Zellic

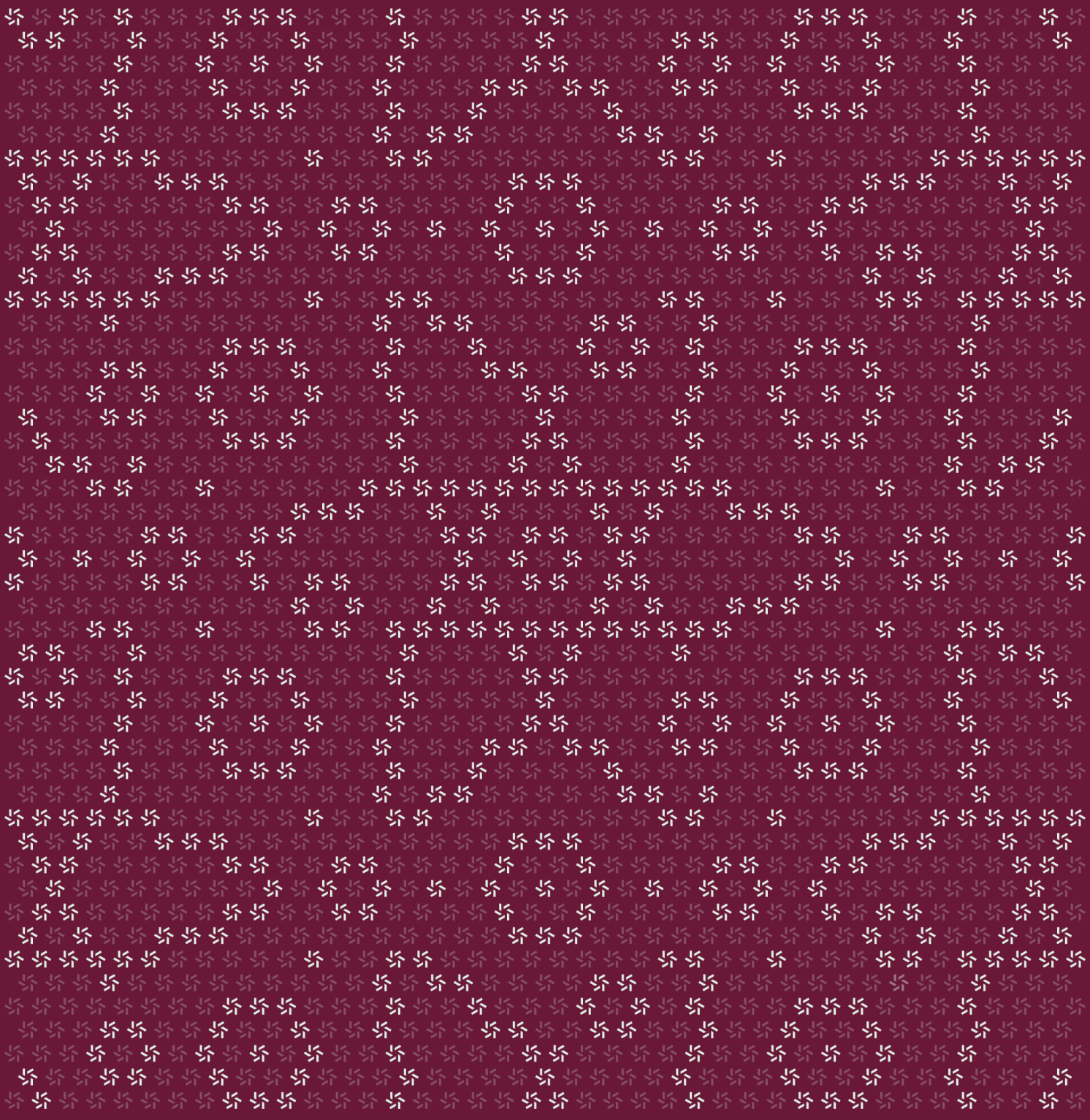
Prepared for
Kristov Atlas
All in Bits

Prepared by
Frank Bachman
Ulrich Myhre
Zellic

March 11, 2025

AtomOne

Smart Contract Security Assessment



Contents

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 All in Bits from February 17th to March 4th, 2025. During this engagement, Zellic reviewed AtomOne'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?
- Is the AtomOne daemon using the SDK securely?
- Does any new functionality introduce additional vulnerabilities (mainly focusing on the photon module and the dynamic 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 AtomOne contracts, there were no security vulnerabilities discovered.

