Audit Report

Budget and Farming Cosmos SDK Modules

v1.0

April 4, 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 All in Bits GmbH to perform a security audit of Farming and Budget 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 GitHub repository:

Farming module -
repository: https://github.com/tendermint/farming
audit-ready version: v1.0.0
commit hash: 7ae0fc3cf2636a266bfaa72e7b39fc9a2933d07f
spec: https://github.com/tendermint/farming/tree/v1.0.0/x/farming/spec
docs: https://github.com/tendermint/farming/tree/v1.0.0/docs
swagger docs: v1.0.0
Budget + Farming module Combined Flow Demo
Budget module -
repository: https://github.com/tendermint/budget
audit-ready version: v1.0.0
commit hash: c180df5431018037548c60c6b022333e7c6128ce

spec: https://github.com/tendermint/budget/tree/v1.0.0/x/budget/spec
docs: https://github.com/tendermint/budget/tree/v1.0.0/docs
swagger docs: v1.0.0

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 codebase implements Cosmos SDK modules aimed at the distribution of the planned budget release and at farming (staking and reward functionality).

The budget module allows the creation of a budget through governance proposals. An active budget will transfer funds from the source account to the destination account at a pre-defined rate. Collection and distribution are done at every epoch block length as configured.

The farming module allows the creation of private and public farming plans. Anyone can create a private plan by paying a fee. A public plan is initiated through a governance proposal. There are two types of plans: Fixed amount plans distribute a fixed amount at every epoch, and ratio plans distribute rewards based on a defined ratio from the source account. The farming module allows farmers to stake tokens and receive rewards based on the farming plans associated with the corresponding tokens.

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	Iteration over farming plans in end blocker can be exploited to halt block production	Critical	Resolved
2	Iteration over queued stakings in end blocker can be exploited to halt block production	Critical	Acknowledged
3	Unbounded staking and epoch amount coins can be exploited to halt block production	Critical	Resolved
4	Farming plans are unbounded, which increases gas consumption of plan creation/updates/deletion and may be exploited by an attacker	Major	Resolved
5	Unstaking of last farmer may fail if many epochs have passed	Minor	Resolved
6	Incorrect budget validation may reject valid overlapping budget plans	Minor	Resolved
7	Farming plan total epoch ratio validation incorrectly considers non-overlapping and expired plans	Minor	Resolved
8	Budget collection epochs greater than one create bias in validator payouts	Minor	Acknowledged
9	Budget plans are unbounded, which increases gas consumption with more budget plans	Minor	Acknowledged
10	Farming plan names have no validation of characters or empty strings	Minor	Resolved
11	Inconsistent handling of farming plan termination and rewards if end time equals block time	Minor	Resolved
12	Skipping farming pools if balances are too small may lead to plans that could execute being skipped	Informational	Resolved
13	Event emission within budget modules' begin block is inefficient	Informational	Resolved
14	Farming module allows creation of expired plans, which is inefficient	Informational	Resolved

15	Farming module's end blocker reads stored values multiple times which is inefficient	Informational	Acknowledged
16	Epoch ratio of a farming plan is validated twice, which is inefficient	Informational	Resolved
17	Getting total stakings from storage for each denom of an allocation in farming module's reward allocation is inefficient	Informational	Resolved
18	Budget could be created with source address being equal to destination address	Informational	Resolved
19	Some errors not checked	Informational	Acknowledged

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	-
Level of documentation	High	-
Test coverage	Medium-High	-

Detailed Findings

1. Iteration over farming plans in end blocker can be exploited to halt block production

Severity: Critical

Within the `EndBlocker` of the farming module, iterations occur over all stored public and private farming plans in `x/farming/abci.go:18` and `x/farming/keeper/reward.go:332`. This is problematic since there is to limit on the number of plans that can be created. An attacker could create a large number of private farming plans, eventually leading to block production surpassing Tendermint's propose timeout, at which point block production will halt. The private farming plan creation fee mitigates this issue, but a rational attacker shorting the blockchain's native tokens might still execute this attack.

This issue is even more pronounced since there is currently no functionality to remove expired farming plans.

