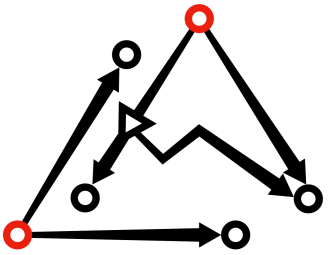

Least Authority

PRIVACY MATTERS

BandChain Cosmos-SDK Oracle Module

Security Audit Report

Band Protocol

Updated Final Report Version: 10 August 2020

Table of Contents

Overview

Background

Band Protocol has requested that Least Authority perform a security audit of the BandChain Cosmos-SDK oracle module in the BandChain chain directory. Bandchain is a Decentralized Data Delivery Network (D3N) built on Cosmos-SDK. BandChain aims to connect public blockchains with off-chain information and aims to achieve the following design goals:

- Speed and Scalability: The system must be able to serve a large quantity of data requests to multiple public blockchains with minimal latency and high throughput. The expected response time must be in the order of seconds.
- Cross-Chain Compatibility: The system must be blockchain-agnostic and able to serve data to most available public blockchains. Verification of data authenticity on the target blockchains must be efficient and trustless.
- Data Flexibility: The system must be generic and able to support different ways to fetch and aggregate data, including both permissionless, publicly available data and information guarded by centralized parties.

Project Dates

- **July 7 - July 21** : Code review completed *(Completed)*
- **July 23**: Delivery of Initial Audit Report *(Completed)*
- **August 3 - 5:** Verification completed *(Completed)*
- **August 6:** Delivery of Final Audit Report *(Completed)*
- **August 10:** Delivery of Updated Final Audit Report *(Completed)*

Review Team

- Nathan Ginnever, Security Researcher and Engineer
- Dylan Lott, Security Researcher and Engineer
- Jan Winkelmann, Cryptography Researcher and Engineer
- Mirco Richter, Cryptography Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the BandChain Cosmos-SDK Oracle Module followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

- Cosmos-SDK Oracle Module Code:
    - https://github.com/bandprotocol/bandchain/tree/master/chain/x
    - https://github.com/bandprotocol/bandchain/tree/master/chain/pkg

Code repositories not listed above and third party vendor code is considered out of scope.

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

`68814332e458767fe09a2856f43df6ce0545713e`

For the verification, we examined the Git revision:

```
4242ca61e68b028ea41946260d124a519bdc2521
```

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

Supporting Documentation

The following documentation is available to the review team:

- Specification: https://github.com/bandprotocol/bandchain/tree/master/spec
- Band Protocol Medium: https://medium.com/bandprotocol
- BandChain Wiki: https://github.com/bandprotocol/bandchain/wiki
- BandChain Developer Guide: https://docs.bandchain.org/

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Vulnerabilities that currently exist in the code;
- Adversarial actions and other attacks on the network;
- Protection against malicious attacks and other methods of exploitation;
- Securely handling large volumes of network traffic;
- Resistance to DDoS and similar attacks;
- Incentive mechanisms and their impact on network interactions;
- Inappropriate permissions and excess authority;and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

The Band Protocol's BandChain Cosmos-SDK oracle module depends on both BandChain code and the Cosmos SDK. We found that it follows the app structure of Cosmos SDK, adhering to Cosmos best practices, as demonstrated in the Cosmos SDK documentation.

Review Scope

The scope of the review was intentionally narrowed to adhere to a shorter audit schedule. Although this presents challenges, this is mitigated with the planned priorities and we have noted areas that we recommend for further review. However, in order to better contextualize the in-scope components, our team lightly reviewed some areas that were outside of the scope of this audit in order to provide meaningful feedback to the Band Protocol team. This encompassed the Deterministic Random Number Generator (DRNG) that is not currently based on a standard construction known to be secure (Suggestion 2), which is used to determine the set of validators responsible for the transaction. We also briefly investigated the Go-Owasm library, which creates an execution environment for the oracle scripts to run when resolving requests. In particular, the Go-Owasm library would benefit from further investigation in order to assess the potential impact and affect on the system at runtime (Suggestion 5).

Code Quality

Overall, we found the packages in-scope to be well organized and structured in a manner that was simple, intuitive, and easy to follow. The code has adequate test coverage and includes the core components of the modules we reviewed. We commend the Band Protocol team for following Cosmos-SDK conventions and approaching the design and development of the project with security in mind. The code is idiomatic and passes linter tests and formatters that our team ran specifically for the Go code base. It is particularly notable that the Band Protocol team was very thorough and carefully considered and properly handled edge cases that could have had implications for security.

