Security Audit Report

Celestia Q2 2025: CIP 31

Last revised 22.04.2025

Authors: Martin Hutle, Tatjana Kirda

Contents

Audit overview

The Project

In April 2025, Celestia engaged Informal Systems to work on a partnership and conduct a security audit of the following items:

1. CIP-31
2. Implementation of the CIP-31 in Cosmos SDK, under the provided PR

Relevant code commits

The audited code was from:

- CIP repository
  - commit hash 39544d04c64568f98d0ba984b01f3fb084139df5
- Cosmos SDK repository
  - commit hash dba8171d9f829d90134b1669468831625ee89b0e

Scope of this report

The audit focused on verifying that the code implementation aligns with the specification outlined in CIP-31. In addition to assessing compliance with the specification, the audit also aimed to identify any potential issues or unintended behaviors introduced by the recent code changes.

Notably, while the `MaxCommissionRate` was included in CIP-31, it was not within the scope of this audit. Additionally, any Cosmos SDK code not covered by the pull request was assumed to be correctly implemented.

Audit plan

The audit was conducted between April 9th, 2025, and April 17th, 2025 by the following personnel:

- Martin Hutle
- Tatjana Kirda

Conclusions

After a thorough review of the submitted PR, we found it to be generally well-implemented and in alignment with the CIP-31 specification. During the audit, we identified three findings: two classified as high severity and one as informational. Full details of these issues can be found on the Findings page.

Audit Dashboard

Target Summary

- **Type:** Protocol and Implementation
- **Platform:** Go
- **Artifacts:**
  - CIP-31
  - CIP-31 implementation in Cosmos SDK

Engagement Summary

- **Dates**: 09.04.2025 - 17.04.2025.
- **Method**: Manual code review

Severity Summary

Finding Severity	Number
Critical	0
High	2
Medium	0
Low	0
Informational	1
Total	3

System Overview

The CIP-31 outlines the incorporation of staking rewards directly into lockup accounts. When a vesting account earns rewards from staking, the system adds the reward amount to the lockup balance and adjusts the daily unlock rate based on the remaining lockup duration (ref).

Account types

A vesting account can be one of the following:

- Continuous vesting account
- Periodic vesting account
- Delayed vesting account
- Permanent vesting account.

CIP-31 is implemented for continuous and delayed vesting accounts only. The creation of periodic and permanent vesting accounts is disabled. Previously created accounts continue to exist, and if a reward is distributed to such an account, it is not locked.

Vesting calculations

`BaseVestingAccount` = (`OV[]`, `DF[]`, `DV[]`, `E`), where

- `OV[i]` is the original vested amount for denomination index `i`; if `OV[i] = 0`, the denomination is not vested.
- `DF[i]` is the delegated free (i.e., unlocked) amount.
- `DV[i]` is the delegated vesting (i.e., locked) amount.
- `E` is the end time of the vesting, after which all coins are unlocked.

`ContinuousVestingAccount` = (`BaseVestingAccount`, `S`), where

- `S` is the (Unix) time the continuous unlocking starts.

`DelayedVestingAccount` = `BaseVestingAccount`.

Further, we denote with `BC[i]` the account balance in the banking module.

`V` is the number of vesting (locked) coins.

`V'` is the number of vested (unlocked) coins.

Continuous vesting account:

`V'[i]` =

- `OV[i]` for `t <= S`
- `(t-S)/(E-S) * OV[i]` for `S < t < E`
- `0` for `t >= E`

`V[i] = OV[i] - V'[i]`

Delayed vesting account:

`V'[i]` =

- `0` for `t < E`
- `OV[i]` for `t >= E`

`V[i] = OV[i] - V'[i]`

Modifying `OV[i]`

If a reward `R[i]` (for denomination `i`) is distributed to the continuous or delayed vesting account, then `OV[i]` is set to

- `OV[i] + R[i]` for `OV[i] > 0`
- `0` for `OV[i] = 0`

In addition, the rewards are transferred to the account balance `BC[i]`.

Threat Model

Account types

Invariant 1: No periodic or permanent vesting account can be created.

Conclusion:

The invariant holds. The provided PR removes the logic responsible for creating periodic and vesting accounts. As a result, any attempt to create these account types now returns an error (code ref). Currently, only continuous vesting accounts and delayed vesting accounts can be created via messages. This restriction is also enforced within the `CreateVestingAccount` method (code ref).

Invariant 2: This list of vesting accounts defined in the system overview is exhaustive.

Conclusion:

The invariant holds. The file `x/auth/vesting/types/vesting_account.go` defines types of `BaseVestingAccount` (code ref), `ContinuousVestingAccount` (code ref), `PeriodicVestingAccount` (code ref), `DelayedVestingAccount` (code ref), and `PermanentLockedAccount` (code ref). The `BaseVestingAccount` implements core functionality that is shared by all vesting account types.

Invariant 3: If a reward is distributed to a periodic or permanent vesting account, the vested amount and the vesting schedule remain unchanged.

Conclusion:

The invariant is maintained through the `nil` implementation of `UpdateSchedule` for both periodic (code ref) and permanent (code ref) vesting accounts. When rewards are distributed, the `UpdateSchedule` method is invoked but returns `nil` for these account types, allowing the standard reward distribution flow to proceed without changing the vesting schedule (code ref).

Reward distribution

Invariant 4: Every time a reward is distributed to a vesting account, the locking schedule is adapted accordingly.

Conclusion:

The invariant does not hold. A user can withdraw rewards and bypass the vesting schedule update by unbonding tokens or setting a non-vesting account as the `WithdrawAddr`. See the threats below for more details.

Threat: Not all sources/mechanisms of rewards are covered.

Conclusion:

This is a legitimate threat. See finding Vesting Schedule Not Updated on Reward Claim After Unbonding.

Threat: Vesting account schedule update can be omitted if the `WithdrawAddrEnabled` parameter is set to true.

Conclusion:

This is a legitimate threat. See finding Vesting Restriction Bypass Through Custom Withdrawal Addresses.

Invariant 5: No other changes of the account balance lead to changes in the vesting schedule.

Conclusion:

The vesting schedule can be affected by three main parameters: `OriginalVesting`, `StartTime`, `EndTime`, and `VestingPeriods`. However, none of the current vesting account types allow direct modification of `StartTime` or `EndTime`, and they cannot be changed after account creation. Additionally, for periodic vesting accounts, there are no other methods that modify `VestingPeriods` after creation. The only mechanism for updating `OriginalVesting` is through the `UpdateSchedule` function, which is implemented for continuous and delayed vesting accounts. This

method is only called within the `withdrawDelegationRewards` function, which further updates the account balance (code ref).

Invariant 6: The vesting schedule that has already been completed cannot be updated.

Conclusion:

The invariant holds. The vesting schedule update is handled by the `UpdateSchedule` function. Implementations for both `ContinuousVestingAccount` (code ref) and `DelayedVestingAccount` (code ref) explicitly check if the current block time is greater than or equal to the end time. If the schedule is completed, they return `nil` without making any changes. The `UpdateSchedule` function always returns `nil` for other vesting account types.

Vesting calculations

Continuous vesting account

Invariant 7: The distribution of locked (vesting) and unlocked (vested) coins is implemented as defined in the specification and the system overview.

Conclusion:

The invariant holds. The calculation of vested coins is handled by the `GetVestedCoins` function (code ref). The `x` calculates how much time has elapsed since vesting started, `y` represents the total duration of vesting, and `s` computes the fraction of vesting time that has passed. For each coin, the original amount is converted to a decimal for precision, multiplied by the vesting scalar `s`, rounded to the nearest integer, and then used to create a new coin that is added to the result.