Recommendation

Firstly, we recommend only processing public farming plans within the `EndBlocker`, while relying on external messages to execute private farming plans. Secondly, we recommend storing aggregate information instead of iterating over all plans in the `EndBlocker`. Thirdly, we recommend enforcing a global upper limit on the number of farming plans that can be created. Fourthly, we recommend making the private plan creation fee exponentially more expensive with the number of active farming plans. Additionally, a deposit per private funding plan could be used together with a function to remove such plans, which would cause a cost of capital to the creator and would encourage removal of no longer needed plans. Lastly, we recommend using distinct storage key prefixes for active and inactive/expired farming plans to remove unnecessary data reads.

Status: Resolved

2. Iteration over queued stakings in end blocker can be exploited to halt block production

Severity: Critical

Within the `EndBlocker` of the farming module, multiple iterations occur over all queued stakings in `x/farming/keeper/staking.go:399`. An attacker could create a large number of stakings in an epoch to cause the end blocker to run too long to finish creation of the block within Tendermint's propose timeout. The cost of this attack is relatively small since the staked amount could be minimal, and staking itself is not gas-intensive.

Recommendation

We recommend removing the processing of queued stakings from the `EndBlocker` and instead triggering it by an external message. Alternatively, a limit to the number of queued stakings could be used to mitigate this issue.

Status: Acknowledged

The client intends to mitigate this issue by increasing the value of delayed gas consumption.

3. Unbounded staking and epoch amount coins can be exploited to halt block production

Severity: Critical

Within the `EndBlocker` of the farming module, multiple iterations occur over a farming plan's `StakingCoinWeights` in `x/farming/keeper/reward.go:414`, and in the case of a fixed plan over `EpochAmount` in `x/farming/keeper/reward.go:377`. These slices are of Cosmos SDK's `Coins` or `DecCoins` types and hence are unbounded. There is currently no validation on the number of coins in each of those slices, and there is no check whether the denom exists, hence an attacker could create a plan with a huge number of coins to cause the end blocker to run too long to finish creation of the block within Tendermint's propose timeout. The cost of this attack is relatively small since many iterations can be caused by a small number of plans. This issue is even more severe with IBC enabled blockchains that can potentially host an unbounded amount of coins. In that case, even ratio plans will cause this issue since the balance of the source account could be positive for a huge number of coins.

Recommendation

We recommend limiting the number of entries in each of the `StakingCoinWeights` and `EpochAmount` slices per plan. A fee could be charged that increases with the number of entries in those slices. We also recommend changing the ratio plan to contain a limited number of coin denoms that will be distributed to prevent issues in IBC enabled chains.

Status: Resolved

4. Farming plans are unbounded, which increases gas consumption of plan creation/updates/deletion and may be exploited by an attacker

Severity: Major

The farming module iterates over all stored plans in several places in the codebase, e. g. in the `ValidateTotalEpochRatio` function in `x/farming/keeper/msg_server.go`, which is called on private and public plan creation/modification/deletion. The plans read from

storage include expired plans. There is currently no mechanism to remove expired private plans. This implies that those messages will increase in gas consumption with a growing number of plans (including private plans) stored.

This is problematic since an attacker can create a big number of private plans, making any creation of further plans as well as updates/removals of existing plans prohibitively expensive, up to the point where the transaction will run out of gas.

Recommendation

We recommend using different storage key prefixes for private and public farming plans. Since private farming plans can't use the same source account, their prefix should include the source account to minimize the number of iterations. We also recommend automatically removing expired plans or moving them under a different storage prefix.

Status: Resolved

5. Unstaking of last farmer may fail if many epochs have passed

Severity: Minor

During the `Unstake` function, `DeleteAllHistoricalRewards` is called if the last farmer of the denom removes their total stake. That function contains an unbounded iteration over all historical rewards for the staked denom in `x/farming/keeper/reward.go:47`. If many epochs have passed, this message will run out of gas, and the unstake will revert.

We consider this to be a minor issue since the last farmer can work around the issue by unstaking all but 1 token.

Recommendation

We recommend setting a limit to the number of historical rewards being deleted during an unstake message.

