hashlock.

2024

Security Audit

Manifest (CosmosSDK)

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

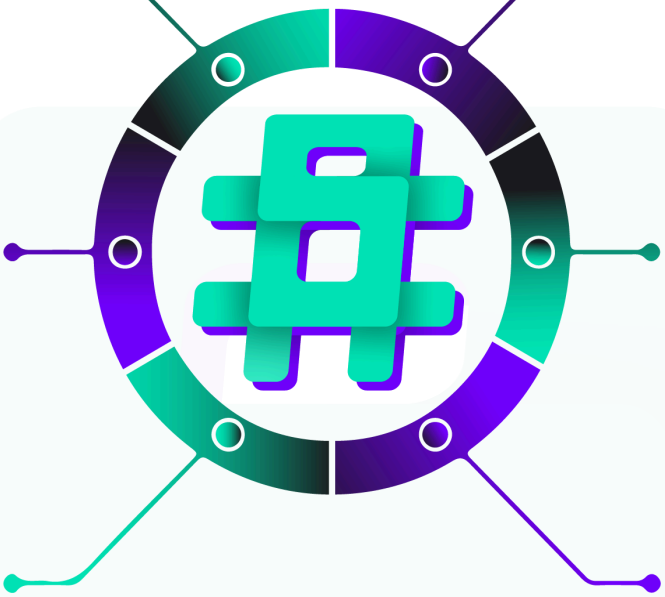
The Lifted Initiative team partnered with Hashlock to conduct a security audit of their Custom Cosmos SDK changes. Hashlock manually and proactively reviewed the code in order to ensure the project's team and community that the deployed contracts are secure.

Project Context

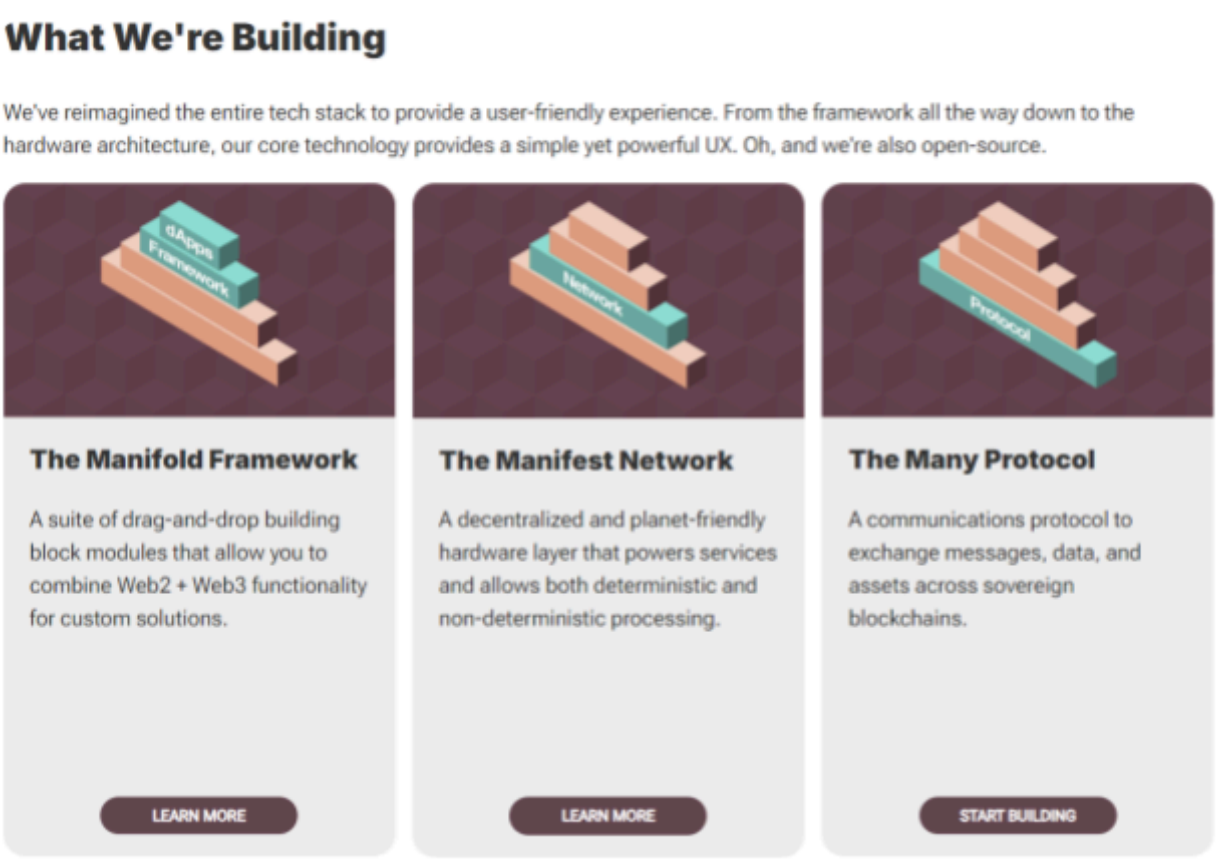
The Lifted Initiative is revolutionizing the Web3 landscape by developing a comprehensive framework that seamlessly integrates Web2 and Web3 functionalities, empowering users to create bespoke layer 1 networks without requiring prior blockchain expertise. By contributing to the Many Protocol, Lifted 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 custom Cosmos SDK changes involves modifications to the core Cosmos SDK framework to support specific features or requirements for LiftedInit's blockchain use cases.

