Taker Protocol
Audit Report

Mon Jul 07 2025

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

ScaleBit

Taker Protocol Audit Report

1 Executive Summary

1.1 Project Information

Description	Taker Crotroller and Taker Node focuse on asset management and staking.
Type	DeFi
Auditors	ScaleBit
Timeline	Thu Jul 03 2025 - Fri Jul 04 2025
Languages	Solidity, Rust
Platform	Taker
Methods	Dependency Check, Fuzzing, Static Analysis, Manual Review
Source Code	https://github.com/takerprotocol/token-controller https://github.com/takerprotocol/taker-node
Commits	https://github.com/takerprotocol/token-controller: 2b84a27f768601a33e3461537d6b122d2a8950c0 253205ca7483a55c71502ef8fe41b9ff0d6ddfe0 https://github.com/takerprotocol/taker-node: d99d2927ff372941090ca8ec305afd6f47311aaf 74e0b69e6daec9b120618e158a263721132ec221

1.2 Files in Scope

The following are the directories of the original reviewed files.

Directory
https://github.com/takerprotocol/token-controller/contracts
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	7	7	0
Informational	0	0	0
Minor	5	5	0
Medium	2	2	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 Controller with the objective of identifying any potential issues and vulnerabilities within the source code of the Taker Protocol 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) Fuzz Testing

Based on the fuzz tool and by writing harnesses.

(4) Manual Code Review

Manually reading and analyzing code to uncover vulnerabilities and enhance overall quality.

During the audit, we identified 7 issues of varying severity, listed below.

ID	Title	Severity	Status
ACU-1	Inconsistent Type Mapping Between Solidity Interface and Rust Implementation	Minor	Fixed
AMA-1	`Initialize` Could Be Front-Run	Medium	Fixed
AMA-2	Inconsistent Reentrancy Guard Implementation in Token Exchange	Minor	Fixed

	Functions		
AMA-3	Missing Event Emission	Minor	Fixed
IMP-1	`converted_payout` May Exceed Total Issuance Cap	Medium	Fixed
VET-1	setMinter Does Not Update Whitelist When Changing Minter Address	Minor	Fixed
VET-2	Redundant `whenLive` Check in `burn` Function	Minor	Fixed

3 Participant Process

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

Admin

- `Admin` can set `tToken` , `vToken` , `tGas` , `tStaking` , `tGasRatio` , `isMulRatio` .
- `Admin` can set `live` through the `toggleLive` function.
- `Admin` can add or remove `whiteList` .

Minter

- `Minter` can mint or burn `VETaker` .

User

- `User` can use `exchangeVToTToken` , `exchangeTToVToken` , `exchangeTgasFromTToken` , `exchangeTgasFromVToken` functions to exchange tokens.
- `User` can use `claimTToken` to exchange tokens `tToken` to `vToken` .
- `User` can use `stakingWithNominate` to exchange tokens `vToken` to `tToken` and call `bondAndNominate` remotely through `tStaking` .
- `User` can use `stakingExtra` to exchange tokens `vToken` to `tToken` and call `bondExtra` remotely through `tStaking` .
- `User` can use `stakingExtra` to exchange tokens `vToken` to `tToken` and call `bondExtraAndNominate` remotely through `tStaking` .
- `User` can use `stakingExtra` to exchange tokens `vToken` to `tToken` and call `bondAndValidate` remotely through `tStaking` .
- `User` can use `burnT/mintT/mintTGas` functions to call the `burn/mintTo` interface of `ITToken` and `ITgas` .

4 Findings

ACU-1 Inconsistent Type Mapping Between Solidity Interface and Rust Implementation

Severity: Minor

Discovery Methods:

Status: Fixed

Code Location:

pallets/precompiles/src/asset_currency.rs#27-120;

pallets/precompiles/src/native_currency.rs#31-64;

pallets/precompiles/src/staking.rs#47-559

Descriptions:

In the provided Rust code for a precompile function `mint_to`, there is a discrepancy between the types used in the Solidity interface (uint256) and those used in the Rust implementation (u128). This inconsistency can lead to unexpected behavior or errors when interacting with the contract from an EVM-compatible environment, especially if the value exceeds the maximum representable by u128.Except for `mint_to`, several functions are type issues.

Suggestion:

To ensure consistency and prevent potential issues related to type mismatches, it's recommended to align the Rust implementation with the Solidity interface by using a type that matches uint256.