The calculation of the vesting coins is based on the `GetVestedCoins`, and it is handled in the `GetVestingCoins` function (code ref). The amount of vesting coins is calculated by subtracting the result of `GetVestedCoins` from the `OriginalVesting`.

The specification captures the implementation with close accuracy.

The audited PR does not modify these functions, and the current calculation implementation is assumed to be correct.

Threat: Corner cases `t <= S` or `t >= E` are not considered.

Conclusion: The threat does not hold. The function `GetVestedCoins` handles the calculation of the vested (unlocked) coins (code ref). In case `blockTime.Unix() <= cva.StartTime`, the function will return an empty `vestedCoins`, which is initialized as `var vestedCoins sdk.Coins` (code ref). If the `blockTime.Unix() >= cva.EndTime` original vesting amount will be returned (code ref). The function `GetVestingCoins` calculates the number of locked coins by subtracting the vested coins from the original vesting (code ref).

Threat: `E > S` is not ensured.

Conclusion: The threat does not hold. The function `GetVestedCoins` doesn't explicitly check if the `cva.EndTime - cva.StartTime` is not equal to zero, which can cause panic when the function `Quo` is called (code ref). However, the function ensures beforehand that the `blockTime` is strictly greater than the start time and strictly less than the end time (code ref). This implies that the `cva.StartTime` $<$ `blockTime` $<$ `cva.EndTime`.

Threat: Incorrect handling of `OV[i] = 0` or `nil`.

Conclusion: The threat does not hold. The function `GetVestedCoins` initializes an empty `vestedCoins` variable (code ref). The `GetVestingCoins` function uses `Coins.Sub` (code ref) that handles empty coins correctly (code ref) and returns `Coins{}` if zero or `nil`.

Delayed vesting account

Invariant 8: The distribution of locked (vesting) and unlocked (vested) coins is implemented as defined in the specification and the system overview.

Conclusion:

The invariant holds. The calculation of vested coins is handled by the `GetVestedCoins` function (code ref). If the vesting has passed, the function returns the `OriginalVesting` amount. Otherwise, `nil` is returned.

The calculation of the vesting coins is based on the `GetVestedCoins` function, and it is handled in the `GetVestingCoins` function (code ref). The amount of vesting coins is calculated by subtracting the result of `GetVestedCoins` from the `OriginalVesting`.

The specification captures the implementation with close accuracy.

The audited PR does not modify these functions, and the current calculation implementation is assumed to be correct.

Threat: Incorrect handling of `OV[i] = 0` or `nil`.

Conclusion: The threat does not hold. The function `GetVestedCoins` returns `nil` if the schedule hasn't elapsed (code ref). The `GetVestingCoins` function uses `Coins.Sub` (code ref) that handles empty coins correctly (code ref) and returns `Coins{}` if zero or `nil`.

Modifying `OV[i]`

Invariant 9: Reward distribution is implemented as described in the system overview and adheres to the provided specification.

Conclusion:

The invariant does not hold. The finding Discrepancy Between CIP-31 and Implementation for Denominations Not in Original Vesting emerged during the code review.

Invariant 10: At any time, the currently locked value must be smaller or equal to the liquid funds plus the locked delegations:

```
V[i] <= DV[i] + BC[i]
```

We first check that the invariant is ensured by the mechanism of increasing `OV[i]` by `R[i]`: If `R[i]` is added to `OV[i]` and `BC[i]`, we have for the continuous vesting account:

```
(t-S)/(E-S) * (OV[i] + R[i]) <= (t-S)/(E-S)*OV[i] + R[i] <= DV[i] + BC[i] + R[i]
```

which shows this property holds in theory. We now also need to check the code for correct implementation.

