Zellic

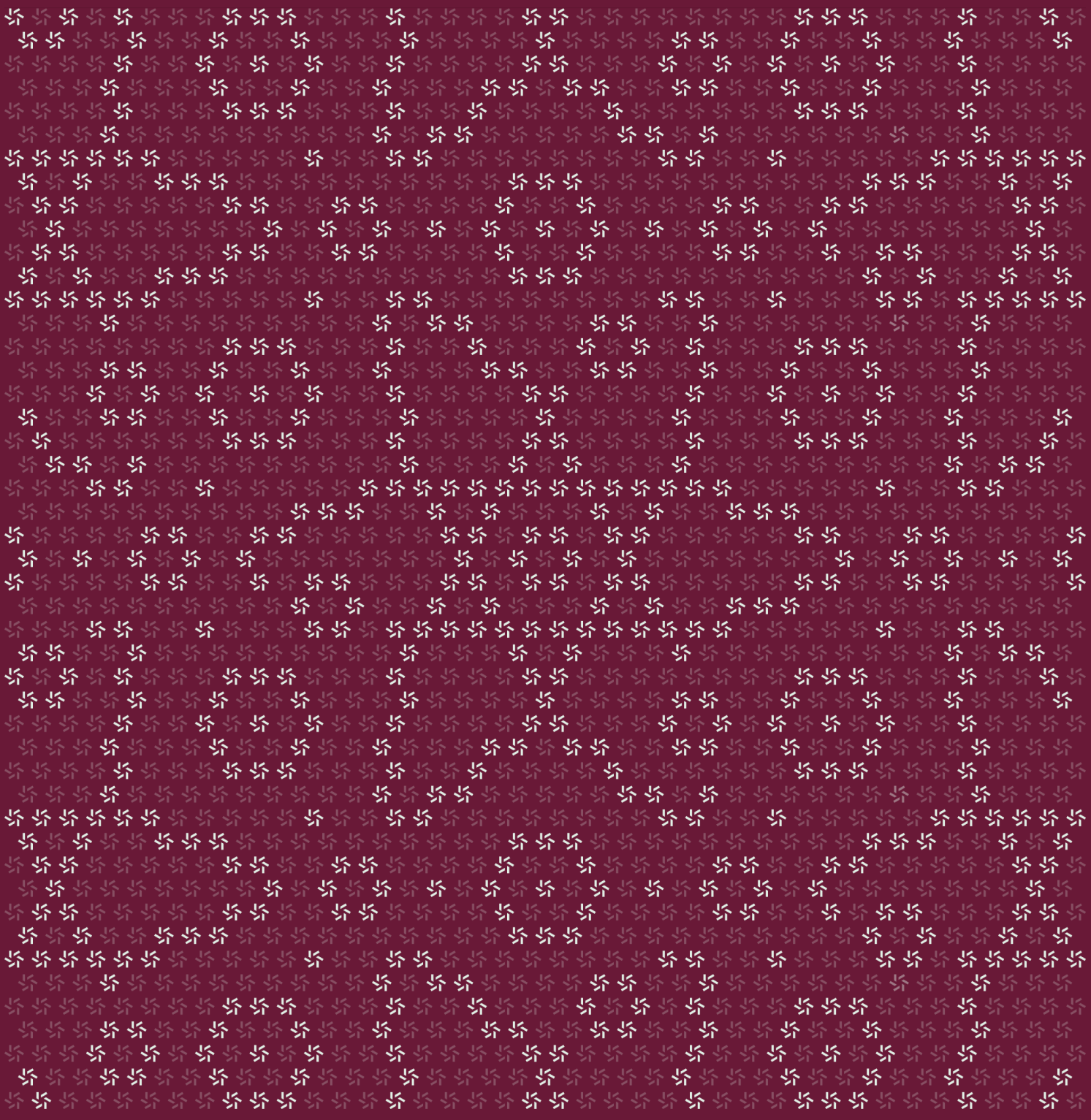
Prepared for
Kristov Atlas
AtomOne

Prepared by
Frank Bachman
Varun Verma
Zellic

July 4, 2025

All in Bits

Cosmos Application Security Assessment



Contents

About Zellic **4**

1. Overview **4**

1.1. Executive Summary 5

1.2. Goals of the Assessment 5

1.3. Non-goals and Limitations 5

1.4. Results 5

2. Introduction **6**

2.1. About All in Bits 7

2.2. Methodology 7

2.3. Scope 9

2.4. Project Overview 10

2.5. Project Timeline 10

3. Detailed Findings **11**

3.1. Proposals can be passed without quorum and threshold requirements being met 12

3.2. Zero `TargetBlockUtilization` disables fee market 14

3.3. The `min_quorum > max_quorum` flips interpolation 16

3.4. Silent unpack failure lowers quorum 17

4. Discussion **18**

4.1. Test suite 19

5. Threat Model **19**

5.1. Component: Dynamic quorum (x/gov) 20

5.2. Component: Fee market (x/feemarket) 21

5.3. Component: Dynamic deposit (x/gov) 21

6. Assessment Results **22**

6.1. Disclaimer 23

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for AtomOne from June 11th to June 23rd, 2025. During this engagement, Zellic reviewed All in Bits's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Are the accepted ADRs implemented according to the documentation?
- Does the AtomOne daemon use the SDK securely?
- Does any new functionality introduce additional vulnerabilities (mainly focusing on dynamic quorum and minimum deposit throttler)?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody
- IBC and ICS functionality

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

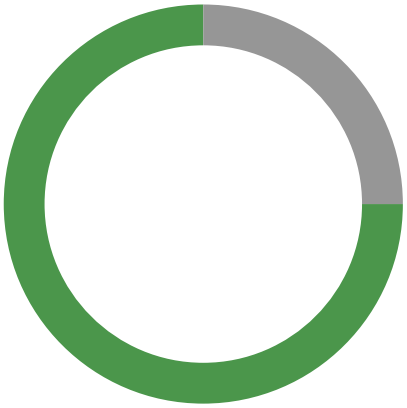
1.4. Results

During our assessment on the scoped All in Bits modules, we discovered four findings. No critical issues were found. Three findings were of low impact and the remaining finding was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of AtomOne in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	3
