesting/keeper/msg_server_release.go:93`

Some of these errors may cause users to lose funds, which is illustrated in separate issues.

Recommendation

We recommend handling the errors gracefully.

Status: Resolved

11. Modules hold unused and excessive permissions

Severity: Minor

In `app/app.go:219-220`, the `x/migration` and `x/selfvesting` modules are granted minting, burning, and staking permissions. However, only the `x/selfvesting` module uses the minting permission, as seen in `x/migration/keeper/msg_server_migrate.go:77`.

Consequently, the unused permissions granted pose an underlying security risk as an attacker can weaponize them to cause further damage if any of the modules were compromised.

Recommendation

We recommend practicing the principle of least privilege and removing the unused permissions from modules.

Status: Resolved

12. Hardcoded `GetParams` ignores stored parameters

Severity: Minor

In `x/selfvesting/keeper/params.go:9`, the `GetParams` function is implemented incorrectly. This function always returns a new, empty `Params` object instead of reading the actual parameters from storage. The `SetParams` function correctly stores parameters, but the `GetParams` function ignores them entirely.

Consequently, this breaks the params query functionality and any other code that relies on reading stored parameters. Users querying parameters through the CLI or API will always receive empty values, regardless of the actual setting.

Recommendation

We recommend implementing `GetParams` to properly read from the parameter store.

Status: Acknowledged

The client states that these issues relate to deprecated modules that have already served their purpose on the mainnet. They do not plan to fix them at this point.

13. Missing configuration validation in the `UpdateConfig` function

Severity: Minor

In `x/migration/keeper/msg_server_update_config.go:24`, the `UpdateConfig` function stores new configuration values without validating them. This allows setting invalid configuration values such as zero vesting duration, zero minimum migration amount, or cliff periods that exceed duration.

These invalid values might cause issues later during migration execution, it would be better to catch them at configuration time.

Recommendation

We recommend adding validation before storing the configuration to ensure that parameters such as `VestingDuration` and `VestingCliff` fall within the expected bounds.

Status: Acknowledged

The client states that these issues relate to deprecated modules that have already served their purpose on the mainnet. They do not plan to fix them at this point.

14. Missing event emissions

Severity: Minor

The `migration` and `selfvesting` modules lack event emissions for critical operations. There are no events emitted for:

- Token migrations,
- Vesting position creation,
- Token releases,
- Admin operations (add/remove migrator),
- Configuration updates.

This makes it extremely difficult to track module activity, build frontends that react to on-chain events, or create monitoring and alerting systems. External applications cannot efficiently detect when important state changes occur without constantly polling the chain state.

Recommendation

We recommend implementing comprehensive event emissions for all state-changing operations.

Status: Acknowledged

The client states that these issues relate to deprecated modules that have already served their purpose on the mainnet. They do not plan to fix them at this point.

15. Genesis accounts can be created with invalid vesting times

Severity: Minor

In `cmd/selfchaind/cmd/genaccounts.go:102-123`, the genesis account creation command lacks validation for vesting schedule parameters. When creating continuous vesting accounts, the system does not verify that the end time is greater than the start time, which may allow the creation of accounts that vest immediately.

Additionally, there are no checks to prevent setting vesting times in the past or unreasonably far in the future.

Recommendation

We recommend implementing validation logic to ensure the vesting end time exceeds the start time for continuous vesting accounts. Additionally, consider adding reasonable bounds for vesting periods (i.e., not in the past and not exceeding a maximum duration, such as 10 years) to prevent configuration errors during genesis setup.

Status: Resolved

16. Centralized admin control over token minting

Severity: Minor

The `x/migration` module is configured at genesis with a single, immutable admin address that holds exclusive authority to add or remove migrators. These migrators are then permitted to mint new tokens.

This centralization is primarily located in the `x/migration/keeper/` message server files (`msg_server_add_migrator.go`, `msg_server_remove_migrator.go`, `msg_server_update_config.go`), which check against a single admin address.

This creates a single point of failure. For example, if the admin account is compromised, the attacker gains control over token minting. If the account key is lost, the migration functionality becomes permanently disabled, with no recovery mechanism available.

Recommendation

We recommend implementing a governance-based admin for the `x/migration` module.

Status: Acknowledged

The client states that these issues relate to deprecated modules that have already served their purpose on the mainnet. They do not plan to fix them at this point.

17. Duplicate vesting module import

Severity: Informational

In `app/app.go`, the application imports both the standard Cosmos SDK vesting module - `vesting.AppModuleBasic{}` and the custom `selfvesting` module. This creates potential confusion and is unnecessary since the custom `selfvesting` module appears to provide all required vesting functionality.

Having both modules could lead to confusion about which module handles vesting operations and may cause unexpected interactions or duplicate functionality.

Recommendation

We recommend removing the standard vesting module import if the custom `selfvesting` module provides all required functionality. If both are needed, clearly document the purpose and boundaries of each module.

Status: Resolved

18. Missing address validation in CLI commands

Severity: Informational

In the `migration` module's CLI commands in `x/migration/client/cli/tx_*.go`, address arguments passed by users are not validated before creating and broadcasting transactions. While the messages' `ValidateBasic` methods check the creator address, they do not validate other address parameters, such as migrator or destination addresses, which are passed as arguments.

This allows malformed addresses to reach the keeper level, potentially causing confusing error messages or unexpected behavior. Users could waste gas fees on transactions that will fail due to invalid addresses.

Recommendation

We recommend validating the addresses in the CLI commands.

Status: Acknowledged

The client states that these issues relate to deprecated modules that have already served their purpose on the mainnet. They do not plan to fix them at this point.

19. Unmodified fee pool distribution

Severity: Informational

In `upgrades/v2/handler.go:155-161`, the fee pool distribution is retrieved and stored without any modifications. As such, the modification of the fee pool is not required and can be removed.

Recommendation

We recommend removing the unneeded fee pool retrieval and setting.

Status: Resolved

20. Panic usage instead of proper error handling

Severity: Informational

Throughout the `migration` module keeper functions, particularly in `msg_server_migrate.go:21`, `msg_server_add_migrator.go:16`, `msg_server_remove_migrator.go:16`, and `msg_server_update_config.go:16`, the code uses `panic()` for error conditions instead of returning proper errors.

Using panic for error handling violates Cosmos SDK best practices and can cause unexpected transaction terminations without proper error messages being returned.

Recommendation

We recommend replacing all panic statements with proper error returns following Cosmos SDK patterns.

Status: Acknowledged

The client states that these issues relate to deprecated modules that have already served their purpose on the mainnet. They do not plan to fix them at this point.

21. Miscellaneous comments

Severity: Informational

Miscellaneous recommendations can be found below:

- In `cmd/selfchaind/cmd/config.go:10-25`, there is an unused `initSDKConfig` function that duplicates logic from `cmd/selfchaind/cmd/main.go`
- In `cmd/selfchaind/cmd/root.go`, the commented-out `initSDKConfig` function call appears to be dead code that should be removed.

Recommendation

We recommend applying the recommendations to improve the overall code quality and readability.

Status: Resolved