Security Audit

Manifest (POA)

December 2024 info@hashlock.com.au Hashlock Pty Ltd

Table of Contents

CAUTION

THIS DOCUMENT IS A SECURITY AUDIT REPORT AND MAY CONTAIN CONFIDENTIAL INFORMATION. THIS INCLUDES IDENTIFIED VULNERABILITIES AND MALICIOUS CODE WHICH COULD BE USED TO COMPROMISE THE PROJECT. THIS DOCUMENT SHOULD ONLY BE FOR INTERNAL USE UNTIL ISSUES ARE RESOLVED. ONCE VULNERABILITIES ARE REMEDIATED, THIS REPORT CAN BE MADE PUBLIC. THE CONTENT OF THIS REPORT IS OWNED BY HASHLOCK PTY LTD FOR USE OF THE CLIENT.

Executive Summary

The Lifted Initiative team partnered with Hashlock to conduct a security audit of their POA module. Hashlock manually and proactively reviewed the code in order to ensure the project's team and community that the deployed contracts are secure.

Project Context

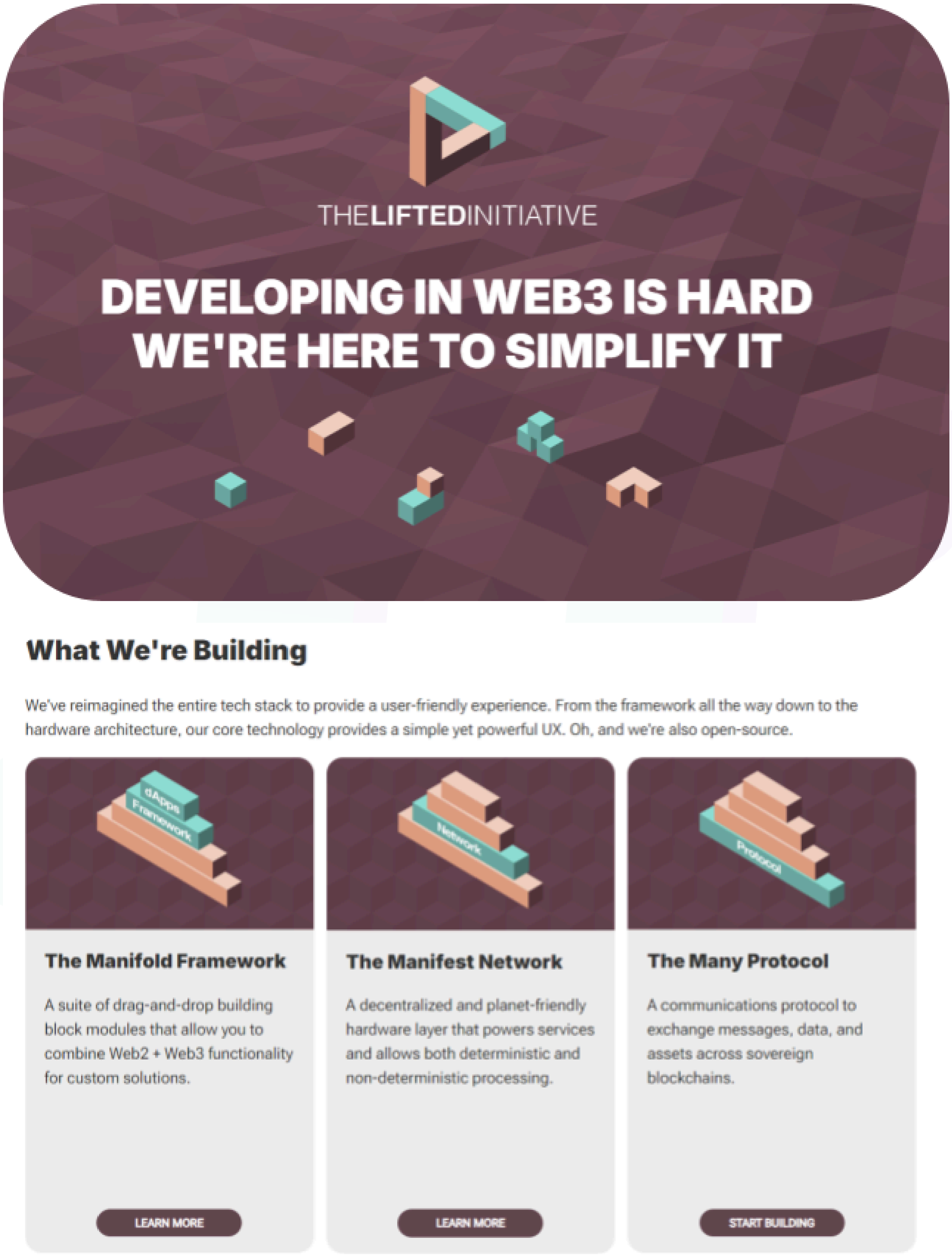
The Lifted Initiative is revolutionizing the Web3 landscape by developing a comprehensive framework that seamlessly integrates Web2 and Web3 functionalities, empowering users to create bespoke layer 1 networks without requiring prior blockchain expertise. By contributing to the Many Protocol, Lifted ensures interoperability across various modules and networks, fostering a cohesive decentralized ecosystem. Their commitment to accessibility and sustainability is evident through initiatives like the Liftoff Program, which supports Web3 builders in refining their products and strategies.
The PoA module is responsible for handling admin actions, such as adding and removing other administrators, setting the staking parameters of the chain, controlling voting power, and allowing/blocking validators.