Status: Resolved

6. Incorrect budget validation may reject valid overlapping budget plans

Severity: Minor

The `ValidateBudgets` function in `x/budget/types/params.go:72` is used to validate that the total rate for a funding source does not exceed 1 at any point. The objective is to ensure that there is enough balance in the source account to satisfy all budget plans at any time.

The implemented logic is too restrictive though since it aggregates all budgets that overlap at any time, even if they are sequential.

To illustrate this, imagine a budget A in date range 1-4, budget B in 1-2, and budget C in 3-4, all using the same source. The current implementation sums up the rate for all those budgets, even though B and C never overlap.

This leads to rejection of budgets even if the overlaps would at no point cause the rate to go over 1.

Recommendation

We recommend enhancing the logic to only consider overlapping budgets when calculating the total rate for each time slot.

Status: Resolved

7. Farming plan total epoch ratio validation incorrectly considers non-overlapping and expired plans

Severity: Minor

The `ValidateTotalEpochRatio` function enforces that the total ratio across all plans for the same farming pool is less than or equal to 1 in `x/farming/types/plan.go:251`. The validation function is called with all farming plans though in `x/farming/keeper/proposal_handler.go:31`, which implies that the total ratio is computed including plans that do not overlap and expired plans.

Recommendation

We recommend changing the logic to only consider overlapping plans when validating the total epoch ratio. One option could be to not store ratios, but rather compute ratios at reward distribution by dividing individual plan weights by the sum of all weights.

Status: Resolved

8. Budget collection epochs greater than one create bias in validator payouts

Severity: Minor

Budgets in the budget module are collected from a `SourceAddress`, which may be set to Cosmos Hub's `FeeCollector` module account, which collects gas fees and part of the ATOM inflation. In that case, the proposer of the block receives smaller fees during that block. Budgets are collected once every `params.EpochBlocks` through

`x/budget/keeper/budget.go:15`. That implies that some block producers receive lower rewards than others.

Since Tendermint uses a weighted round-robin algorithm to assign block producers, this may lead to a situation where some producers will consistently receive fewer rewards than others, which might make certain validator slots less attractive than others and may lead to incentive issues.

Additionally, in the extreme case where the total budget rate equals 1, the proposer will receive no rewards, which may lead to blocks being intentionally not produced..

Recommendation

We recommend requiring `params.EpochBlocks` to be either 0 or 1 to remove any potential bias in fee distribution. We also recommend restricting the total budget rate for the `FeeCollector` module account to a value that always leaves a residual reward for the proposer.

Status: Acknowledged

The client updated the documentation to clarify these implications.

9. Budget plans are unbounded, which increases gas consumption with more budget plans

Severity: Minor

The budget module collects all budgets within begin block by iterating over all stored plans in `x/budget/keeper/budget.go:13-21`.

The plans read from storage include expired plans. There is currently no mechanism to remove expired plans.

This may become problematic if a large number of plans are registered, as the computation performed within begin block increases linearly with the number of plans.

We consider the severity of this issue to be only minor since budget plans can only be added through governance.

Recommendation

We recommend adding a limit to the number of budget plans. We also recommend automatically removing expired plans or moving them under a different storage prefix to keep iterations over storage to a minimum.

Status: Acknowledged

10. Farming plan names have no validation of characters or empty strings

Severity: Minor

Farming plan names have no validation that restricts the character set or enforces a minimum length within the plan validation function in `x/farming/types/plan.go:164`. Budget plans do have a validation for characters and non-empty names in `x/budget/types/budget.go:80`.

This allows empty plan names as well as names with invisible or control characters, which may open the possibility for an attacker to create plans that look as if they have the same name, tricking users.

Recommendation

We recommend adding the budget module's plan name validation to farming plans.

Status: Resolved

11. Inconsistent handling of farming plan termination and rewards if end time equals block time

Severity: Minor

In the rare event in which a farming plan's end-time falls exactly on the block time, no rewards will be allocated due to the condition in `x/farming/types/plan.go:365`, but the plan will only be terminated in the next epoch due to the condition in `x/farming/abci.go:19`. This is counter-intuitive and inconsistent since in all other instances the end time of a plan is exclusive.