Conclusion:

The invariant holds. When the `withdrawDelegationRewards` function is called, it first updates the vesting schedule through `UpdateSchedule` (code ref) and then transfers coins using `k.bankKeeper.SendCoinsFromModuleToAccount` (code ref). The bank keeper's `SendCoins` function (code ref) internally calls `subUnlockedCoins` (code ref), which uses the `LockedCoins` function (code ref) to determine how many coins are locked. The `LockedCoins` calculation computes the locked coins by subtracting the minimum of vesting coins and `DelegatedVesting` from the original vesting amount (e.g., code ref) based on the current block time and vesting schedule for the specific vesting account type, and then subtracts delegated vesting from this amount (code ref). The check `spendable, hasNeg := sdk.Coins{balance}.SafeSub(locked)` from the `subUnlockedCoins` function (code ref) ensures that the account's balance is greater than or equal to the locked coins and that the remaining spendable amount (balance - locked) is sufficient for the transfer.

Since the locked amount is derived from the vesting schedule and delegated vesting, and balance represents `BC[i]`, this check effectively enforces that `BC[i] >= V[i] - min(V[i], DV[I])`. The expression `BC[i] >= V[i] - min(V[i], DV[I])` can be rewritten as `BC[i] >= V[i] - DV[I]` when `V[i] > DV[I]`, otherwise it's trivially satisfied as `BC[i] >= 0`, which directly rearranges to `V[i] <= DV[i] + BC[I]`. Therefore, the implementation is equivalent to the invariant `V[i] <= DV[i] + BC[I]`.

Threat: Rewards can be withdrawn before the vesting schedule is properly updated in situations where the schedule adjustment should occur.

Conclusion:

The original vesting amount `OV[I]` is only updated if the denomination was part of the original vesting schedule, the account is still within its vesting period, and the vesting account is the correct type.

If these conditions are met, when the rewards are withdrawn in the `withdrawDelegationRewards` function, both bank coins and outstanding rewards are updated atomically (code ref). Additionally, the `BeforeDelegationSharesModified` hook doesn't move the claimed rewards to the user's account since the variable `withdrawNow` is set to false (code ref), nor is the vesting schedule updated (code ref).

However, when rewards are withdrawn after the user no longer has a delegation to the validator, the issue described in the Vesting Schedule Not Updated on Reward Claim After Unbonding finding occurs.

Threat: `OV[I]` can be influenced by factors other than delegation rewards.

Conclusion:

The threat does not hold. The `OV[I]` amount can only be modified via the `UpdateSchedule` method, which is called exclusively from the `withdrawDelegationRewards` function (code ref). For both `PeriodicVestingAccount` and `PermanentLockedAccount`, the OV[i] value cannot be increased at all, as the `UpdateSchedule` function returns `nil`.

Threat: `OV[I]` is decreased due to negative `R[i]`.

Conclusion:

The threat does not hold. The `UpdateSchedule` function uses the `finalRewards` variable as the amount being added to the `OV[I]` (code ref). When `finalRewards` is created, it is guaranteed to be a positive amount, as the `TruncateDecimal` function is used during its creation (code ref). The `TruncateDecimal` function employs the `NewCoin` constructor (code ref), which ensures that the coin amount is both valid (code ref) and non-negative (code ref). Prior to calling `UpdateSchedule`, the `outstanding.Rewards` are added to the final rewards using the `Add` function (code ref). The implementation of the `Add` function notes that `*Add` will never return Coins where one Coin has a non-positive amount* (code ref). For the purposes of this audit, this statement is assumed to be accurate.

Delegations

Note: `x/vesting` does not implement the delegation but only tracks them with respect to vesting.

As mentioned in the documentation of the standard Cosmos SDK, if a validator gets slashed, it might be the case that `DV > 0` even after all funds have been unlocked (`V = 0`). The modification to increase `OV` if rewards are distributed does not alter this scenario compared to the standard Cosmos SDK case.

Findings

Finding	Type	Severity	Status
Vesting Restriction Bypass Through Custom Withdrawal Addresses	Implementation	High	Resolved
Vesting Schedule Not Updated on Reward Claim After Unbonding	Implementation	High	Resolved
Discrepancy Between CIP-31 and Implementation for Denominations Not in Original Vesting	Documentation	Informational	Resolved

Vesting Restriction Bypass Through Custom Withdrawal Addresses

ID	IF-FINDING-001
Severity	High
Impact	2 - Medium
Exploitability	3 - High
Type	Implementation
Status	Resolved

Involved artifacts

- x/distribution/keeper/delegation.go
- x/distribution/keeper/msg_server.go

Description

When the `withdrawDelegationRewards` function is invoked, the vesting schedule is only updated if the `withdrawAddr` is of type `types.VestingAccount` (code ref). This address is derived from the `delAddr` (code ref). However, the `withdrawAddr` can be modified using the `SetWithdrawAddress` method (code ref) if the `WithdrawAddrEnabled` parameter is set to true, potentially bypassing the vesting logic.

Problem Scenarios

Users with vesting accounts can set a regular account they control as their withdrawal address through the `SetWithdrawAddr` function. When rewards generated from delegated vesting tokens are claimed, they are sent directly to this regular account without any vesting restrictions.

Recommendation

After sharing the finding with the client, the issue was resolved by modifying the account type check to verify whether the delegator account is a vesting account. If it is, the vesting schedule is updated and the rewards are sent to the vesting account. If it is not, the rewards are sent to the `WithdrawAddr`. Additionally, the PR disabled the setting of the withdraw address for vesting accounts.

Status

Resolved

The Celestia team outlined the following scenario: A vesting account on Mainnet initially sets its withdraw address to a separate address. After CIP-31 activates, the account attempts to update its withdraw address to its own address but encounters the error: `sdkerrors.ErrInvalidRequest.Wrapf("cannot set withdraw address for vesting account %s", delegatorAddress)`. It was concluded that this behavior is acceptable, as rewards will continue to be withdrawn to the vesting account by default.

Vesting Schedule Not Updated on Reward Claim After Unbonding

ID	IF-FINDING-002
Severity	High
Impact	2 - Medium
Exploitability	3 - High
Type	Implementation
Status	Resolved

Involved artifacts

- README.md
- x/distribution/keeper/hooks.go
- x/staking/keeper/delegation.go
- x/distribution/keeper/delegation.go
- x/distribution/keeper/keeper.go
- x/distribution/keeper/msg_server.go

Description

For all vesting accounts, the owner of the vesting account is able to delegate and undelegate from validators (ref).

The vesting account can unbond tokens, which triggers the `BeforeDelegationSharesModified` hook (code ref) and sets the rewards as the user's outstanding rewards (code ref). However, the vesting schedule is not updated as part of this sequence.

The rewards are then withdrawn by calling the `WithdrawDelegationRewards` function (code ref), which does not subject the rewards to vesting if the user doesn't have a delegation anymore to the validator (code ref).

Problem Scenarios

The user can undelegate from a validator and then claim the rewards by utilizing the `WithdrawDelegationRewards` function. By doing so, the user bypasses the vesting schedule update and claims the rewards without them being incorporated into the vesting rewards.

Recommendation

After sharing the finding with the client, the issue was resolved by adjusting the flow for cases where the user no longer has a delegation to the validator. The fix involved checking whether the delegator account is a vesting account. If it is, the vesting schedule is updated and the rewards are sent to the vesting account. If it is not, the rewards are sent to the withdraw address.

Status

Resolved

Discrepancy Between CIP-31 and Implementation for Denominations Not in Original Vesting

ID	IF-FINDING-003
Severity	Informational
Impact	1 - Low
Exploitability	0 - None
Type	Documentation
Status	Resolved

Involved artifacts

- x/auth/vesting/types/vesting_account.go
- CIP-31

Description

In both `ContinuousVestingAccount` (code ref) and `DelayedVestingAccount` (code ref) implementations, the `UpdateSchedule` function contains logic that excludes the denominations that weren't originally vesting **from being added to the vesting schedule.

