0xCommit

Security Audit Report

Play AI

Version: Final

Date: 20th May 2025

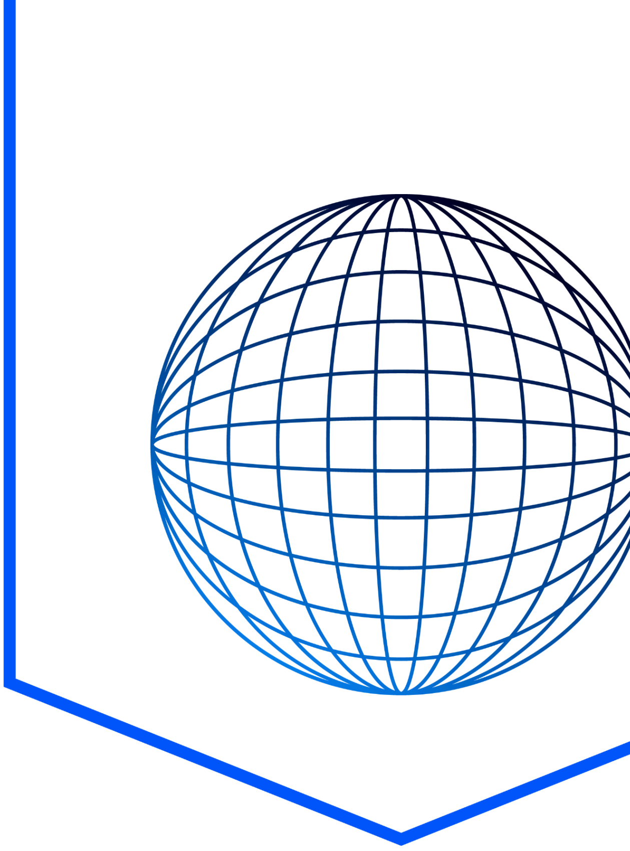

Table of Contents

License

Disclaimer

Introduction

Purpose of this report

0xCommit has been engaged by **Play AI - Web 2 Service** to perform a security audit of its back end code.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine solana program bugs, which might lead to unexpected behaviour.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebases Submitted for the Audit

The audit has been performed on the following GitHub repositories:

Version	Contract Address	Source
Final	Web 2 codebase	https://github.com/PlayAINetwork/hub/blob/27c3006b2d7c2697a8c559eab10ff36de19d5136/src/service/wallet.ts#L61 https://github.com/PlayAINetwork/hub/blob/27c3006b2d7c2697a8c559eab10ff36de19d5136/src/service/chat.ts#L222C49-L222C64

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending**, **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Overview

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base’s intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Summary of Findings

Sr. No.	Description	Severity	Status
1	Hardcoded API Keys and Secrets	Critical	Resolved
2	Missing Rate Limiting	Medium	Acknowledged
3	Insufficient Logging and Monitoring	Medium	Acknowledged
4	Weak Cryptographic Practices	Medium	Acknowledged
5	Information Disclosure in Headers	Low	Resolved
6	Missing Security Headers	Low	Resolved
7	Dependency Vulnerabilities	Low	Resolved

Detailed Findings

Critical Vulnerabilities

🔴 CRIT-001: Hardcoded API Keys and Secrets

Description: Multiple API keys and sensitive credentials are hardcoded in the source code.

Location:

- `hub/src/mcp/evm.ts` (lines 27-29): Hardcoded Moralis API key
- Various MCP files contain embedded credentials

Code Example:

```
await Moralis.start({
   // TODO: Remove test API key
   apiKey:
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJub25jZSI6IjkzYzdhNDQyLTQyMTQtNDEzNy1iY
WE3LTU1NDc3Yzk1Zjc3ZSIsIm9yZ0lkIjoiNDMyNDIwIiwidXNlcklkIjoiNDQ0ODAyIiwidHlwZUlkI
joiNjc5NWQ5YmUtNjhjYy00NGU4LTk2ZWItOTM1N2M5NTg4NDg0IiwidHlwZSI6IlBST0pFQ1
QiLCJpYXQiOjE3Mzk5NjEwNTksImV4cCI6NDg5NTcyMTA1OX0.xKlcGQeeLuy46JsZ9I__JsvI
HcvbOZb8go5I0g_bHUQ"
});
```

Risk: Exposed API keys can lead to unauthorized access to third-party services, potential data breaches, and service abuse.

Remediation:

1. Remove all hardcoded API keys immediately
2. Move all secrets to environment variables
3. Implement proper secret management (AWS Secrets Manager, HashiCorp Vault)
4. Rotate all exposed API keys
5. Add pre-commit hooks to prevent secret commits

🟡 MED-001: Missing Rate Limiting

Description: Critical endpoints lack rate limiting, making them vulnerable to abuse.

Location: Authentication endpoints, file upload endpoints, and API calls

Risk: Denial of service attacks and resource exhaustion.

Remediation:

1. Implement rate limiting on all public endpoints
2. Use exponential backoff for failed attempts
3. Add CAPTCHA for suspicious activity

🟡 MED-002: Insufficient Logging and Monitoring

Description: Security events are not adequately logged for monitoring and incident response.

Location: Authentication failures, authorization violations, and suspicious activities

Risk: Inability to detect and respond to security incidents.

Remediation:

1. Implement comprehensive security logging
2. Add real-time monitoring and alerting
3. Create incident response procedures

Low Severity Issues

🟢 LOW-001: Information Disclosure in Headers

Description: Verbose error responses and debug information in production.

Remediation: Remove debug information and implement proper error handling for production.

🟢 LOW-002: Missing Security Headers

Description: Important security headers are not implemented.

Remediation: Add Content Security Policy, HSTS, and other security headers.

🟢 LOW-003: Dependency Vulnerabilities

Description: Some dependencies may have known vulnerabilities.

Remediation: Regular dependency updates and vulnerability scanning.