Informational	1



2. Introduction

2.1. About All in Bits

AtomOne contributed the following description of All in Bits:

> AtomOne is a community-driven, constitutionally governed blockchain designed to prioritize security, decentralization, and innovation within the Cosmos ecosystem. Serving as a minimal fork of the Cosmos Hub, it supports IBC and ICS for scalable interchain solutions.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the modules.

Nondeterminism. Nondeterminism is a leading class of security issues on Cosmos. It can lead to consensus failure and blockchain halts. This includes but is not limited to vectors like wall-clock times, map iteration, and other sources of undefined behavior (UB) in Go.

Arithmetic issues. This includes but is not limited to integer overflows and underflows, floating-point associativity issues, loss of precision, and unfavorable integer rounding.

Complex integration risks. Several high-profile exploits have been the result of unintended consequences when interacting with the broader ecosystem, such as via IBC (Inter-Blockchain Communication Protocol). Zellic will review the project's potential external interactions and summarize the associated risks. If applicable, we will also examine any IBC interactions against the ICS Specification Standard to look for inconsistencies, flaws, and vulnerabilities.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather

than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped modules itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

All in Bits Modules

Type	Go
Platform	Cosmos L1
Target	PR 135
Repository	https://github.com/atomone-hub/atomone ↗
Version	`0c62f999b1d1ad24128a3f934e8cfeb9104b9b34`
Programs	`https://github.com/atomone-hub/atomone/pull/135`
Target	PR 69
Repository	https://github.com/atomone-hub/atomone ↗
Version	`9989cfd58e3fad1618c457085d77099e8659fb2a`
Programs	`https://github.com/atomone-hub/atomone/pull/69`