Recommendation

We recommend changing the condition in `x/farming/abci.go:19` to also terminate a plan if the end time equals the block time.

Status: Resolved

12. Skipping farming pools if balances are too small may lead to plans that could execute being skipped

Severity: Informational

During farming pool allocation, all plans are skipped in `x/farming/keeper/reward.go:375` if the balance of just one denomination of a farming pool is smaller than the amount that should be distributed.

Imagine we have a ratio plan with 10 % distribution of token A, and a fixed amount plan with 10 token B, both from the same source account. If the source account only has 5 token B at allocation, the 10% ratio plan of token A would also be skipped, even though it could be executed.

Recommendation

We recommend executing plans proportionally to the available funds or only skipping plans that contain denominations that have too small balances.

Status: Resolved

13. Event emission within budget modules' begin block is inefficient

Severity: Informational

The budget module is emitting several events with the same information within begin block in `x/budget/keeper/budget.go:60`, which is inefficient. This information only changes when budget plans change.

Recommendation

We recommend emitting the `name`, `destination_address`, `source_address`, and `rate` attributes on plan creation/updates, and only emitting the `name` and `amount` when budgets are distributed.

Status: Resolved

14. Farming module allows creation of expired plans, which is inefficient

Severity: Informational

The farming module allows creation of expired plans since there is no assertion in the `HandlePublicPlanProposal` function in `x/farming/keeper/proposal_handler.go:12` that a plan's `EndTime` is in the future. Adding expired plans is inefficient.

Recommendation

We recommend rejecting proposals with expired plans.

Status: Resolved

15. Farming module's end blocker reads stored values multiple times which is inefficient

Severity: Informational

The following stored values are read multiple times within the farming module's `EndBlocker`, which is inefficient:

- Farmers staking for a denom is read in `x/farming/keeper/reward.go:223`, `x/farming/keeper/staking.go:398` and `x/farming/keeper/reward.go:184`
- The current epoch is read in `x/farming/keeper/reward.go:228` and `x/farming/keeper/reward.go:413`

Recommendation

We recommend reading the staking only once and passing it into/out of functions that require the value.

Status: Acknowledged

16. Epoch ratio of a farming plan is validated twice, which is inefficient

Severity: Informational

In `x/farming/types/plan.go:284` and `x/farming/types/msgs.go:128`, the epoch ratio of a farming plan is validated twice to be less than or equal to one, which is inefficient.

Recommendation

We recommend removing the second validation in `x/farming/types/plan.go:284`.

Status: Resolved

17. Getting total stakings from storage for each denom of an allocation in farming module's reward allocation is inefficient

Severity: Informational

During the `AllocateRewards` function, `GetTotalStakings` is called in `x/farming/keeper/reward.go:417` for every denom of every allocation, potentially loading the same data many times from storage. This is inefficient.

Recommendation

We recommend caching the total stakings per denom as done with `farmingPoolBalances` in `x/farming/keeper/reward.go:323`.

Status: Resolved

18. Budget could be created with source address being equal to destination address

Severity: Informational

The `Validate` function for a budget in `x/budget/types/budget.go:35` allows a budget with a source address that equals its destination address. While not a security issue, this would be inefficient.

Recommendation

We recommend adding a check to the `Validate` function that ensures that source and destination addresses are different.

Status: Resolved

19. Some errors not checked

Severity: Informational

In a few places in the codebase, errors are not checked. That goes against best practice. Instances are:

- `x/budget/types/budget.go:21`
- `x/budget/types/params.go:51`
- `x/farming/keeper/plan.go:228`
- `x/farming/keeper/reward.go:446-447`
- `x/farming/types/params.go:63`
- `x/farming/types/plan.go:73`
- `x/farming/types/plan.go:84`

Recommendation

We recommend checking all errors.

Status: Acknowledged

Threat Model Analysis

Methodology

Process Applied

The process performed to analyze the system for potential threats and build a comprehensive model is based on the approach first pioneered in Microsoft in 1999 that has developed into the STRIDE model (https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20).
Whilst STRIDE is aimed at traditional software systems, it is generic enough to provide a threat classification suitable for blockchain Layer 1 applications with little adaptation (see below).
The result of the STRIDE classification has then been applied to a risk management matrix with simple counter-measures and mitigations suitable for blockchain applications.

STRIDE Interpretation in the Blockchain Context

STRIDE was first designed for closed software applications in permissioned environments with limited network capabilities. However, the classification provided can be adapted to blockchain systems with small adaptions. The below table highlights a blockchain-centric interpretation of the STRIDE classification:

Spoofing	In a blockchain context, the authenticity of communications is built into the underlying cryptographic public key infrastructure. However, spoofing attack vectors can occur at the off-chain level and within a social engineering paradigm. An example of the former is a Sybil attack where an actor uses multiple cryptographic entities to manipulate a system (wash-trading, auction smart contract manipulation, etc.). The latter usually consists of attackers imitating well-known actors, for instance, the creation of an impersonation token smart contract with a malicious implementation.
Tampering	Similarly to spoofing, tampering of data is usually not directly relevant to blockchain data itself due to cryptographic integrity. M data but is a more subtle attack vectors exist though. One example of this is a supply chain attack that manages to inject malicious code or substitute trusted software that interacts with the blockchain (node software, wallets, libraries).
Repudiation	Repudiation, i.e. the ability of an actor to deny that they have taken an action is usually not relevant at the

	transaction level of blockchains. However, it makes sense to maintain this category, since it may apply to additional software used in blockchain applications, such as user-facing web services. An example is the claim of a loss of a private key and hence assets.
Information Disclosure	Information disclosure has to be treated differently at the blockchain layer and the off-chain layer. Since blockchain state is inherently public in most systems, information leakage here relates to data that is discoverable on the blockchain, even if it should be protected. Predictable random number generation could be classified as such, in addition to simply storing private data on the blockchain. In some cases, information in the mempool (pending/unconfirmed transactions) can be exploited in front-running or sandwich attacks. At the off-chain layer, the leakage of private keys is a good example of operational threat vectors.
Denial of Service	Denial of service threat vectors translate directly to blockchain systems at the infrastructure level. At the smart contract or protocol layer, there are more subtle DoS threats, such as unbounded iterations over data structures that could be exploited to make certain transactions unexecutable.
Elevated Privileges	Elevated privilege attack vectors directly translate to blockchain services. Faulty authorization at the smart contract level is only one example where users might obtain access to functionality that should not be accessible to them.

Assets

The following describes assets that may be valuable to an attacker or other stakeholders of the system.

Native tokens

Tokens of any denomination (e.g. ATOM, UST, or JUNO) of type `sdk.Coin`, potentially IBC-enabled, which are used for budget collection, staking, and farming rewards are considered low-level security assets. An attacker finds those valuable since many tokens can be sold on the market or used to create income in other ways.

Accounts

Budget source, destination, farming fee collector, and farming termination accounts may be either protocol or externally owned. Staking reserve, farming pool, and reward reserve accounts are protocol owned. An attacker finds access to those accounts valuable since that would allow them to re-distribute tokens held in the accounts.

Private keys controlling external accounts

In the case of externally owned accounts, some entity controls a private key associated with the account. An attacker finds access to a private key valuable since that gives them control over funds and permissions the account holds.

Authorization to execute permissioned functions

In both budget and farming modules, governance has special permissions to create (and modify/delete) public budget and farming plans. An attacker finds access to governance permissions valuable since that allows them to create/modify/delete plans.

Plans

Budget, as well as farming plans, specify values that control budget and farming distribution. An attacker finds write access to plans valuable since that allows them to redirect funds in existing plans, create new plans to distribute funds to the attacker or delete plans to increase the share of funds the attacker gets.

Integrity of allocation/distribution mechanism

An attacker may try to game the allocation/distribution mechanism, e. g. by staking and unstaking around reward distribution, harvesting multiple times, or staking with many identities.

Liveness of the protocol

An attacker may find it valuable to attack the liveness of the protocol (e. g. prevent budget distribution, staking, unstaking, or harvesting), which could allow the attacker to profit from shorting the native tokens or extort stakeholders to stop the attack.

