Least Authority
PRIVACY MATTERS

Fundraising Module

Security Audit Report

All in Bits

Final Audit Report: 25 July 2022

Table of Contents

Overview

Background

All in Bits has requested that Least Authority perform a security audit of their Cosmos SDK Fundraising Module, which serves as a fundraising feature and provides an opportunity for onboarding new types of projects within the Cosmos ecosystem.

Project Dates

- **May 9 - June 3:** Initial Code Review *(Completed)*
- **June 8:** Delivery of Initial Audit Report *(Completed)*
- **July 21 - 22:** Verification Review *(Completed)*
- **July 25:** Delivery of Final Audit Report *(Completed)*

Review Team

- DK, Security Researcher and Engineer
- Nishit Majithia, Security Researcher and Engineer
- Xenofon Mitakidis, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Fundraising Module followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repository is considered in-scope for the review:

- Fundraising Module:
  https://github.com/tendermint/fundraising/releases/tag/v0.3.0

Specifically, we examined the Git revision for our initial review:

```
7524ab763e18891752ddddf03bdc69d38791a8c4
```

For the verification, we examined the Git revision:

```
f0e0f86d14f491128b1c48e97a4bdd9bdee29cd6
```

For the review, this repository was cloned for use during the audit and for reference in this report:

Fundraising Module:
https://github.com/LeastAuthority/All-in-Bits_Fundraising_Module

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in-scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- README.md:

https://github.com/LeastAuthority/All-in-Bits_Fundraising_Module/tree/main/docs
- README.md: https://github.com/LeastAuthority/All-in-Bits_Fundraising_Module/blob/main/x/fundraising/spec/README.md

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Common and case-specific implementation errors;
- Adversarial actions and other attacks on the Fundraising Module;
- Potential misuse and gaming of the Fundraising Module;
- Attacks that impact funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of Service (DoS) and other security exploits that would impact the Fundraising Module's intended use or disrupt the execution;
- Vulnerabilities in the Fundraising Module's code;
- Vulnerabilities in the Fundraising Module's interactions with other existing Cosmos SDK modules and components;
- Protection against malicious attacks and other ways to exploit the Fundraising Module;
- Performance of the components under load leading to potential security vulnerabilities;
- Exposure of critical information during interactions with external libraries;
- Use and management of dependencies and the process for updating dependent libraries;
- Data privacy, data leaking, and information integrity;
- Inappropriate permissions and excess authority; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

The Fundraising module is a Cosmos SDK module designed to provide fundraising functionality whereby projects can raise funds by auctioning project tokens. There are two types of auctions that an auctioneer can perform, Fixed Price Auction and Batch Auction. The auctioneer adds bidders who are authorized to make bids. If the bidder's price is matched, then the bidder gets the amount of the auctioned coin in the bid. Otherwise, the bid is returned to the bidder. The Fundraising module is dependent on the Starport module, which provides interaction with the underlying blockchain layer.

Our team checked the fixed price and batch auction functionality for susceptibility to obtaining tokens without price matching or by changing the matching price. We looked for opportunities to participate in the auction without authorization and could not find any. We further checked for a circumstance in which an auctioneer or bidder would be unable to return legitimate assets after an auction and did not identify any.

We checked for the possibility of forcefully canceling an ongoing auction or manipulating the auction end time. We also examined bid denials or bid cancellations for specific bidders, the reduction of bids by bidders, or the modification of bids by third parties and could not identify any issues.

We found the design and implementation of the Fundraising module generally adheres to best practices and is well documented. In addition, sufficient test coverage has been implemented.

System Design

Our team found that security has been taken into consideration in the design of the Fundraising module as demonstrated by appropriate data input validation and access controls. Although measures to protect the system from off-chain sniping bots are in place, we suggest that the system be secured against on-chain bots (Suggestion 3).

