BLOCKSEC

Security Audit

Report for FlapAIProvider Contracts

Date: March 18, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Flap.sh
Target	FlapAIProvider Contracts

Version History

Version	Date	Description
1.0	March 18, 2026	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-03-19 11:13:52+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About the Audit Target

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The audit target (hereinafter referred to as the Target) of this audit is the code repository [1] of FlapAIProvider Contracts of Flap.sh.

The FlapAIProvider Contracts of Flap.sh are an upgradeable, role-governed AI oracle protocol for on-chain request handling, fulfillment, and refund settlement. The contract allows users to submit AI reasoning jobs via the function reason(), pay model-based fees, and track each request by ID. The role FULFILLER_ROLE delivers outcomes via the function fulfillReasoning(), including the selected choice and an IPFS CID for reasoning evidence, while failed jobs can be processed via the function refundRequest(). The role DEFAULT_ADMIN_ROLE can register and disable models, set pricing-related limits, tune callback gas, and withdraw undelivered balances, giving the protocol configurable operations.

Note this audit only focuses on the smart contracts in the following directories/files:

- src/plugins/AIProvider/FlapAIProvider.sol

Other files are not within the scope of the audit. Additionally, all dependencies of the Target are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Project	Version	Commit Hash
FlapAIProvider Contracts	`Version 1`	`078daf60170b463758b7342ec3ddc2868bc52149`
	`Version 2`	`3d0614802a06c831943990a0a086c89e0b559cad`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset.

[1] https://github.com/flap-sh/FlapTaxVaults

Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the Target, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of the Target.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan the Target with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of the Target and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency
* Improper use of the proxy system
* Reentrancy
* Denial of Service (DoS)
* Untrusted external call and control flow
* Exception handling
* Data handling and flow
* Events operation
* Error-prone randomness
* Oracle security
* Business logic correctness
* Semantic and functional consistency
* Emergency mechanism

* Economic and incentive impact

1.3.2 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we found **two** potential security issues. Besides, we have **one** recommendation and **one** note.

- Low Risk: 2
- Recommendation: 1
- Note: 1

ID	Severity	Description	Category	Status
1	Low	Potential data truncation risk	Security Issue	Fixed
2	Low	Potential operational risks due to improper disable model ID mechanism	Security Issue	Fixed
3	-	Add a check in the function `withdrawStuckFee()`	Recommendation	Confirmed
4	-	Potential centralization risks	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Potential data truncation risk

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `FlapAIProvider`, the function `reason()` is responsible for submitting an AI reasoning request to the oracle and recording the information corresponding to `requestId`. However, when recording the variable `modelId` corresponding to `requestId`, the function casts it from `uint256` to `uint16`. Therefore, if the value of `modelId` is greater than `type(uint16).max`, this may lead to a risk of data truncation.

```
229   function reason(uint256 modelId, string calldata prompt, uint8 numOfChoices)
230       external
231       payable
232       returns (uint256 requestId)
233   {
234       // 1. Model must be registered.
235       if (!_modelRegistered[modelId]) revert FlapAIProviderModelNotRegistered(modelId);
236
237       // 2. Model must be enabled.
238       Model storage model = _models[modelId];
239       if (!model.enabled) revert FlapAIProviderModelNotEnabled(modelId);
240
241       // 3. Must have at least one choice.
242       if (numOfChoices == 0) revert FlapAIProviderInvalidNumOfChoices(numOfChoices);
243
244       // 4. Prompt length must not exceed the cap.
```

```
245         uint256 promptLen = bytes(prompt).length;
246         if (promptLen > _maxPromptLength) {
247             revert FlapAIProviderPromptExceedsMaxLength(promptLen, _maxPromptLength);
248         }
249
250         // 5. BNB fee must meet the model price.
251         if (msg.value < model.price) revert FlapAIProviderInsufficientFee(msg.value, model.price);
252
253         // Store the request.
254         requestId = _nextRequestId++;
255         _requests[requestId] = Request({
256             consumer: msg.sender,
257             modelId: uint16(modelId),
```

Listing 2.1: src/plugins/AIProvider/FlapAIProvider.sol

```
394     function registerModel(uint256 modelId, string calldata name, uint256 price)
395         external
396         onlyRole(DEFAULT_ADMIN_ROLE)
397     {
398         if (_modelRegistered[modelId]) revert FlapAIProviderModelAlreadyRegistered(modelId);
399
400         _modelRegistered[modelId] = true;
401         _models[modelId] = Model({name: name, price: price, enabled: true});
402
403         emit FlapAIProviderModelRegistered(modelId, name, price);
404     }
```

Listing 2.2: src/plugins/AIProvider/FlapAIProvider.sol

Impact If the value of `modelId` is greater than `type(uint16).max`, this may lead to a risk of data truncation.

Suggestion Revise the code accordingly.

2.1.2 Potential operational risks due to improper disable model ID mechanism

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `FlapAIProvider`, the function `disableModel()` permanently disables a registered model ID (i.e., the model ID cannot be re-enabled or re-registered once disabled). However, multiple vault contracts may interact with this provider and rely on specific model IDs. If a model ID is permanently disabled, any vault depending on that model ID would need to update its configuration accordingly. Since such updates can only be performed by privileged roles (`guardians`), disabling a model ID would require `guardians` to manually locate and update all affected vaults. This may introduce operational overhead and increase the risk of delayed updates or missed configurations.

```
411     function disableModel(uint256 modelId) external onlyRole(DEFAULT_ADMIN_ROLE) {
412         if (!_modelRegistered[modelId]) revert FlapAIProviderModelNotRegistered(modelId);
```

```
413         if (!_models[modelId].enabled) revert FlapAIProviderModelNotEnabled(modelId);
414
415         _models[modelId].enabled = false;
416
417         emit FlapAIProviderModelDisabled(modelId);
418     }
```

Listing 2.3: src/plugins/AIProvider/FlapAIProvider.sol

Impact Permanently disabling model IDs may introduce operational overhead and increase the risk of delayed updates or missed configurations.

Suggestion Revise the code accordingly.

2.2 Recommendation

2.2.1 Add a check in the function `withdrawStuckFee()`

Status Confirmed

Introduced by `Version 1`

Description In the contract `FlapAIProvider`, the function `withdrawStuckFee()` allows the role `DEFAULT_ADMIN_ROLE` to withdraw a specified amount of BNB from the contract. It is recommended to add a check to ensure that the input `amount` is less than or equal to `address(this).balance`.

```
424     function withdrawStuckFee(address payable recipient, uint256 amount) external onlyRole(
            DEFAULT_ADMIN_ROLE) {
425         if (recipient == address(0)) revert FlapAIProviderZeroAddress();
426
427         (bool success,) = recipient.call{value: amount}("");
428         if (!success) revert FlapAIProviderWithdrawFailed();
429
430         emit FlapAIProviderFeesWithdrawn(recipient, amount);
431     }
```

Listing 2.4: src/plugins/AIProvider/FlapAIProvider.sol

Suggestion Add a check in the function `withdrawStuckFee()`.

2.3 Note

2.3.1 Potential centralization risks

Introduced by `Version 1`

Description In this project, several privileged roles (e.g., `DEFAULT_ADMIN_ROLE`) can conduct sensitive operations, which introduces potential centralization risks. For example, the role `DEFAULT_ADMIN_ROLE` can withdraw the contract `FlapAIProvider`'s BNB via the `withdrawStuckFee()` function. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS