Mango Network

Audit Report

contact@movebit.xyz

https://twitter.com/movebit_

Fri Apr 19 2024

Mango Network Audit Report

1 Executive Summary

1.1 Project Information

Description	Mango Network is a Layer1 blockchain based on the Dpos consensus mechanism and supported by Move to build a transaction-based omni-chain facility infrastructure network.
Type	L1
Auditors	MoveBit
Timeline	Sun Apr 07 2024 - Fri Apr 19 2024
Languages	Rust, Move
Platform	Mango Network
Methods	Dependency Check, Fuzzing, Static Analysis, Manual Review
Source Code	https://github.com/MangoNet-Labs/mango
Commits	3e0faebf7d375a09d094bb6a21e191ef00512ed0 436e772921ee0a6a36d21d0e95f14a9b9399cc90 50ac27c53ddcc5fd0b2edc73e16e317717e8731d

1.2 Files in Scope

The following are the directories of the original reviewed files.

Directory
https://github.com/MangoNet-Labs/mango/mgo-execution
https://github.com/MangoNet-Labs/mango/consensus
https://github.com/MangoNet-Labs/mango/crates
https://github.com/MangoNet-Labs/mango/external-crates
https://github.com/MangoNet-Labs/mango/narwhal

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	2	2	0
Informational	1	1	0
Minor	0	0	0
Medium	0	0	0
Major	1	1	0
Critical	0	0	0

1.4 MoveBit Audit Breakdown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Integer overflow/underflow
- Infinite Loop
- Infinite Recursion
- Race Condition
- Traditional Web Vulnerabilities
- Memory Exhaustion Attack
- Disk Space Exhaustion Attack
- Side-channel Attack
- Denial of Service
- Replay Attacks
- Double-spending Attack
- Eclipse Attack
- Sybil Attack
- Eavesdropping Attack
- Business Logic Issues
- Contract Virtual Machine Vulnerabilities
- Coding Style Issues

1.5 Methodology

Our security team adopted **"Dependency Check"**, **"Automated Static Code Analysis"**, **"Fuzz Testing"**, and **"Manual Review"** to conduct a comprehensive security test on the code in a manner closest to real attacks. The main entry points and scope of the security testing are specified in the **"Files in Scope"**, which can be expanded beyond the scope according to actual testing needs. The main types of this security audit include:

(1) Dependency Check

A comprehensive check of the software's dependency libraries was conducted to ensure all external libraries and frameworks are up-to-date and free of known security vulnerabilities.

(2) Automated Static Code Analysis

Static code analysis tools were used to find common programming errors, potential security vulnerabilities, and code patterns that do not conform to best practices.

(3) Fuzz Testing

A large amount of randomly generated data was inputted into the software to try and trigger potential errors and exceptional paths.

(4) Manual Review

The scope of the code is explained in section 1.2.

(5) Audit Process

- Clarify the scope, objectives, and key requirements of the audit.
- Collect related materials such as software documentation, architecture diagrams, and lists of dependency libraries to provide background information for the audit.
- Use automated tools to generate a list of the software's dependency libraries and employ professional tools to scan these libraries for security vulnerabilities, identifying outdated or known vulnerable dependencies.
- Select and configure automated static analysis tools suitable for the project, perform automated scans to identify security vulnerabilities, non-standard coding, and potential risk points in the code. Evaluate the scanning results to determine which findings require further manual review.

- Design a series of fuzz testing cases aimed at testing the software's ability to handle exceptional data inputs. Analyze the issues found during the testing to determine the defects that need to be fixed.
- Based on the results of the preliminary automated analysis, develop a detailed code review plan, identifying the focus of the review. Experienced auditors perform line-by-line reviews of key components and sensitive functionalities in the code.
- If any issues arise during the audit process, communicate with the code owner in a timely manner. The code owners should actively cooperate (this may include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- Necessary information during the audit process will be well documented in a timely manner for both the audit team and the code owner.

2 Summary

This report has been commissioned by Mango Network with the objective of identifying any potential issues and vulnerabilities within the source code of the Mango Network repository, as well as in the repository dependencies that are not part of an officially recognized library. In this audit, we have employed the following techniques to identify potential vulnerabilities and security issues:

(1) Dependency Check

A comprehensive analysis of the software's dependency libraries was conducted using the Cargo-Audit tool.

