Taker Node
Audit Report

Tue Dec 31 2024

contact@bitslab.xyz https://twitter.com/scalebit_

ScaleBit

Taker Node Audit Report

1 Executive Summary

1.1 Project Information

Description	Taker node is built on the Substrate framework, utilizing the Nominated Proof-of-Liquidity mechanism to combine the nomination and voting system of NPOS with the liquidity incentives of POL to achieve a robust consensus mechanism
Type	L1
Auditors	ScaleBit
Timeline	Fri Dec 20 2024 - Tue Dec 31 2024
Languages	Rust
Platform	Taker
Methods	Dependency Check, Fuzzing, Static Analysis, Manual Review
Source Code	https://github.com/takerprotocol/taker-node
Commits	99ee9c44f3d52d06633e85f79c35badc9bcab7d3 8cb0b745d7b2e09ec34f3c5b854818dc64688df7

1.2 Files in Scope

The following are the directories of the original reviewed files.

Directory
https://github.com/takerprotocol/taker-node/pallets/precompiles
https://github.com/takerprotocol/taker-node/pallets/asset-currency

1.3 Issue Statistic

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

1.4 ScaleBit Audit Breakdown

ScaleBit 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 Taker Protocol with the objective of identifying any potential issues and vulnerabilities within the source code of the Taker Node 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 dependency check tool.

(2) Automated Static Code Analysis

The code quality was examined using a code scanner.

(3) Manual Code Review

The primary focus of the manual code review was:

- takerprotocol/taker-node

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

ID	Title	Severity	Status
LIB-1	Potential Imbalance Due to Saturating Subtraction in `taker_mint_to` Function	Medium	Fixed
LIB-2	Missing Validation for Zero Amount in `taker_mint_to` Function	Minor	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Taker Node repository :

Admin

- `set_controller` : Allows the root account to assign a new controller to the list of token controllers.
- `transfer_whitelist_admin` : Transfers the whitelist admin privileges to a new account.
- `update_whitelist` : Adds or removes accounts from the whitelist based on the provided `add` flag.

User

- `taker_mint_to` : Allows a controller to mint specific tokens to a specified account.
- `taker_burn` : Allows a controller to burn tokens from a specified account.
- `native_mint_to` : Allows a controller to mint native currency to a specified account.
- `transfer` : Allows a whitelisted account to transfer tokens to another account.

4 Findings

LIB-1 Potential Imbalance Due to Saturating Subtraction in `taker_mint_to` Function

Severity: Medium

Discovery Methods: Manual Review

Status: Fixed

Code Location:

pallets/asset-currency/src/lib.rs#321

Descriptions:

```
        pub fn taker_mint_to(
            origin: OriginFor<T>,
            amount: T::Balance,
            to_account: T::AccountId,
        ) -> DispatchResultWithPostInfo {
            let sender: <T as Config>::AccountId = ensure_signed(origin)?;
            ensure!(TokenControllers::<T>::get().contains(&sender), Error::<T>::NotController);
            Account::<T>::mutate(&Self::account_id(), |account| account.free =
account.free.saturating_sub(amount));
            Account::<T>::mutate(&to_account, |account| account.free =
account.free.saturating_add(amount));
            Ok(().into())
        }
```

The `taker_mint_to` function uses the saturating_sub method to subtract the amount from the `account.free` balance. If `account.free` is insufficienamount, the subtraction will result in zero without causing an error, but the `to_account.free` balance will still be incremented by the full amount, creating an imbalance in the system's token accounting.

Suggestion:

Ensures the pallet's account has sufficient funds before proceeding with the transfer. This ensures that the operation fails gracefully if the balance is insufficient.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LIB-2 Missing Validation for Zero Amount in `taker_mint_to` Function

Severity: Minor

Discovery Methods: Manual Review

Status: Fixed

Code Location:

pallets/asset-currency/src/lib.rs#316

Descriptions:

```
        #[pallet::weight(0)]
        #[transactional]
        pub fn taker_mint_to(
            origin: OriginFor<T>,
            amount: T::Balance,
            to_account: T::AccountId,
        ) -> DispatchResultWithPostInfo {
            let sender: <T as Config>::AccountId = ensure_signed(origin)?;
            ensure!(TokenControllers::<T>::get().contains(&sender), Error::<T>::NotController);
            Account::<T>::mutate(&Self::account_id(), |account| account.free =
account.free.saturating_sub(amount));
            Account::<T>::mutate(&to_account, |account| account.free =
account.free.saturating_add(amount));
            Ok(().into())
        }
```

The `taker_mint_to` function does not include a validation check to ensure that the `amount` parameter is greater than zero.

Suggestion:

Add a validation check for the amount parameter to ensure it is greater than zero.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

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

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.