Documentation

The Band Protocol team has taken considerable effort to create and maintain comprehensive documentation. The BandChain Wiki provides a fair amount of conceptual information, there are numerous blog posts that cover major design choices, and the developer documentation provides meaningful and relevant use cases. In addition, most of the code was sufficiently commented, with the exception of a number of uncommented exports, including several exported names, such as types, functions, variables, and constants. For example, it is best practice for exported types, functions and values to have a short comment that provides an explanation of what the function is intended to do. We also found several instances throughout the code where function comments only describe which interface it implements and some cases where the type is a struct when it should be an interface type. We recommend further improvement to the code documentation such that it is coherent and helpful, allowing users and implementers to comprehend the code more easily and efficiently (Suggestion 1).

System Design

Our team found the design of the system to be robust and well thought out from a security perspective. Although the Band Protocol team has made calculated trade-offs, it is clear that security and performance has been considered at each step of the design and in each module. In addition, we found evidence of edge case considerations in areas where there could have been incentive misalignments or opportunities for Denial-of-Service (DoS) attacks.

Oracle Data Validation

The current system does not validate or come to consensus on data that oracles return, meaning that the data must be implicitly trusted, but allows any data requestor to flexibly specify how to aggregate raw data points with a WebAssembly script. The Band Protocol team has noted that they are aware of this trade-off and the subsequent consequences of this key design choice. Nonetheless, we recommend that extra care is taken with this approach of not validating the returned data. Due to the limited scope of this audit and the open ended nature of the way the oracles may be used without a specific use case defined, we did not explore how the raw data is used in parts of the system beyond the scope of this audit. As a result, it is unclear to our team what the specific security implications of potentially incorrect data being supplied. This could result in a potential attack vector, however, more information and investigation is required in order to assess to what degree this is a vulnerable area of the system. The Band Protocol team has also noted that it is not a permanent choice and will update the protocol in the future if they find it necessary. Future iterations of the protocol may expand on the base system provided in this audit. This work could include a consensus protocol independent of Tendermint for data validity such as Proof of Work (PoW), Proof of Stake (PoS), a hybrid of both, majority voting, trusted third parties, or weighted voting (Suggestion 3).

Oracle Mechanism

Our team paid particular attention to the oracle module, given that it is the core of the protocol. While blockchain oracles are fairly new, they are a considerably well understood problem. Still, this is an area of

active research that warrants careful study in order to determine how to safely bridge blockchain data with external data, and investigate carefully the mechanisms that accomplish this effort. In the BandChain Protocol, the oracle module splits the task of data retrieval into transactions that maintain deterministic block state transitions and randomness is derived in a decentralized protocol for selecting data validators, providing additional security.

Overall, we found the mechanism to be straight-forward and effective for the trade-offs the Band Protocol team has selected. However, we have made some recommendations for future improvement (Suggestion 2; Suggestion 3), which will increase complexity and will require resources and effort to implement correctly.

Validator Incentives

There are several places where the Band Protocol team accounted for and mitigated potential DoS attacks and incentive misalignment issues due to computational burden on nodes. For example, the execution of oracle scripts is bound in order to consume fewer resources and limit possible abuse. We also observed several other examples of due diligence while investigating for potential attack vectors, including limiting the number of validators that an oracle script may request, the presence of a governance parameter defining the expire block for requests such that all validators have a chance to report without being maliciously marked as inactive, and limiting sample retries required from validators. We believe these mitigation strategies are sufficient to defend against potential DoS attack vectors.

Block Time Considerations

As we approached the transaction layer of the protocol, we were initially concerned that the block time computation is too complex. The Band Protocol team stated that they have run tests against the worst case transaction complexity and have found that the runtimes are within reason for the target they have set of a two to three second block time and reasonable verification costs. We did not identify any module code that could consume an unbounded amount of resources, mitigating against a potential DoS attack on the entire chain.

Specific Issues

We list the issues we found in the code 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.

ISSUE / SUGGESTION	STATUS
Issue A: Module Ordering During Application Instantiation	Resolved
Suggestion 1: Improve Documentation	Resolved
Suggestion 2: Use a Well-Understood DRNG	Resolved
Suggestion 3: Use a Decentralized Consensus Mechanism for Data Validation	Unresolved
Suggestion 4: Increase Test Coverage	Resolved
Suggestion 5: Conduct a Third Party Audit of the Go-Owasm Library	Unresolved

Issue A: Module Ordering During Application Instantiation

Location

https://github.com/LeastAuthority/bandchain/blob/master/chain/app/app.go#L221-L225

Synopsis

The comments in the code base suggest that the oracle module must execute before the staking in order for violators of the oracle protocol to be penalized. The order that appears in the code is such that the staking module runs before the oracle in the `SetOrderEndBlockers` function.

Impact

If the staking module is performed before the oracle, then actions such as penalizing validators stake based on validations done in the oracle module would not be possible.

Mitigation

The Band Protocol team responded to this issue noting that the comment is outdated. The oracle module in the current version will not be providing information to the staking module.

Remediation

Correct the comment before the module registry function to avoid confusion.

Status

The outdated comment has been removed and an additional comment has been added for further clarification of the Cosmos SDK module ordering in `SetOrderBeginBlockers`.

```
      // NOTE: Oracle module must occur before distr as it takes some fee to
distribute to active oracle validators.
```

```
         -        // NOTE: During begin block slashing happens after
            distr.BeginBlocker so that there is nothing left
```

Verification

Resolved.

Suggestions

Suggestion 1: Improve Documentation

Location

BandChain Wiki

https://github.com/LeastAuthority/bandchain/tree/master/chain/x/oracle/client/common/proof

https://github.com/LeastAuthority/bandchain/blob/master/chain/x/oracle/module.go

https://github.com/LeastAuthority/bandchain/blob/master/chain/x/oracle/types/msgs.go

https://github.com/LeastAuthority/bandchain/blob/master/chain/x/oracle/types/params.go

Synopsis

The codebase would benefit from additional comments and improved technical documentation, resulting in more comprehensive project documentation.

We found several uncommented exports including several exported names, such as types, functions, variables, and constants. Every exported name should have a comment describing the purpose of the associated type, function or value. There are several instances of exported functions that only have a comment describing which interface they implement. While this information is helpful, a function comment should first and foremost describe the behavior of the function.

Finally, the current technical documentation lacks comprehensive coverage and provides inadequate explanation of certain technical details around the design and architecture, including a message route for a feature called reporters, a detailed explanation of the state of data validation, and verification and any future plans or thoughts for providing validation.

Mitigation

We recommend that the Band Protocol team update the function comments such that they describe the behavior of the function as well as which interface they are implementing, if any. In addition, we suggest further improvement to the documentation so that coverage is thorough and comprehensive, allowing users and implementers to comprehend the code more easily and efficiently.

Status

The Band Protocol team has created a pull request to improve code comments in the suggested interfaces and created comments to explain the params and module files. A second pull request also adds comments to the `proof.go` file including comments that go so far as to illustrate merkle tree diagrams for various state roots.

Verification

Resolved.

Suggestion 2: Use a Well-Understood DRNG

Location

https://github.com/LeastAuthority/bandchain/blob/master/chain/x/oracle/keeper/owasm.go#L33

Synopsis

The design of the DRNG used for validator selection is ad hoc. While our team did not identify issues in the DRNG implemented by the Band Protocol team, using well-understood and established algorithms that currently exist significantly reduces the risk of making subtle, yet critical and consequential mistakes.

Mitigation

We recommend replacing the DRNG with an established algorithm and, if possible, with an established implementation. We suggest review and consideration of the recommendations on random number generation by NIST as the documentation contains multiple suitable algorithms. We recommend using the HMAC_DRBG construction, as the security has been rigorously analyzed by Hirose. While Least Authority has not audited any of them, there are multiple implementations available in many languages, including one developed in Go by Oasis Labs.

Status

The Band Protocol team reviewed the suggested replacements to the rolling hash seed random number generator that our team reviewed and have decided to use an implementation of HMAC_DRBG, specified

in NIST SP 800-90, developed by Oasis Labs. We have reviewed the new code and tests and have not identified any issue and expect that using this standard will strengthen the random number generation in the Band Protocol.

While reviewing the library developed by Oasis Labs, we identified a minor mistake and submitted a pull request. However, the error only concerns an edge case that is not reached by BandChain's code and therefore does not constitute a new security issue.

Verification

Resolved.

Suggestion 3: Use a Decentralized Consensus Mechanism for Data Validation

Location

https://github.com/LeastAuthority/bandchain/wiki/System-Overview#validators

https://github.com/LeastAuthority/bandchain/blob/chain/x/oracle/keeper/validator_status.go

Synopsis

The BandChain Protocol will record validators that do not report data when requested and reward validators based upon their previous activity. In an instance that a `MissReport` function is called on a validator, they are then marked as inactive and unable to get a portion of the oracle reward. While this provides incentive for data availability, which is stated to be the main goal of this phase of the BandChain oracle, it does not provide any guarantees on data correctness. Considering future implementations that could incrementally expand upon the oracle validator role in the direction of scoring and distributed reputation will help to ensure that the data retrieved by oracles is valid.

Mitigation

Many of the traditional consensus mechanisms such as PoW, PoS, or a hybrid version of the two have been attempted as ways of allowing validators to come to consensus on the correctness of oracle data retrieved. The mechanisms are expensive and would likely not be suitable for the underlying Tendermint consensus. A voting protocol and reputation system baked into the current scoring system could provide incremental incentives to report correct data if a portion of the reward not only relies on being active but also on being correct. This introduces the complexities of decentralized reputation or consensus protocols, which may be out of the scope of the BandChain use case at the time this report was created. Previous oracle systems have chosen to implement one of the suggested consensus mechanisms or have provided trusted validators.

Status

The Band Protocol team has responded that, given the tradeoff of maintaining flexible data sources, it will be difficult to create a consensus procedure to determine data correctness at this time. The requests made are able to create custom rule sets that allow for some control of the data validation by every user or external system making requests. For instance, the Band Protocol team can require all data reports to be within a certain range or filter outliers, thus compromising just a portion of validators will not make oracle data entirely invalid. However, this does not provide a clear protocol for validation and requires each script programmer or use case of the oracle to understand the data correctness guarantees they have programmed.

While there is an anti-sybil measure in place by virtue of requiring a Tendermint stake to become a validator, there are no protocol penalties to discourage bad behavior in regards to data validity once

staked. Further information on the Band Protocol team's protocol choices for not implementing decentralized data validation in this current version can be found in the project documentation.

Verification

Unresolved.

Suggestion 4: Increase Test Coverage

Location

Example: https://github.com/LeastAuthority/bandchain/blob/master/chain/x/oracle/keeper

Synopsis

While our team found there to be considerable test coverage throughout the code, we identified a few small functions that do not have an accompanying test. For Example, In `keeper.go`, the functions `SetDataSourceCount`, `SetOracleSciptCount`, and `GetDataSourceNextID` do not have tests. While these functions are simple increments or set a counter, they could be added to tests for full coverage.

Mitigation

Add more test cases to capture full coverage of the codebase.

Status

The Band Protocol team has added more tests to increase coverage to the keeper module.

Verification

Resolved.

Suggestion 5: Conduct a Third Party Audit of the Go-Owasm Library

Synopsis

While a review of the Go-Owasm library is out of scope for this audit, the library has deep-reaching implications in the protocol, given that it used to create and run the oracle scripts. The execution environment of these scripts was observed to be contained reasonably in the oracle module, however, the wasm execution was not explored in this audit. Thus, there may be recommendations for optimizing instruction costs such that more expensive instructions require more gas consumption, or verification that the wasm is compiled and used correctly. As a result, we recommend a full audit of that library.

Mitigation

Conduct an independent third party audit of the Go-Owasm library in an effort to enhance the overall security of the protocol.

Status

The Band Protocol team has responded that they agree an audit of the Go-Owasm library should be conducted in the future and they are adding new features that should be implemented prior to a follow up review.

Verification

Unresolved.

Recommendations

We recommend that the unresolved *Suggestions* stated above are addressed as soon as possible and followed up with verification by the auditing team.

Given the limited scope of our review, focusing only on the Cosmos-SDK oracle module, we recommend a more comprehensive audit of the BandChain platform repository, particularly the Go-Owasm library (as noted in the unresolved Suggestion 5 above) in addition to other libraries related to oracle script preparation and execution.

Furthermore, we recommend further research into validator consensus models for data sources (as noted in the unresolved Suggestion 3 above). The Band Protocol team has noted that this is an active area of research and while we understand the team's design choices and justifications, we encourage that this be considered.

Finally, we commend the Band Protocol team for clearly considering security in their design choices and implementation.

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 C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. 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. Although we are a small team, we believe that 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 included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site 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. 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. We hypothesize what vulnerabilities may be present, creating Issue entries, and for each we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative, 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 though we have not yet 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 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 report, and before the details are made public.

Responsible Disclosure

Before our report or any details about our findings and suggested solutions are made public, 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 resolution that balances the impact on the users and the needs of your project team. We take this agreed timeline into account before publishing any reports to avoid the necessity for full disclosure.