Breakdown of Finding Impacts

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

2. Introduction

2.1. About AtomOne

All in Bits contributed the following description of AtomOne:

> AtomOne is a community-driven fork of the Cosmos Hub, originating from concerns raised around Cosmos Hub Proposal 82 ↗ and Proposal 848 ↗. AtomOne aims to provide a security-conscious, constitutionally governed, streamlined IBC/ICS hub, preserving the foundational ethos of the Cosmos Hub. By prioritizing security, scalability, and decentralization, and operating under a written constitution, AtomOne strives to establish a secure and adaptable platform for interchain security and decentralized governance. It seeks to enhance interoperability, foster inclusivity, and serve as a neutral, community-governed base within the Cosmos ecosystem.

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 contracts.

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.

2.3. Scope

The engagement involved a review of the following targets:

AtomOne Contracts

Type	Go
Platform	Cosmos
Target	Only changes between ca0724f0...49fa9f96
Repository	https://github.com/atomone-hub/atomone ↗
Version	`49fa9f961d4dc8c7046b5e6a1cdad4adceaa5637`
Target	atomone
Repository	https://github.com/atomone-hub/atomone ↗
Version	`266ea299bd307a1f46e630b4bdc944dfecd40656`
Programs	`x/photon/**`
Target	Pull request #69
Repository	https://github.com/atomone-hub/atomone ↗
Version	`f3dce66d9f25eaab0c208eff9ac2a7dc023543b1`
Programs	`x/gov/**`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 2.7 person-weeks. The assessment was conducted by two consultants over the course of 2.4 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

Ulrich Myhre
Engineer
unblvr@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

February 17, 2025	Kick-off call
February 17, 2025	Start of primary review period
March 4, 2025	End of primary review period

3. Threat Model

This provides a description of the high-level components of the system and how they interact, including details like a function's externally controllable inputs and how an attacker could leverage each input to cause harm or which invariants or constraints of the system are critical and must always be upheld.

Not all components in the audit scope may have been modeled. The absence of a component in this section does not necessarily suggest that it is safe.

3.1. Photon

The photon module is responsible for managing the PHOTON token in the AtomOne protocol. The PHOTON token is intended to serve as the sole fee token for AtomOne. The module allows minting of PHOTON tokens by burning ATONEs. The supply for PHOTON is capped at 1B tokens.

Module state

The state maintained by the module is as follows:

- `MintDisabled` — This is a boolean value, which, if enabled, disables minting of PHOTON.
- `TxFeeExceptions` — This is a list of message-type URLs that are exempt from using PHOTON as the fee token.

AnteHandler

The photon module implements an AnteDecorator to enforce transaction fees only being paid in PHOTON. However, it might exempt certain messages from using PHOTON as the fee token. The keeper holds a reference to the `TxFeeExceptions` parameter, which is a list of message-type URLs. The `allowsAnyTxFee` helper function is used with this parameter to check if the transaction qualifies for the exception. In this case, it allows payment of the fee through a different denomination.

Moreover, the checks are skipped if no fees are attached to the transaction.

`MsgMintPhoton`

This message enables users to mint PHOTON tokens by burning a specified amount of bond-denomination coins (ATONE), using a dynamic conversion rate.

Parameters

- `ToAddress` — Recipient address for the minted PHOTONs.
- `Amount` — Amount of bond-denomination coins (ATONE) to burn.

Control flow

1. Ensure minting is enabled.
2. Confirm `Amount` is in the bond denomination.
3. Calculate the conversion rate based on remaining mintable PHOTONs and bond-denom supply.
4. Burn the bond-denomination coins from the sender.
5. Mint the calculated PHOTONs to the recipient.
6. Emit an event with transaction details.

`MsgUpdateParams`

This message allows an authorized entity to update photon module parameters, such as enabling/disabling minting.

Parameters

- `Authority` — Address authorized to make updates.
- `Params` — The value for `MintDisabled`.

Control flow

1. Verify the `Authority` matches the stored authority.
2. Apply the new parameters.

3.2. ADR 01 — Late quorum extension

Late quorum extension is a new functionality that automatically extends the voting period of active governance proposals. It only triggers if quorum is achieved close to the end of the voting period, and ensures that voters get enough time to both discuss the proposal and mitigating any manipulation attempts. In practice, the extension changes the `x/gov` module and adds multiple parameters to tweak it.

Quorum-check mechanism

Changes

- It adds a quorum-check iteration in the `EndBlocker` to process proposals due for quorum evaluation.
- Proposals are removed from the quorum-check queue after evaluation. If quorum is met after a time-out, the voting period may extend; if not met, the proposal is requeued for further checks.

Control flow

- It uses `IterateQuorumCheckQueue` to process proposals based on the block time.
- If quorum is met after the first check, the voting period extends by `MaxVotingPeriodExtension`, updating the active proposal queue.
- If quorum is not met and checks remain, the next check time is calculated using a period derived from `VotingEndTime - QuorumTimeoutTime` divided by `QuorumCheckCount`.

Constitution management

Changes

- It introduces `GetConstitution`, `SetConstitution`, and `ApplyConstitutionAmendment` methods to manage the constitution.

Control flow

- It stores the constitution in the keeper under `KeyConstitution`.
- Amendments are applied as unified diffs using `ApplyUnifiedDiff`, returning an error if the patch fails or if the amendment is empty.

Deposit validation

Changes (Keeper)

- `AddDeposit` was updated to validate deposits against `MinDepositRatio` and `MinDepositAmount`.
- It adds initial deposit validation in `validateInitialDeposit` to check coin validity.

Control flow

- Deposits must meet `MinDepositAmount * MinDepositRatio` for at least one denomination (skipped if ratio is zero).
- It returns an error with a list of required minimums if the threshold is not met.
- It ensures deposits are valid and positive via `IsValid` and `IsAllPositive` checks.

Changes (Message Server)

- It adds `validateDeposit` for `MsgDeposit` to enforce coin validity.

Control flow

- It rejects invalid or negative amounts.

Proposal and voting changes

File: keeper/proposal.go

Changes

- It adds proposals to the quorum-check queue during `ActivateVotingPeriod` if `QuorumCheckCount` is greater than zero.
- It updates `DeleteProposal` to remove entries from the quorum-check queue.

Control flow

- It inserts into the quorum-check queue with a `QuorumCheckQueueEntry` at `VotingStartTime + QuorumTimeout`.
- It iterates from `VotingStartTime` to find and remove quorum-check entries during deletion.

File: keeper/tally.go

Changes

- It removes the veto option from tallying (now requiring a two-thirds majority for quorum).
- It adds `HasReachedQuorum` to check quorum without full tallying.
- It introduces `getQuorumAndThreshold` to adjust quorum and threshold based on proposal type.

Control flow

- `Tally` calculates voting power from delegations only (no validator vote inheritance).
- It fails if the total bonded tokens are zero or quorum is not met (burning deposits if `BurnVoteQuorum` is true).
- It passes if yes votes exceed the threshold among nonabstaining voters.
- `HasReachedQuorum` skips validator prechecks and uses the total voting power against bonded tokens.
- `getQuorumAndThreshold` increases quorum/threshold for `MsgProposeConstitutionAmendment` or `MsgProposeLaw` if their specific parameters are higher.

File: keeper/keeper.go

Changes

- It adds `InsertQuorumCheckQueue`, `RemoveFromQuorumCheckQueue`, and `IterateQuorumCheckQueue` for queue management.

Control flow

- It uses `QuorumCheckQueueKey` for storage and retrieval, with iteration via `QuorumCheckQueueIterator`.

Message server changes

File: keeper/msg_server.go

Changes

- It adds `ProposeLaw` and `ProposeConstitutionAmendment` message handlers.

Control flow

- `ProposeLaw` requires authority but does nothing currently.
- `ProposeConstitutionAmendment` applies the amendment via `ApplyConstitutionAmendment` and updates the constitution, rejecting empty amendments (requires authority).

