DeagentAI
Audit Report

Mon Dec 08 2025

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

ScaleBit

DeagentAI Audit Report

1 Executive Summary

1.1 Project Information

Description	The SwapQuota contract is a centralized token exchange system designed to facilitate controlled swaps between two specific ERC20 tokens.
Type	Infrastructure, SocialFi
Auditors	fishmen,N0n3
Timeline	Wed Dec 03 2025 - Wed Dec 03 2025
Languages	Solidity
Platform	ETH ARB POL BSC
Methods	Architecture Review, Unit Testing, Manual Review

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
SQU	SwapQuota.sol	79a15dc5eb271e1d48135bbb7b3a4005161b13a3

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	3	2	1
Centralization	1	0	1
Critical	0	0	0
Major	1	1	0
Medium	0	0	0
Minor	1	1	0
Informational	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):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Joe to identify any potential issues and vulnerabilities in the source code of the DeagentAI smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 3 issues of varying severity, listed below.

ID	Title	Severity	Status
SQU-2	Lack of Balance Validation in Quota Assignment Leads to Management Confusion and Unclear Reverts	Major	Fixed
SQU-3	Centralization Risks	Centralizatio n	Acknowledged
SQU-4	Missing Zero Address Validation	Minor	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the DeagentAI Smart Contract :
The `SwapQuota` contract facilitates a controlled token exchange mechanism where participants can swap a designated input token (`tokenIn`) for an output token (`tokenOut`), subject to pre-assigned limits. The interaction workflow is as follows:

1. **Quota Allocation (Admin Action)** Before a participant can interact with the system, the contract owner must assign a swap quota to the participant's address. This is achieved via the `setQuota` or `setQuotas` functions, which define the maximum amount of `tokenIn` the participant is permitted to swap.

2. **Token Approval** The participant must authorize the `SwapQuota` contract to spend their `tokenIn` tokens. This is a standard ERC20 approval process performed on the `tokenIn` contract, allowing the `SwapQuota` contract to transfer tokens on the participant's behalf.

3. **Swap Execution** The participant initiates the exchange by calling the `swap` function with the desired `amount` . The contract performs the following operations:

    - **Validation**: Verifies that the `amount` is greater than zero and does not exceed the participant's remaining quota.
    - **Quota Update**: Deducts the swap `amount` from the participant's available quota.
    - **Asset Transfer**: Transfers the specified `amount` of `tokenIn` from the participant to the contract and transfers an equivalent `amount` of `tokenOut` from the contract to the participant.

4 Findings

SQU-2 Lack of Balance Validation in Quota Assignment Leads to Management Confusion and Unclear Reverts

Severity: Major

Status: Fixed

Code Location:

SwapQuota.sol

Descriptions:

The `setQuota` and `setQuotas` functions allow the administrator to assign swap quotas to users without verifying their current `tokenIn` balance. This creates a discrepancy between the assigned quota and the user's actual ability to swap, leading to potential confusion in quota management. Furthermore, if a user attempts to swap with insufficient funds, the transaction will revert directly. The contract lacks a mechanism to emit a warning event during quota setting or a clear error message during swapping to inform the user of the insufficient balance.

Suggestion:

It is recommended to check the user's `tokenIn` balance within `setQuota` and `setQuotas`. If the assigned quota exceeds the user's current balance, the contract should emit a warning event (e.g., `QuotaExceedsBalance`) to notify the administrator and the user of the mismatch.

Resolution:

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

SQU-3 Centralization Risks

Severity: Centralization

Status: Acknowledged

Code Location:

SwapQuota.sol

Descriptions:

Several centralization risks were identified in the protocol:

1. **Privileged Access Control**: The `owner` possesses the authority to arbitrarily set and modify user quotas via `setQuota` and `setQuotas`. This centralization of power allows the owner to restrict user access or censor specific transactions at will.
2. **Asset Custody Risk**: The `withdraw` function grants the `owner` the ability to transfer any ERC20 tokens held by the contract to an arbitrary address. This creates a significant trust issue, as it gives the owner full custody over the contract's funds, potentially leading to unauthorized asset transfers or misappropriation of user funds.

Suggestion:

It is recommended to transfer ownership to a multi-signature wallet or a governance contract to distribute trust and reduce the risk of a single point of failure.

SQU-4 Missing Zero Address Validation

Severity: Minor

Status: Fixed

Code Location:

SwapQuota.sol

Descriptions:

The `constructor` initializes `tokenIn` and `tokenOut` addresses, and the `setQuota` function sets quotas for user addresses. However, neither function includes a validation check to ensure that the provided addresses are not the zero address (`address(0)`). If `tokenIn` or `tokenOut` is initialized as the zero address, all subsequent `swap` operations will fail, effectively rendering the contract's core functionality useless. Additionally, setting quotas for the zero address is a meaningless operation that consumes gas.

Suggestion:

It is recommended to add a check `require(address != address(0), "ZERO_ADDR");` in the `constructor` for `_tokenIn` and `_tokenOut` , and in `setQuota` (and `setQuotas`) for the `user` address to prevent accidental zero address usage.

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 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 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.