Project Name: Manifest POA
Compiler Version: go1.22.5 darwin/arm64
Website: https://liftedinit.org
Logo:

Visualised Context:

Project Name	Manifest
Launch Date	TBA
Compiler Version	go1.22.5 darwin/arm64
Language	Go
Network	Cosmos
Token Ticker	N/A

Project Visuals:



DEVELOPING IN WEB3 IS HARD
WE'RE HERE TO SIMPLIFY IT

What We're Building

We've reimagined the entire tech stack to provide a user-friendly experience. From the framework all the way down to the hardware architecture, our core technology provides a simple yet powerful UX. Oh, and we're also open-source.

The Manifold Framework

A suite of drag-and-drop building block modules that allow you to combine Web2 + Web3 functionality for custom solutions.

The Manifest Network

A decentralized and planet-friendly hardware layer that powers services and allows both deterministic and non-deterministic processing.

The Many Protocol

A communications protocol to exchange messages, data, and assets across sovereign blockchains.

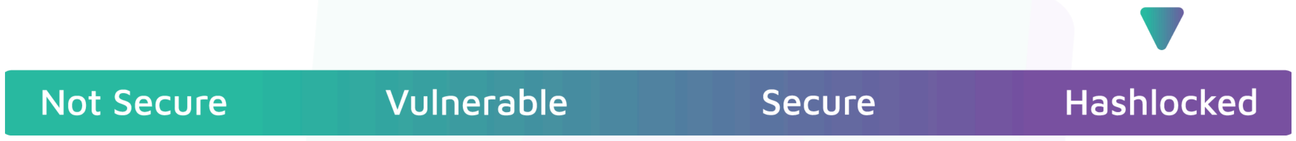
Audit scope

We at Hashlock audited the Go code within the POA module, the scope of work included a comprehensive review of the smart contracts listed below. We tested the smart contracts to check for their security and efficiency. These tests were undertaken primarily through manual line-by-line analysis and were supported by software-assisted testing.

Description	**POA module**
Platform	**Cosmos / Go**
Audit Date	**November/December, 2024**
Repository URL	https://github.com/strangelove-ventures/poa
GitHub Commit hash	6922e22a617a892501b9150662c597ab06bfb805

Security Rating

After Hashlock's Audit, we found the smart contracts to be **"Hashlocked"**. The contracts all follow simple logic, with correct and detailed ordering. We initially identified some significant vulnerabilities that have since been addressed.



The 'Hashlocked' rating is reserved for projects that ensure ongoing security via bug bounty programs or on chain monitoring technology.

All issues uncovered during automated and manual analysis were meticulously reviewed and applicable vulnerabilities are presented in the Audit Findings section. The general security overview is presented in the Standardised Checks section and the project's contract functionality is presented in the Intended Smart Contract Functions section.

All vulnerabilities initially identified have now been resolved and acknowledged.

Hashlock found:

1 High severity vulnerabilities

1 Medium severity vulnerabilities

4 Low severity vulnerabilities

1 Gas Optimisations

Caution: *Hashlock's audits do not guarantee a project's success or ethics, and are not liable or responsible for security. Always conduct independent research about any project before interacting.*

Intended Smart Contract Functions

Claimed Behaviour	Actual Behaviour
POA - Allows users to: - Create validator - Allows the POA admin to: - Set a validator's power - Remove bonded validators - Remove pending validators - Update staking params	**Module achieves this functionality.**

Code Quality

This audit scope involves the smart contracts of the POA module, as outlined in the Audit Scope section. All contracts, libraries, and interfaces mostly follow standard best practices and to help avoid unnecessary complexity that increases the likelihood of exploitation, however, some refactoring was required.

The code is very well commented on and closely follows best practice nat-spec styling. All comments are correctly aligned with code functionality.