Genesis and migration

Changes

- It adds constitution to genesis state and initializes quorum-check queues for voting proposals.
- It includes constitution in `ExportGenesis`.

Control flow

- It sets constitution via `SetConstitution` during `InitGenesis`.
- For proposals in `StatusVotingPeriod`, it adds to quorum-check queue if `QuorumCheckCount` is greater than zero and quorum is not met post–time-out (setting `QuorumChecksDone = 1` for potential extension).
- It exports constitution alongside proposals, votes, and parameters.

3.3. Governance proposal deposit auto-throttler

The proposal auto-throttler is an addition to the default governance functionality that dynamically adjusts the `MinDeposit` value for proposals, and with that ensures that the number of active proposals is near a specified target *N*. When this target is exceeded, `MinDeposit` increases exponentially but will change over time or when a proposal is activated or deactivated.

The goal for the feature is to reduce governance spam and voter fatigue by letting stakers pay more attention to a few proposals at any time. The spam itself can be filtered on the front-end level as well, by calling `GetProposalsFiltered()` with a set of parameters to filter by. However, many active proposals are not strictly prohibited, just prohibitively costly. If an important proposal needs to cut through, the deposit can be crowdsourced as well.

Control flow

The minimum deposit required for a proposal (D_{t+1}) at time t can by fetched by `GetMinDeposit()` at any time. It will calculate the dynamic deposit based on

- the current number of active proposals, n_t
- the previous deposit value, D_t
- the time delta since the previous update

- some constants, the most important ones being the target active proposal count N and the minimum deposit D_{min}.

ADR 003 ↗ covers the rationale and recommended values for the constants, and the calculation is

$$D_{t+1} = \max(D_{\text{min}}, D_t \times (1 + sign(n_t - N) \times \alpha \times \sqrt[k]{|n_t - N|}))$$

Minimum deposit is not updated every single tick but can be lazily calculated from the time delta between the current time and the previous update.

Whenever a proposal is created with `SubmitProposal()`, `IncrementInactiveProposalsNumber()` is called to keep track of the number of inactive proposals. When a proposal either goes active or fails and gets deleted, the corresponding `DecrementInactiveProposalsNumber()` is called for bookkeeping. Whenever a proposal is removed, the new minimum deposit is recalculated and could save an otherwise dead proposal at the time of removal if its current deposit meets the new minimum. That ends up calling `ActivateVotingPeriod()`, which has the similar functions `IncrementActiveProposalsNumber()` and `DecrementActiveProposalsNumber()` for bookkeeping. These two functions both make sure to call `SetLastMinDeposit(minDeposit, time)` to reset the time of the last change, otherwise the lazy evaluation would not work properly. The only other place to set the time and deposit is through `UpdateParams` locked behind governance.

Test coverage

- grpc_query.go
  - ☐ Positive test coverage for `MinDeposit` and `MinInitialDeposit`.
- min_deposit.go
  - ☑ Positive test coverage for incrementing, decrementing, setting, and getting the active proposal number.
  - ☐ Negative test coverage for decrementing below 0 or setting to an invalid number.
  - ☑ Positive test coverage for `GetMinDeposit` when above, at, and under the target.
  - ☑ Positive test coverage for `SetLastMinDeposit`.
  - ☐ Positive test coverage for `GetLastMinDeposit`.
- min_initial_deposit.go
  - ☑ Positive test coverage for incrementing, decrementing, setting, and getting the inactive proposal number.
  - ☐ Negative test coverage for decrementing below 0 or setting to an invalid number.
  - ☑ Positive test coverage for `GetMinInitialDeposit` when above, at, and under the target.
  - ☑ Positive test coverage for `SetLastMinInitialDeposit`.
  - ☐ Positive test coverage for `GetLastMinInitialDeposit`.
- types/v1/genesis.go

- ☑ Positive test coverage for throttler parameters.
- ☑ Negative test coverage for throttler parameters.

4. Assessment Results

At the time of our assessment, the reviewed code was partially deployed.

During our assessment on the scoped AtomOne contracts, there were no security vulnerabilities discovered.

4.1. 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.