Problem Scenarios

When a vesting account with `originalVesting[FOO] = 0` receives rewards in `FOO` denomination, these rewards are not added to the vesting schedule. This exception is not documented in the CIP-31 specification.

Recommendation

Our recommendation is to update the specification to explicitly document that only reward denominations matching those in the original vesting schedule will be subject to vesting restrictions.

Status

Resolved

Appendix: Vulnerability classification

For classifying vulnerabilities identified in the findings of this report, we employ the simplified version of Common Vulnerability Scoring System (CVSS) v3.1, which is an industry standard vulnerability metric. For each identified vulnerability we assess the scores from the *Base Metric Group*, the Impact score, and the Exploitability score. The *Exploitability score* reflects the ease and technical means by which the vulnerability can be exploited. That is, it represents characteristics of the *thing that is vulnerable*, which we refer to formally as the *vulnerable component*. The *Impact score* reflects the direct consequence of a successful exploit, and represents the consequence to the *thing that suffers the impact*, which we refer to formally as the *impacted component*. In order to ease score understanding, we employ CVSS Qualitative Severity Rating Scale, and abstract numerical scores into the textual representation; we construct the final *Severity score* based on the combination of the Impact and Exploitability sub-scores.

As blockchains are a fast evolving field, we evaluate the scores not only for the present state of the system, but also for the state that deems achievable within 1 year of projected system evolution. E.g., if at present the system interacts with 1-2 other blockchains, but plans to expand interaction to 10-20 within the next year, we evaluate the impact, exploitability, and severity scores wrt. the latter state, in order to give the system designers better understanding of the vulnerabilities that need to be addressed in the near future.

Impact Score

The Impact score captures the effects of a successfully exploited vulnerability on the component that suffers the worst outcome that is most directly and predictably associated with the attack.

ImpactScore	Examples
High	Halting of the chain; loss, locking, or unauthorized withdrawal of funds of many users; arbitrary transaction execution; forging of user messages / circumvention of authorization logic
Medium	Temporary denial of service / substantial unexpected delays in processing user requests (e.g. many hours/days); loss, locking, or unauthorized withdrawal of funds of a single user / few users; failures during transaction execution (e.g. out of gas errors); substantial increase in node computational requirements (e.g. 10x)
Low	Transient unexpected delays in processing user requests (e.g. minutes/a few hours); Medium increase in node computational requirements (e.g. 2x); any kind of problem that affects end users, but can be repaired by manual intervention (e.g. a special transaction)
None	Small increase in node computational requirements (e.g. 20%); code inefficiencies; bad code practices; lack/incompleteness of tests; lack/incompleteness of documentation

Exploitability Score

The Exploitability score reflects the ease and technical means by which the vulnerability can be exploited; it represents the characteristics of the vulnerable component. In the below table we list, for each category, examples of actions by actors that are enough to trigger the exploit. In the examples below:

- *Actors* can be any entity that interacts with the system: other blockchains, system users, validators, relayers, but also uncontrollable phenomena (e.g. network delays or partitions).
- *Actions* can be
  - *legitimate*, e.g. submission of a transaction that follows protocol rules by a user; delegation/redelegation/bonding/unbonding; validator downtime; validator voting on a single, but alternative block; delays in relaying certain messages, or speeding up relaying other messages;
  - *illegitimate*, e.g. submission of a specially crafted transaction (not following the protocol, or e.g. with large/incorrect values); voting on two different alternative blocks; alteration of relayed messages.
- We employ also a *qualitative measure* representing the amount of certain class of power (e.g. possessed tokens, validator power, relayed messages): *small* for $< 3\%$; *medium* for 3-10%; *large* for 10-33%, *all* for $>33\%$. We further quantify this qualitative measure as relative to the largest of the system components. (e.g. when two blockchains are interacting, one with a large capitalization, and another with a small capitalization, we employ *small* wrt. the number of tokens held, if it is small wrt. the large blockchain, even if it is large wrt. the small blockchain)