Liveness of the underlying blockchain

An attacker may find it valuable to attack the liveness of the underlying blockchain, which again could allow the attacker to profit from shorting the native tokens or extort stakeholders to stop the attack.

Stakeholders/Potential Threat Actors

Traders/token holders

Interested in entering short/long positions in the tokens used by the protocol. May attack the protocol to benefit from loss of trust.

Farmers/users

Farmers that stake/harvest/unstake tokens. May try to game the distribution mechanism.

Private farming plan creators

Creators of private farming plans. May try to use the system in unintended ways.

Community

Parties benefitting or enduring losses due to farming plans. May try to support plans or prevent plans from being enacted.

Voters/governance

Controls the change of parameters and hence the creation, modification, and deletion of budget and public farming plans. Part of the community.

Admins

N/A in either budget or farming modules.

Validators

Block producers of the underlying blockchain. May try to block plans that are potentially leading to lower rewards for validators.

Threat Model

STRIDE Classification

The following threat vectors have been identified using the STRIDE classification, grouped by components of the system.

	Spoofing	**Tampering**	**Repudiation**	**Information Disclosure**	**Denial of Service**	**Elevated Privileges**
Externally owned account	Lost claims	Pharming/ phishing/s social engineerin g	Compromise d claim	Private key leakage Doxxing/ide ntity disclosure	DOS of infrastruct ure	Compromis ed private key
Cosmos SDK modules	-	Backdoors	-	Spyware	-	‘
Integratio n into Cosmos SDK chain	-	Wrong implement ation	-	-	-	Backdoors
Budget and farming plan creation/ modificati on/deletio n	Malicious plan proposals	Plan modificatio n	-	Private information in plan names Fund visibility	End blocker running out of gas Grief voters	-
Staking/u nstaking/ harvestin g	Sybil attacks	Flash staking Timed staking	-	Farmer-plan association	End blocker running out of gas	-
Advancin g epoch (debug function)	-	-	-	-	-	Enabled debug function

Mitigation Matrix

The following mitigation matrix describes each of the threat vectors identified in the STRIDE classification above, assigning an impact and likelihood and suggesting countermeasures and mitigation strategies. Countermeasures can be taken to identify and react to a threat, while mitigations strategies prevent a threat or reduce its impact or likelihood.

Externally owned account

Threat Vector	Impact	Likelihood	Mitigation	Counter-measures
Lost claim: Attacker claims that they are user and that access to private key has been lost	Medium	Low	Have a clear policy not to refund lost assets or restore privileges	Enforce policy
Pharming/phishing/social engineering: Attacker may manipulate users wallet, user interface/front-end, lure to malicious front-end, manipulate DNS record or use social engineering to trick users/team into signing manipulated transactions transferring funds/permissions	Medium	Medium	Educate users and team, protect DNS records, create awareness and offer blacklists with malicious sites, create activity on social channels to build reputable channels, deploy front-ends on IPFS or other decentralized infrastructure	Monitor all systems, monitor communities and impersonations/malicious copies of official channels, communicate attempted pharming/phishing/social engineering, have processes in place to quickly recover from DNS manipulation, attacks on front-ends
Compromised claim: Attacker claims they are a victim of scapegoating, denying responsibility for their attack	Low	Low	Have a clear policy not to refund lost assets or restore privileges	Enforce policy

Private key leakage: Private keys are accidentally shared, logged	Medium	Medium	Educate users and team, ensure private keys are properly handled in wallet software, use hardware wallets/air-gapped devices, security keys, multi-sigs	-
Doxxing/identity disclosure: Private data such as the off-chain identity of users disclosed	Low	Medium	Educate users and team, no storage of identity/sensible data in databases such as newsletters that link identity to account addresses, follow privacy guidelines and regulations	-
DOS of infrastructure: DOS attack on end-user's device/network or on the blockchain node they interact with	Low	Low	Educate users and team, use firewalls, sentry architecture, load balancers, VPNs	Monitor infrastructure, have processes in place to elastically provision and deploy additional resources
Compromised private key: Private keys may be compromised	Medium	Medium	Educate users and team	-