Project Name: Custom Cosmos SDK changes
Compiler Version: go version go1.22.5 darwin/arm64
Website: https://liftedinit.org
Logo:

Visualised Context:



Project Visuals:



Audit scope

We at Hashlock audited the Go code within the Custom Cosmos SDK, the scope of work included a comprehensive review of the blockchain components listed below. We tested the blockchain components 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	Custom Cosmos SDK
Platform	Cosmos / Go
Audit Date	December, 2024
Repository URL	https://github.com/liftedinit/cosmos-sdk/pull/1
Commit reviewed until	c8c96a0017c6f29ac34a8a764eecafafcd6b2830

Security Rating

After Hashlock's Audit, we found the blockchain components to be **"Hashlocked"**. The components all follow advanced and complicated 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 Blockchain Components Functions section.

All vulnerabilities initially identified have now been resolved and acknowledged.

Hashlock found:

2 Low severity vulnerabilities

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 Blockchain Components Functions

Claimed Behaviour	Actual Behaviour
https://github.com/liftedinit/cosmos-sdk/pull/1 - Removes the `mint` module from the `BeginBlocker` to allow override with Manifest chain. - Apply `.gitpatches` on top of Cosmos SDK.	**Code achieves this functionality.**

Code Quality

This audit scope involves the Go code of the Custom Cosmos SDK changes, 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 Custom Cosmos SDK changes code in the form of Github access.

As mentioned above, code parts are well commented. The logic is very complicated and advanced, and therefore it is time consuming 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 blockchain components infrastructure are based on well-known industry standard open source projects.

Severity Definitions

The severity levels assigned to findings represent a comprehensive evaluation of both their potential impact and the likelihood of occurrence within the system. These categorizations are established based on Hashlock's professional standards and expertise, incorporating both industry best practices and our discretion as security auditors. This ensures a tailored assessment that reflects the specific context and risk profile of each finding.

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 informational and don't impact functionality. Supports clients improve the clarity, maintainability, or overall structure of the code.

Status Definitions

Each identified security finding is assigned a status that reflects its current stage of remediation or acknowledgment. The status provides clarity on the handling of the issue and ensures transparency in the auditing process. The statuses are as follows:

Significance	Description
Resolved	The identified vulnerability has been fully mitigated either through the implementation of the recommended solution proposed by Hashlock or through an alternative client-provided solution that demonstrably addresses the issue
Acknowledged	The client has formally recognized the vulnerability but has chosen not to address it due to the high cost or complexity of remediation. This status is acceptable for medium and low-severity findings after internal review and agreement. However, all high-severity findings must be resolved without exception.
Unresolved	The finding remains neither remediated nor formally acknowledged by the client, leaving the vulnerability unaddressed.

Audit Findings

Low

[L-01] simapp/app.go#L450, app_config.go#L117 - Unneeded mint module in BeginBlocker

Description

The `BeginBlockers` ordering configuration includes the `mint` module. This is unneeded because the `mint` module has been removed.

Recommendation

Consider removing the `minttypes.ModuleName` from the `BeginBlockers`.

Note

The Lifted Initiative team stated that while Manifest does not use the mint module, the upstream SDK still uses it. To leave minimal scope modified upstream, this is left as is (better UX and easier git diff vs changing things not needed upstream). They prefer keeping as close to the source as possible.

Status

Acknowledged

[L-02] x/authz/keeper/keeper.go#L52, x/authz/module/module.go#L116 - Dangerous SetBankKeeper implementation

Description

The `authz` module in Cosmos SDK implements a non-standard keeper initialization pattern where the `BankKeeper` dependency is injected through a setter method marked as deprecated with an explicit `"DO NOT USE"` comment in `x/authz/keeper/keeper.go:50`.

This setter is exclusively used during module initialization in `NewAppModule` as a version compatibility mechanism between v0.47 and v0.50. While this pattern doesn't present immediate operational risks due to its confined usage during single-threaded initialization, it introduces technical debt and architectural concerns.

The pattern bypasses the standard dependency injection approach used throughout the SDK, making the initialization flow less transparent. The explicit warning comment contradicting actual usage creates documentation inconsistency and may impact code review quality.

Recommendation

The initialization pattern should be refactored to align with Cosmos SDK's standard keeper initialization practices. The `BankKeeper` dependency should be provided during keeper construction rather than through post-construction mutation. Documentation should be added explaining the temporary nature of this pattern and its planned deprecation timeline. Future breaking changes should prioritize removing this compatibility layer in favor of standardized initialization patterns, reducing potential developer confusion.

Note

The Lifted Initiative team stated that this is an upstream SDK design that was not modified on their end for Manifest. This may be more useful to the binary team as an issue than for them, as they just consume.

Status

Acknowledged

Centralisation

The custom Cosmos SDK changes project 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 Hashlock's analysis, the Custom Cosmos SDK project 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 blockchain components 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 blockchain components in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in the blockchain components 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 these chain components.

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

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

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