ExploitabilityScore	**Examples**
High	illegitimate actions taken by a small group of actors; possibly coordinated with legitimate actions taken by a medium group of actors
Medium	illegitimate actions taken by a medium group of actors; possibly coordinated with legitimate actions taken by a large group of actors
Low	illegitimate actions taken by a large group of actors; possibly coordinated with legitimate actions taken by all actors
None	illegitimate actions taken in a coordinated fashion by all actors

Severity Score

The severity score combines the above two sub-scores into a single value, and roughly represents the probability of the system suffering a severe impact with time; thus it also represents the measure of the urgency or order in which vulnerabilities need to be addressed. We assess the severity according to the combination scheme represented graphically below.

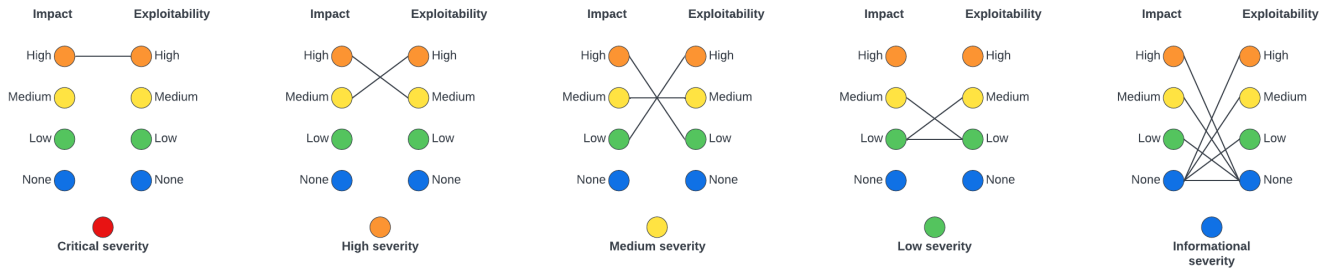

Figure 1: Severity classification

As can be seen from the image above, only a combination of high impact with high exploitability results in a Critical severity score; such vulnerabilities need to be addressed ASAP. Accordingly, High severity score receive vulnerabilities with the combination of high impact and medium exploitability, or medium impact, but high exploitability.

SeverityScore	Examples
Critical	Halting of chain via a submission of a specially crafted transaction
High	Permanent loss of user funds via a combination of submitting a specially crafted transaction with delaying of certain messages by a large portion of relayers
Medium	Substantial unexpected delays in processing user requests via a combination of delaying of certain messages by a large group of relayers with coordinated withdrawal of funds by a large group of users
Low	2x increase in node computational requirements via coordinated withdrawal of all user tokens
Informational	Code inefficiencies; bad code practices; lack/incompleteness of tests; lack/incompleteness of documentation; any exploit for which a coordinated illegitimate action of all actors is necessary

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability, etc.) set forth in the associated Services Agreement. This report provided in connection with the Services set forth in the Services Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement.

This audit report is provided on an "as is" basis, with no guarantee of the completeness, accuracy, timeliness or of the results obtained by use of the information provided. Informal has relied upon information and data provided by the client, and is not responsible for any errors or omissions in such information and data or results obtained from the use of that information or conclusions in this report. Informal makes no warranty of any kind, express or implied, regarding the accuracy, adequacy, validity, reliability, availability or completeness of this report. This report should not be considered or utilized as a complete assessment of the overall utility, security or bugfree status of the code.

This audit report contains confidential information and is only intended for use by the client. Reuse or republication of the audit report other than as authorized by the client is prohibited.

This report is not, nor should it be considered, an "endorsement", "approval" or "disapproval" of any particular project or team. This report is not, nor should it be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts with Informal to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor does it provide any indication of the client's business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should it be leveraged as investment advice of any sort.

Blockchain technology and cryptographic assets in general and by definition present a high level of ongoing risk. Client is responsible for its own due diligence and continuing security in this regard.