Audit Resources

We were given the POA module smart contract code in the form of Github access.

As mentioned above, code parts are well commented. The logic is straightforward, and therefore it is easy to quickly comprehend the programming flow as well as the complex code logic. The comments are helpful in providing an understanding of the protocol's overall architecture.

Dependencies

As per our observation, the libraries used in this smart contracts infrastructure are based on well-known industry standard open source projects.

Apart from libraries, its functions are used in external smart contract calls.

Severity Definitions

Significance	Description
High	High-severity vulnerabilities can result in loss of funds, asset loss, access denial, and other critical issues that will result in the direct loss of funds and control by the owners and community.
Medium	Medium-level difficulties should be solved before deployment, but won't result in loss of funds.
Low	Low-level vulnerabilities are areas that lack best practices that may cause small complications in the future.
Gas	Gas Optimisations, issues, and inefficiencies
QA	Quality Assurance (QA) findings are purely informational and don't impact functionality. These notes help clients improve the clarity, maintainability, or overall structure of the code, ensuring a cleaner and more efficient project. They should be addressed for optimization but are not critical to the system's performance or security.

Audit Findings

High

[H-01] keeper/msg_server.go#CreateValidator - Pending validators can be spammed to cause a denial of service

Description

The `CreateValidator` message allows users to create validators as pending statuses, which will be either accepted or rejected by the POA admin. However, this approach introduces a potential denial of service issue due to unbounded iterations.

For example, an attacker can repeatedly call the `CreateValidator` to spam the `PendingValidators` state. Once the state entries grow, iterating it would fail due to gas limits. Since the `RemovePendingValidator` and `GetPendingValidator` functions iterate through the `PendingValidators` state entries, the `MsgRemovePending` and `MsgSetPower` messages will fail because they rely on these functions internally.

Impact

The attacker can spam pending validators. POA admin will not be able to call the `MsgRemovePending` and `MsgSetPower` messages, causing DoS.

Recommendation

Consider modifying the `PendingValidators` state design to use a mapping that records the validator address as the key and the struct itself as the value. This would remove the need to iterate all the pending validators, effectively decreasing the complexity from `O(n)` to `O(1)`.

This approach would also fix the issue where the same validator address can be added repeatedly to the `PendingValidators` state, as the map key will overwrite it

Note

The Lifted Initiative team stated that, rather than transitioning to a map, they implemented an iteration with an if-check to avoid a module migration at this time. This migration will instead be considered for a future upgrade (e.g., SDK v52), and they plan to create an issue to apply the recommended patch at that point.

They further explained that, as a Proof-of-Authority (PoA) network, token distribution is managed differently, including fees, which serves to mitigate the scope of potential attacks targeting PendingValidator transactions.

Building upon M-01, the team noted that if an out-of-gas situation were to occur, it could halt the entire chain due to the StakingModule iterating through all validators first. While the O(n) complexity is deemed acceptable within the context of PoS and the limited PoA scope, they acknowledged that there is room for improvement.

Status

Resolved

Medium

[M-01] keeper/msg_server.go#RemoveValidator - Potential out-of-gas error when removing validator

Description

When removing a validator, all the validators from the `x/staking` module are iterated to ensure the validator to remove exists and is in bonded status. However, an out-of-gas error might occur when iterating all the validators because the complexity is `O(n)`, causing the transaction to fail.

Impact

POA admin will not be able to remove validators.

Recommendation

Consider using the `GetValidator` function from the `x/staking` module instead. This approach modifies the complexity to `O(1)`, thus preventing a potential out-of-gas error.

Status

Resolved

Low

[L-01] keeper/msg_server.go#UpdateStakingParams - Staking parameters are not validated

Description

When updating the staking parameters, `Params.Validate()` is not called to ensure the updated parameters are valid.

Impact

Potential unintended bugs due to misconfiguration by the POA admin.

Recommendation

Consider calling `Params.Validate()` before setting the `x/staking` module's parameters.

Status

Resolved

[L-02] ante/commission_limit.go#NewCommissionLimitDecorator - RateFloor is not validated to be less than RateCeil

Description

When constructing the `CommissionLimitDecorator`, the `RateFloor` is not validated to be lesser than `RateCeil`.

Impact

If misconfigured, commission decorators will not work correctly because the `rateCheck` function validation cannot be satisfied.

Recommendation

Consider adding validation to ensure `RateFloor` is less than `RateCeil`.

Status

Resolved

[L-03] keeper/msg_server.go#UpdateStakingParams - Validators commission rate may be lesser than minimum commission rate

Description

When updating the staking parameters, the function does not handle a case where the new `MinCommissionRate` is less than the old `MinCommissionRate` and updates all the validator's `CommissionRates` accordingly.