Resolution:

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

AMA-1 `Initialize` Could Be Front-Run

Severity: Medium

Discovery Methods:

Status: Fixed

Code Location:

contracts/AssetsManager.sol#73-80

Descriptions:

In the contract, by calling the `initialize` function to initialize the contracts, there is a potential issue that malicious attackers preemptively call the initialize function to initialize and there is no access control verification for the initialize functions.

Suggestion:

It is suggested that the `initialize` function can be called only by privileged addresses or in the same transaction immediately after the contract is created to avoid being maliciously called by the attacker.

Resolution:

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

AMA-2 Inconsistent Reentrancy Guard Implementation in Token Exchange Functions

Severity: Minor

Discovery Methods:

Status: Fixed

Code Location:

contracts/AssetsManager.sol#123-147

Descriptions:

Several functions in the contract implement `reentrancy` protection using the `nonReentrant` modifier, while others lack this safeguard. Specifically, `exchangeVToTToken` and `exchangeTToVToken` are protected against reentrancy attacks, but functions such as `exchangeTgasFromTToken`, `exchangeTgasFromVToken`, and `exchangeTgasToVToken` do not have the nonReentrant modifier applied.

Suggestion:

Ensure consistent use of reentrancy protection across all externally callable functions that modify critical state variables or handle token transfers. Apply the nonReentrant modifier to all relevant functions to maintain a uniform security posture and reduce the risk of reentrancy-based exploits.

Resolution:

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

AMA-3 Missing Event Emission

Severity: Minor

Discovery Methods:

Status: Fixed

Code Location:

contracts/AssetsManager.sol#258-315

Descriptions:

The functions `stakingBondAndValidate` and `stakingBondExtraAndValidate` do not emit any events upon successful execution, despite performing critical actions such as exchanging tokens and interacting with the staking module via delegatecall. This lack of event emission reduces transparency and makes it difficult for off-chain systems, wallets, or explorers to track user actions or provide proper feedback.

Suggestion:

Add explicit event emissions at the end of both functions to log key actions.

Resolution:

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

IMP-1 `converted_payout` May Exceed Total Issuance Cap

Severity: Medium

Discovery Methods: Manual Review

Status: Fixed

Code Location:

pallets/staking/src/pallet/impls.rs#389

Descriptions:

In the `end_era` function, the validator payout is initially calculated using :

```
        let (mut validator_payout, remainder) =
            T::EraPayout::era_payout(staked, issuance, era_duration);

        let mut converted_payout: u128 = validator_payout.saturated_into();
        let ratio = Self::rewards_ratio();
        if ratio.1 != 0 {
            converted_payout =
converted_payout.saturating_mul(ratio.0).saturating_div(ratio.1);
        } else {
            converted_payout = 0;
        }
```

`era_payout` which enforces a total issuance cap (RELEASE_LIMIT) to prevent minting beyond the allowed limit.

However, the result of `era_payout` is subsequently scaled by a configurable reward ratio:

```
converted_payout = converted_payout.saturating_mul(ratio.0).saturating_div(ratio.1);
```

This scaling happens after the issuance cap has been applied, meaning that if the ratio is greater than 1, it can inflate the totle payout amount beyond the `RELEASE_LIMIT`.

Suggestion:

validate the capped limit after applying the ratio in `end_era`.

Resolution:

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

VET-1 setMinter Does Not Update Whitelist When Changing Minter Address

Severity: Minor

Discovery Methods: Manual Review

Status: Fixed

Code Location:

contracts/VETaker.sol#80

Descriptions:

The `setMinter` function allows the contract owner to update the minter address:

```
function setMinter(address _minter) external onlyOwner {
    minter = _minter;
}
```

However, this function does not update any associated whitelist.

Specifically, it does not remove the previous minter from the whitelist, nor does it add the new minter address to the whitelist.

Suggestion:

Modify the setMinter function to update the whitelist accordingly.

Resolution:

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

VET-2 Redundant `whenLive` Check in `burn` Function

Severity: Minor

Discovery Methods:

Status: Fixed

Code Location:

contracts/VETaker.sol#67-71

Descriptions:

In the `burn` function, there is a check to ensure that the caller is the `minter` (`require(msg.sender == minter, "VETaker: only minter");`) and an additional modifier `whenLive` is applied to the function. The `whenLive` modifier presumably checks if the contract is operational or live. However, given that the primary security check is already ensuring that only the designated `minter` can call this function, applying the `whenLive` check may be considered redundant for this specific scenario.

Suggestion:

To avoid redundancy and streamline the function's execution flow, consider removing the `whenLive` modifier from the `burn` function.

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.