Target	PR 114
Repository	https://github.com/atomone-hub/atomone ↗
Version	`0ff0dad957450e9186c3fb29752049e82b50d6d2`
Programs	`https://github.com/atomone-hub/atomone/pull/114`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 2.4 person-weeks. The assessment was conducted by two consultants over the course of 1.8 calendar weeks.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io ↗

Chad McDonald
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Frank Bachman
Engineer
frank@zellic.io ↗

Varun Verma
Engineer
varun@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

June 11, 2025	Kick-off call
June 11, 2025	Start of primary review period
June 23, 2025	End of primary review period

3. Detailed Findings

3.1. Proposals can be passed without quorum and threshold requirements being met

Target	x/gov/keeper/tally.go		
Category	Business Logic	**Severity**	Medium
Likelihood	Low	**Impact**	Low

Description

The `Tally` helper in the keeper is used to iterate over votes and return the tally results of a proposal based on the voting power.

The quorum and threshold required for the proposal are fetched through the `getQuorumAndThreshold` helper. The previous implementation for `getQuorumAndThreshold` would iterate over all messages and return the highest quorum/threshold, which would be required for the entire proposal to pass.

Now, at the time of writing, the returned quorum and threshold are based on priority ordering:

```
func (keeper Keeper) getQuorumAndThreshold(ctx sdk.Context,
    proposal v1.Proposal) (quorum sdk.Dec, threshold sdk.Dec) {
    params := keeper.GetParams(ctx)
    kinds := keeper.ProposalKinds(proposal)
    if kinds.HasKindConstitutionAmendment() {
        quorum = keeper.GetConstitutionAmendmentQuorum(ctx)
        threshold =
    sdk.MustNewDecFromStr(params.ConstitutionAmendmentThreshold)
        return
    } // [0]
    if kinds.HasKindLaw() {
        quorum = keeper.GetLawQuorum(ctx)
        threshold = sdk.MustNewDecFromStr(params.LawThreshold)
        return
    } // [1]
    quorum = keeper.GetQuorum(ctx) // [2]
    threshold = sdk.MustNewDecFromStr(params.Threshold)
    return
}
```

If a proposal has constitutional amendments or law proposals, these are prioritized, ignoring the quorum/threshold for lower-priority proposals.

Impact

Since the quorum value is now computed dynamically for each proposal kind, it is possible for constitutional amendments and law proposals to have lower-required quorums than regular proposals. If this is the case, proposals could be passed without meeting the minimum participation requirement. This lets proposals, even major ones like constitutional amendments, pass without enough voter participation. It breaks core governance guarantees and opens the door for low-effort attacks or manipulation.

Recommendations

Revert to the previous implementation, returning the highest-required quorum/threshold for any message in a proposal.

Remediation

This issue has been acknowledged by AtomOne, and a fix was implemented in PR #161 ↗.

3.2. Zero `TargetBlockUtilization` disables fee market

Target	x/feemarket/types/state.go		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Low	**Impact**	Low

Description

The `UpdateBaseGasPrice` computes the following:

```
baseGasPrice = someValue / (MaxGas \* TargetBlockUtilization)
```

Currently, at the time of writing, `ValidateBasic` allows `TargetBlockUtilization = 0`. A governance-param change to zero triggers a divide-by-zero panic every block.

The panic is caught by

```
defer func() {
    if rec := recover(); rec != nil {
        logger.Error("Panic recovered in state.UpdateBaseGasPrice", "err", rec)
        s.BaseGasPrice = params.MinBaseGasPrice
        gasPrice = s.BaseGasPrice
    }
}()
```

so the chain stays alive, but `BaseGasPrice` is reset to the floor each block, and an error is logged every block.

Impact

This has the following impacts.

- The fee-market mechanism is effectively disabled; the gas price sticks at `MinBaseGasPrice`.
- Continuous error logs pollute node output and can degrade performance over time.
- A griefing vector is created: one malicious param vote breaks dynamic pricing until another vote passes.

Recommendations

We recommend tightening param validation — enforce `0 < TargetBlockUtilization ≤ 1` and `BaseFeeChangeDenom > 0`.

Also, add unit tests covering edge values (0, 1, negative, >1).

Remediation

This issue has been acknowledged by AtomOne, and a fix was implemented in PR #166↗.

3.3. The `min_quorum > max_quorum` flips interpolation

Target	x/gov/keeper/participation_ema.go		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The quorum is defined as:

```
quorum = min_quorum + (max_quorum - min_quorum) * participationEMA
```

However, validation only enforces each value is in `[0, 1]`. If `min_quorum` is set *higher* than `max_quorum`, the `(max - min)` term becomes negative and the curve is inverted. Higher participation lowers quorum and vice versa. Although still bounded in `[min, max]`, this contradicts the intended design and can surprise governance.

Impact

Misconfigured parameters may raise quorum when participation is low and drop it when participation is high, weakening governance safeguards.

Additionally, confusing behavior increases the risk of accidental misgovernance.

Recommendations

Add a guard in the param validation.

```
if minQuorum > maxQuorum {
    return fmt.Errorf("min_quorum must be ≤ max_quorum")
}
```

Remediation

This issue has been acknowledged by AtomOne, and a fix was implemented in PR #163 ↗.

3.4. Silent unpack failure lowers quorum

Target	x/gov/keeper/participation_ema.go		
Category	Business Logic	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

In x/gov/keeper/proposal.go, each proposal message is unpacked with the following:

```
if err := k.cdc.UnpackAny(msg, &sdkMsg); err == nil {
    // process message
}
```

If `UnpackAny` returns an error, the code simply skips the message; no error is surfaced. When *all* messages in a proposal fail to unpack (e.g., due to corruption), `ProposalKinds` stays empty (`kinds == 0`).

Then, `getQuorumAndThreshold()` evaluates

```
if kinds.HasKindConstitutionAmendment() || kinds.HasKindLaw() { /* higher
    quorum */ }
```

Both checks return `false`, so the default (lowest-quorum) path executes. Thus, a corrupted constitution-amendment proposal would be evaluated with the minimal quorum.

Impact

The scenario is unlikely in normal operation because messages are validated during `SubmitProposal`; however, if state corruption or future code changes allow a bad message through, the proposal would receive an unexpectedly low quorum requirement.

Recommendations

Guard against silent skips:

```
if err := k.cdc.UnpackAny(msg, &sdkMsg); err != nil {
    return fmt.Errorf("failed to unpack proposal msg: %w", err)
}
```

Alternatively, track a flag indicating that at least one message unpacked successfully, and abort otherwise.

Remediation

This issue has been acknowledged by AtomOne, and a fix was implemented in PR #167 ↗.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Test suite

The coverage appears comfortably adequate across all three pull requests, and each suite does a credible job of simulating real-world usage.

PR 114 (fee market) ships the most rigorous package. Dozens of new test files test AnteHandler deductions, keeper math, and end-to-end block production.

PR 135 (dynamic quorum) tests marching participation from its minimum to maximum values, confirming that the clamp logic kicks in when turnout collapses and fails on malformed parameters. What it lacks is a property or fuzz layer for the participation exponential-moving-average (EMA) formula, so extremely skewed data could still uncover edge cases.

PR 69 (deposit throttler) delivers a broad keeper suite that hits `GetMinDeposit` branches, ABCI hooks, CLI queries, and migration code. An end-to-end test submits a proposal, walks deposits over several blocks, and confirms the ledger state matches expectations. The numerical backoff function is not fuzzed, which leaves a small window for atypical deposit patterns to slip through.

Overall, functional coverage is good, integration paths are proven, and negative scenarios such as invalid parameters, clamped quorum limits, and mismatched deposit denoms are explicitly asserted.

5. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the modules and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

5.1. Component: Dynamic quorum (x/gov)

Description

This module continuously tunes the minimum voter-participation threshold (quorum) required for proposals to pass, so governance remains possible even when participation fluctuates. It does so by maintaining an exponential moving average (EMA) of participation for general, constitution-amendment, and law proposals. It then computes the next quorum as `quorum = min_quorum + (max_quorum − min_quorum) × participationEMA`.