(2) Automated Static Code Analysis

The code quality was examined using a code scanner named rust-clippy.
For Rust language, risk codes detected include:

- cast_slice_different_sizes
- invalid_regex
- not_unsafe_ptr_arg_deref
- panicking_unwrap
- read_line_without_trim
- uninit_assumed_init
- while_immutable_condition
- wrong_transmute
- unused_io_amount
- overly_complex_bool_expr
- other risk codes

(3) Fuzz Testing

Based on the PyJFuzz tool and by writing harnesses, we have performed fuzz testing on the JSON-RPC interfaces.

(4) Manual Code Review

We are focusing on the JSON-RPC and wallet-related code.

During the audit, we identified 2 issues of varying severity, listed below.

ID	Title	Severity	Status
CAR-1	[Dependency: h2] - May Lead To DoS	Informational	Fixed
IAV-1	A "sync-from-async" Issue Can Potentially Lead To Node Crashes	Major	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Mango Network repository :

Validator

Validators run special nodes and have additional tasks and responsibilities beyond those of Full node operators.

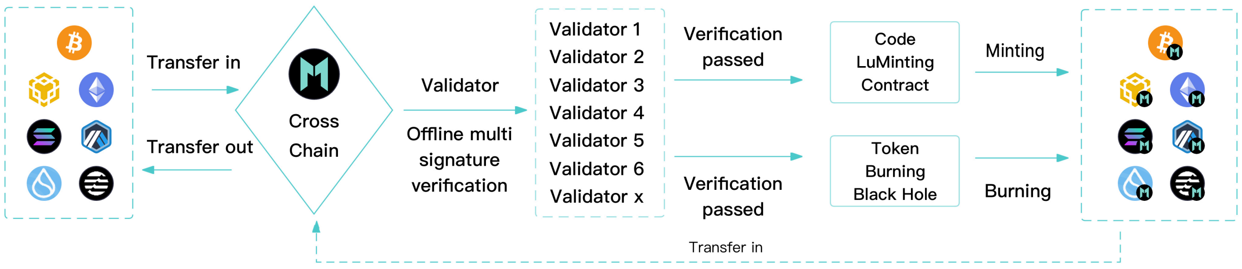
Full Node

Full nodes validate blockchain activities, including transactions, checkpoints, and epoch changes. Each Full node stores and services the queries for the blockchain state and history.

Indexer

Indexer is an off-Full node data service that serves data from the Mango protocol, including both data directly generated from chain and derivative data.

Mango Architecture



4 Findings

CAR-1 [Dependency: h2] - May Lead To DoS

Severity: Informational

Discovery Methods: Dependency Check

Status: Fixed

Code Location:

crates/mgo-swarm-config/Cargo.toml

Descriptions:

An attacker with an HTTP/2 connection to an affected endpoint can send a steady stream of invalid frames to force the generation of reset frames on the victim endpoint. By closing their recv window, the attacker could then force these resets to be queued in an unbounded fashion, resulting in Out Of Memory (OOM) and high CPU usage. RUSTSEC-2024-0003

Suggestion:

Using the command cargo update -p h2 will update the `h2` dependency to the latest version, which can help avoid potential denial-of-service (DoS) issues. Alternatively, you can also avoid the problem by disabling HTTP/2 support.

IAV-1 A "sync-from-async" Issue Can Potentially Lead To Node Crashes

Severity: Major

Discovery Methods: Manual Review

Status: Fixed

Code Location:

crates/mgo-indexer/src/apis/indexer_api_v2.rs#231

Descriptions:

In the "Indexer Apis", the asynchronous function "get_dynamic_field_object" calls "bcs_name_from_dynamic_field_name", which is a time-consuming synchronous operation. We know that in tokio asynchronous tasks, time-consuming synchronous operations are not permitted, as they may lead to a "sync-from-async" problem, which is very likely to cause the program to crash.

Suggestion:

It is recommended to place the "bcs_name_from_dynamic_field_name" function inside a `tokio::spawn_blocking` task for execution.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information or assets at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information or assets at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer