fuzzland

Aagent.ai Audit Report

Signature: e4981a7b993ddd805b74793d7a849081816ad629c6c755d6afc66d4a915da02c

Aagent.ai Audit Report

Executive Summary

From Jan 8, 2024, to Jan 9, 2024, the Aagentai team engaged Fuzzland to conduct a thorough security audit of their on-chain agent project. The primary objective was identifying and mitigating potential security vulnerabilities, risks, and coding issues to enhance the project's robustness and reliability. Fuzzland conducted this assessment over 2 person days, involving 2 engineers who reviewed the code over a span of 1 day. Employing a multifaceted approach that included static analysis, fuzz testing, formal verification, and manual code review, the Fuzzland team identified 1 issue across different severity levels and categories.

Scope

Project Name	Aagent.ai
Repo	aagent_token_contract
Commit	`1bf6ace248e0d003a2f804e67142fb5eb177b5bc`
Fixed Commit	`cde99ca34513a20138096960021fc8890e2ce4f3`
Language	Solidity
Scope	aagent.sol

Disclaimer

The audit does not ensure that it has identified every security issue in the smart contracts, and it should not be seen as a confirmation that there are no more vulnerabilities. The audit is not exhaustive, and we recommend further independent audits and setting up a public bug bounty program for enhanced security verification of the smart contracts. Additionally, this report should not be interpreted as personal financial advice or recommendations.

Auditing Process

- Static Analysis: We perform static analysis using our internal tools and Slither to identify potential vulnerabilities and coding issues.
- Fuzz Testing: We execute fuzz testing with our internal fuzzers to uncover potential bugs and logic flaws.
- Invariant Development: We convert the project into Foundry project and develop Foundry invariant tests for the project based on the code semantics and documentations.
- Invariant Testing: We run multiple fuzz testing tools, including Foundry and ItyFuzz, to identify violations of invariants we developed.
- Formal Verification: We develop individual tests for critical functions and leverage Halmos to prove the functions in question are not vulnerable.
- Manual Code Review: Our engineers manually review code to identify potential vulnerabilities not captured by previous methods.

Vulnerability Severity

We divide severity into three distinct levels: high, medium, low. This classification helps prioritize the issues identified during the audit based on their potential impact and urgency.

- **High Severity Issues** represent critical vulnerabilities or flaws that pose a significant risk to the system's security, functionality, or performance. These issues can lead to severe consequences such as fund loss, or major service disruptions if not addressed immediately. High severity issues typically require urgent attention and prompt remediation to mitigate potential damage and ensure the system's integrity and reliability.
- **Medium Severity Issues** are significant but not critical vulnerabilities or flaws that can impact the system's security, functionality, or performance. These issues might not pose an immediate threat but have the potential to cause considerable harm if left unaddressed over time. Addressing medium severity issues is important to maintain the overall health and efficiency of the system, though they do not require the same level of urgency as high severity issues.
- **Low Severity Issues** are minor vulnerabilities or flaws that have a limited impact on the system's security, functionality, or performance. These issues generally do not pose a significant risk and can be addressed in the regular maintenance cycle. While low severity issues are not critical, resolving them can help improve the system's overall quality and user experience by preventing the accumulation of minor problems over time.

Below is a summary of the vulnerabilities with their current status, highlighting the number of issues identified in each severity category and their resolution progress.

	Number	Resolved
High Severity Issues	0	0
Medium Severity Issues	0	0
Low Severity Issues	0	0
Info Severity Issues	1	1

Findings

[Info] Centralization risk

1. **Single Point of Failure**: The ownership of the contract is entirely concentrated in one address (the initial owner). If the private key of this address is compromised or lost, an attacker could gain complete control over the contract, including the ability to pause all accounts' transactions, rendering users unable to access their assets.
2. **Lack of Transparency**: The contract does not provide any mechanism for auditing or monitoring the owner's actions. Users cannot verify whether the owner will misuse their privileges.

Recommendations:

Consider using a multi-signature wallet to manage the contract's ownership, reducing the risk of a single point of failure.

Status: Acknowledged