The result is used during tallying and exposed through gRPC/CLI queries. This is how it works.

1. **State keys.** These include `KeyParticipationEMA`, `KeyConstitutionAmendmentParticipationEMA`, and `KeyLawParticipationEMA`.
2. **Update flow.** After every vote tally, `UpdateParticipationEMA` is called, and it applies the new quorum requirements.
3. **Quorum ranges.** These are stored in `Params.QuorumRange`, `ConstitutionAmendmentQuorumRange`, and `LawQuorumRange` as `{Min, Max}` strings (`sdk.Dec` 0–1).
4. **Genesis/upgrade.** The V3 upgrade handler seeds all EMAs to 12% and copies default quorum ranges into params.

Invariants

- `0 ≤ participationEMA ≤ 1`.
- `0 ≤ minQuorum ≤ maxQuorum ≤ 1`.
- `minQuorum ≤ computedQuorum ≤ maxQuorum` for each bucket.
- For every proposal tally, the matching EMA is updated exactly once.

Test coverage

Covered

- Get/set each EMA key.
- `computeQuorum` correctness across custom (min, max, EMA) triples.

- `UpdateParticipationEMA` path for no-kind, law, constitution, and mixed proposals.
- Genesis/upgrade seeding of params and EMA.
- gRPC query returns expected quorum values.

Not covered

- Negative tests for malformed params (`min > max`, out-of-range values).
- Overflow/precision edge cases on large EMAs.
- Governance param-change integration tests (end-to-end vote → new quorum).

Attack surface

- **Governance param changes.** An attacker can propose params with
  - `min_quorum > max_quorum` → inverted curve.
  - values outside `[0,1]` → runtime panics or impossible quorums.
  - an extremely wide range → quorum swings to 0% or 100%.
- **Direct state manipulation** via migrations/scripts could set EMA outside `[0,1]`; quorum then overflows bounds.

5.2. Component: Fee market (x/feemarket)

Description

The module is forked from https://github.com/skip-mev/feemarket ↗, which implements EIP-1559–like AIMD dynamic gas pricing.

The PR includes some minor changes to the module, including the removal of fee tip and fee escrowing before distribution. This is implemented through a change in the AnteHandler, with the fee now being deducted from the payer account during transaction execution. Previously, the fee was escrowed and only deducted in the PostHandler along with the tip.

5.3. Component: Dynamic deposit (x/gov)

Description

The PR implements changes to the previously added dynamic deposit throttler, now making the deposit increasing time-dependent.

The minimum amount required for proposal submission is fetched through `GetMinInitialDeposit` and the minimum amount for the proposal through `GetMinDeposit`. Both of these helpers return the minimum amount required as stored in the keeper state instead of computing the value as invoked.

The minimum deposit values are updated in the keeper state dynamically through the

`UpdateMinDeposit` / `UpdateMinInitialDeposit` helpers. Both of these helpers implement the same logic for computation of the dynamic deposit amount. The updates happen both in the EndBlocker after dead/complete proposals are removed and when a new proposal is added.

It works as follows:

- The last minimum deposit and deposit time is fetched through `GetLastMinDeposit`.
- If called by the EndBlocker, and the elapsed time is lower than the update period, the minimum deposit is not updated. This would result in no decrease in the minimum deposit if multiple proposals are removed in a single block or within the update period.
- If called by the EndBlocker, and if the number of active proposals still exceed target proposals, the function returns without updating the minimum deposit. Otherwise, the alpha value is computed based on the decrease ratio and distance from the target active proposals to reduce the minimum deposit.
- If called during proposal submission/deposit, and if the number of active proposals exceed target proposals, the alpha value is set to the `IncreaseRatio` from `MinDepositThrottler` params. Otherwise, no updates are made, leading to early return.
- The minimum deposit is increased/decreased by multiplying it with the alpha value computed previously.
- The minimum deposit and update time is updated in the keeper state.

6. Assessment Results

During our assessment on the scoped All in Bits modules, we discovered four findings. No critical issues were found. Three findings were of low impact and the remaining finding was informational in nature.

6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.