Cosmos SDK modules

Threat Vector	Impact	Likelihood	Mitigation	Counter-measures
Backdoors: Builder team/deployer may implement back doors into the module that enables rug pulls	High	Low	Internal reviews, audits, ensure process verifies that code has been audited	Monitor upgrade attempts, enforce processes on deployment, offer bug-bounties
Spyware: Builder team/deployer may manipulate code to share private key or private information of users	Medium	Low	Internal reviews, audits, ensure process verifies that code has been audited	Monitor upgrade attempts, enforce processes on deployment, offer bug-bounties

Integration into Cosmos SDK chain

Threat Vector	Impact	Likelihood	Mitigation	Counter-measures
Wrong implementation: Integrator team may implement the module wrongly	High	Low	Provide proper documentation, provide test cases for incorrect module integration, perform reviews, audits, ensure process verifies that code has been audited	Run test cases, offer bug-bounties, run chain-upgrades
Backdoors: Integrator team may build in backdoors that enables rug pulls	High	Low	Internal reviews, audits, ensure process verifies that code has been audited	Monitor upgrade attempts, enforce processes on deployment, offer bug-bounties

Budget and farming plan creation/modification/deletion

Threat Vector	Impact	Likelihood	Mitigation	Counter-measures
Malicious plan proposals: Attacker may create malicious public plan creation/modification/deletion proposals, tricking voters to vote on them	High	Medium	Educate users, deploy user interfaces that show actual data that is voted on, add a sufficiently high deposit for proposals that do not reach quorum, ensure mininum quorum and threshold	Monitor proposals, establish trusted communication channels with voters to discuss proposals
Plan modification: Modification of plans through other modules running on the same chain	High	Low	Use keeper architecture to depend on storage permissions of Cosmos SDK, internal reviews,	Run node with invariant checks enabled that control verify plan integrity,

			audits, ensure process verifies that code has been audited	monitor param changes
Private information in plan names: Plan creators might expose private information accidentally	Low	Low	Educate users, set low limit of plan name length	-
Fund visibility: Visibility of source and destination as well as fund flows may not be desired	Low	Low	Educate users	-
End blocker running out of gas: Attacker or unsuspecting users may create many (public and private) plans that make end blocker run out of gas	High	Medium	Seperate storage prefixes for active, inactive, private and public plans, set upper limit on number of plans, allow removal of plans, increase cost of additional plans exponentially, add a deposit that plan creators will get back after plan deletion to have them endure a cost of capital, see issue 1 above	Monitor gas usage in end blocker, have processes in place to execute chain upgrades quickly if needed
Grief voters: Attacker may create many public plan proposals to exhaust the attention and willingness to participate in governance of voters	Medium	Low	Set a high enough quorum, set a sufficiently high proposal deposit that's lost when quorum is not reached	Adjust deposit for proposals, notify users about important proposals in trusted channels

Staking/unstaking/harvesting

Threat Vector	Impact	Likelihood	Mitigation	Counter-measures
Sybil attacks: Attacker might stake with many identities	None	None	Not found during the audit	-
Flash staking: User stakes and unstakes in the same block	None	None	Not found during audit, prevented by delayed staking mechanism	Monitor staking activity
Timed staking: User stakes directly before distribution in the current epoch, unstakes directly after	None	None	Not found during audit, prevented by delayed staking mechanism	Monitor staking activity
Farmer-plan association: Staking towards individual plans reveals information about farmers	None	None	Not found during audit, since staking is not plan specific, but rather across all plans	-
End blocker running out of gas: Attacker or unsuspecting users may create many queued staking entries, either with the same or with multiple source accounts	High	Medium	Move processing of queued stakings into external calls, call them through external messages, e.g. from a bot, set upper limit on queued staking entries, see issue 2 above	Monitor gas usage in end blocker, have processes in place to execute chain upgrades quickly if needed

Advancing epoch (debug function)

Threat Vector	Impact	Likelihood	Mitigation	Counter-measures
Enabled debug function: advance epoch function may be accidentally enabled in a production chain if compiled with wrong linker flags	High	Low	Provide proper documentation, provide test cases for enabled debug functionality, perform reviews, audits, ensure process verifies that code has been audited	Run test cases, monitor for advance epoch message triggering, enforce processes on deployment