Impact

The invariant where the validator's commission rate must be equal or larger than the `params.MinCommissionRate` will not be enforced.

Recommendation

Consider implementing the logic according to the `x/staking` module, which can be found here.

Status

Resolved

[L-04] keeper/genesis.go#InitGenesis - Missing duplicate validators check during genesis initialization

Description

When executing the `InitGenesis` functions, parameters such as `data.Vals` are utilized, which contain validators that will be initially assigned by the caller. However, there is no validation if that vector does not contain duplicated entries, like the same validators assigned multiple times.

Impact

The issue could lead to problematic situations while working on the validator objects. Potentially, it could lead to inefficient validator removal or power miscalculations.

Recommendation

Consider implementing a validation to ensure only unique validators are passed to `data.Vals`.

Status

Resolved

Gas

[G-01] keeper/* - Duplicate function calls

Description

In the following instances, there are duplicate calls occurring:

- `ms.k.UpdateBondedPoolPower` is called in `keeper/msg_server.go#L83`, but it was previously called in `keeper/poa.go#L233`
- `ms.k.UpdateBondedPoolPower` is called in `keeper/msg_server.go#L137`, but it was previously called in `keeper/poa.go#L233`
- `newVal.ConsensusPubkey` is called in `keeper/pending.go#L27`, but it was already set in `keeper/msg_server.go#L210`
- `k.slashKeeper.DeleteMissedBlockBitmap` is called in `keeper/slashing.go#L35`, but it was previously called in `keeper/poa.go#L107`

These duplicate calls incur unnecessary gas consumption.

Recommendation

Consider removing the duplicate calls.

Note

The Lifted Initiative team stated that the SDK exhibits some unusual behavior, and the duplicates are intentional due to interactions with the staking module. They will create an issue to revisit the matter at a later time.

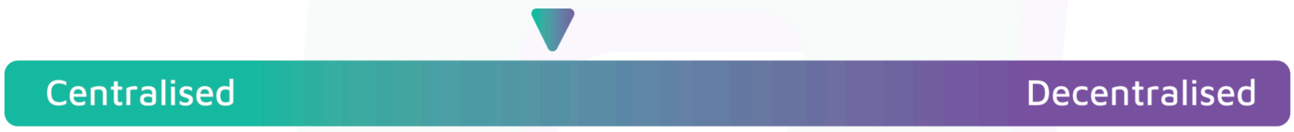
Status

Acknowledged

Centralisation

The POA module values security and utility over decentralisation.

The owner executable functions within the protocol increase security and functionality but depend highly on internal team responsibility.



Conclusion

After Hashlocks analysis, the POA module seems to have a sound and well-tested code base, now that our vulnerability findings have been resolved and acknowledged. Overall, most of the code is correctly ordered and follows industry best practices. The code is well commented on as well. To the best of our ability, Hashlock is not able to identify any further vulnerabilities.

Our Methodology

Hashlock strives to maintain a transparent working process and to make our audits a collaborative effort. The objective of our security audits is to improve the quality of systems and upcoming projects we review and to aim for sufficient remediation to help protect users and project leaders. Below is the methodology we use in our security audit process.

Manual Code Review:
In manually analysing all of the code, we seek to find any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behaviour when it is relevant to a particular line of investigation.

Vulnerability Analysis:
Our methodologies include manual code analysis, user interface interaction, and white box penetration testing. We consider the project's website, specifications, and whitepaper (if available) to attain a high-level understanding of what functionality the smart contract under review contains. We then communicate with the developers and founders to gain insight into their vision for the project. We install and deploy the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:
We undergo a robust, transparent process for analysing potential security vulnerabilities and seeing them through to successful remediation. When a potential issue is discovered, we immediately create an issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is vast because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyse the feasibility of an attack in a live system.

Suggested Solutions:
We search for immediate mitigations that live deployments can take and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the contract details are made public.

Disclaimers

Hashlock's Disclaimer

Hashlock's team has analysed these smart contracts in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in the smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Hashlock is not responsible for the safety of any funds and is not in any way liable for the security of the project.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to attacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

About Hashlock

Hashlock is an Australian-based company aiming to help facilitate the successful widespread adoption of distributed ledger technology. Our key services all have a focus on security, as well as projects that focus on streamlined adoption in the business sector.

Hashlock is excited to continue to grow its partnerships with developers and other web3-oriented companies to collaborate on secure innovation, helping businesses and decentralised entities alike.

Website: hashlock.com.au
Contact: info@hashlock.com.au