Code Quality

The Fundraising module is well organized and generally adheres to Cosmos SDK's best practice recommendations. However, our team identified unresolved TODOs in the codebase, which reduced the readability of the code and raised questions about its completeness. We recommend that outstanding TODOs be resolved and removed from the codebase (Suggestion 1). We also found that error handling in the implementation can be improved by limiting and avoiding the use of panics (Suggestion 2).

Tests

A sufficient test suite has been implemented that tests for all the states that the system can transition to. This helps check for edge cases and potential implementation errors.

Documentation

Our team was provided project documentation that sufficiently describes the Fundraising module and interface for onboarding new projects into the ecosystem. One improvement that we recommend implementing is adding links to the modules.

Code Comments

The codebase contains sufficient code comments, which describe the intended functionality of security-critical functions and components.

Scope

We found the scope of the audit for the Fundraising Module to be sufficient, in that it encompassed all security-critical components. However, the Fundraising module is intended to work interdependently with other cosmos ecosystem modules, which were assumed to function as intended for this review.

Specific Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

SUGGESTION	STATUS
Suggestion 1: Resolve TODOs in the Codebase	Resolved
Suggestion 2: Improve Error Handling, Limit and Avoid Using Panics	Unresolved
Suggestion 3: Improve Anti-sniping Measures	Unresolved

Suggestions

Suggestion 1: Resolve TODOs in the Codebase

Location

Examples (non-exhaustive).

`app/state.go`

`simapp/simapp.go`

Synopsis

Our team identified several instances of unresolved TODOs. Unresolved TODOs decrease code readability and may create confusion about the completeness of the codebase and the intended functionality of each of the system components. This can hinder the ability for security researchers to identify implementation errors.

Mitigation

We recommend identifying and resolving all pending TODOs in the codebase.

Status

The All in Bits team resolved the pending TODOs in the codebase.

Verification

Resolved.

Suggestion 2: Improve Error Handling, Limit and Avoid Using Panics

Location

Examples (non-exhaustive).

`app/state.go:43`

`x/fundraising/types/auction.go:76`

`x/fundraising/types/auction.go:133`

`x/fundraising/types/msgs.go:90`

`x/fundraising/types/msgs.go:98`

`x/fundraising/types/keys.go:99`

`x/fundraising/keeper/auction.go:446`

`x/fundraising/module.go:81`

Synopsis

There are multiple instances in the code that can trigger a panic if an error occurs, or where returned errors are not checked. Functions that can cause the code to panic at runtime may lead to denial of service.

Mitigation

We recommend refactoring the code and removing panics where possible. One of the possible improvements is to propagate errors to the caller and handle them on the upper layers. Note that error handling does not exclude using panics. In addition, if a caller can return an error, the callee function may not panic but, instead, propagate an error to the caller.

Status

The All in Bits team have responded that the panics and error handling are intended and satisfy their engineering practices. Therefore, they will not make the recommended changes.

Verification

Unresolved.

Suggestion 3: Improve Anti-sniping Measures

Location

`x/fundraising/types/fundraising.pb.go#L302`

Synopsis

The current measure to prevent sniping in batch auctions is to extend the end time of the auction by a predetermined amount of time. While this mitigates the use of web-based off-chain sniping bots, it is possible for an on-chain bot to see that the auction continues.

Mitigation

One way to prevent sniping is to change the auction model to a sealed bid style. Currently, that requires either the use of zero-knowledge proofs that can be uneconomical or the use of a Trusted Execution Environment, which introduces new attack vectors that should be considered.

An alternative approach to prevent sniping would be to implement a dynamic end time to the auctions. One example is based on tracking the time since the last bid, such that that miner collusion is unprofitable (up to a hard capped limit of bids) and proportional to the number of allowed bidders. Combining this strategy with a minimum bidding increment will force a bot to bid as if the auction was English. It would have to bid every time it is outbidded up to its maximum allowance, giving human participants a chance to compete.

Status

The All in Bits team have responded that the current mitigation measure suffices to achieve their business goals and, therefore, will not make the recommended changes.

Verification

Unresolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, and zero-knowledge protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for 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 behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As 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. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative 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, 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 analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive 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 scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.